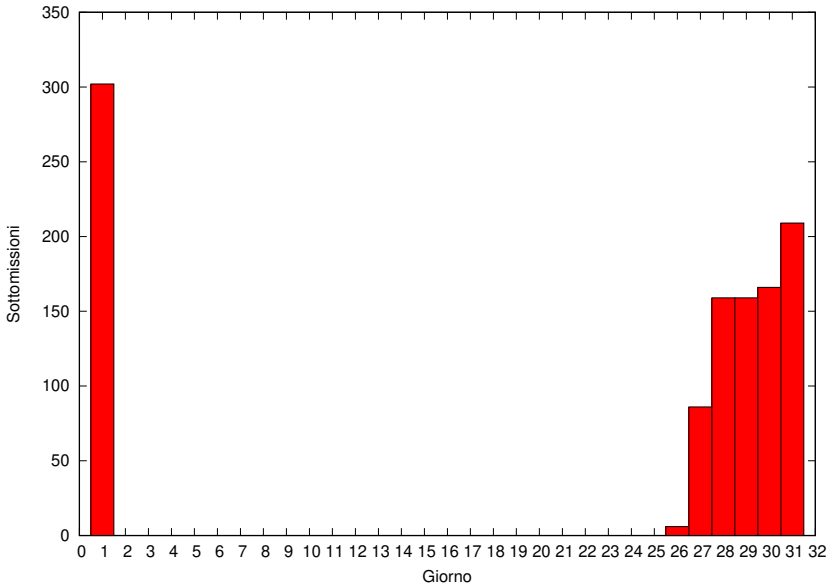
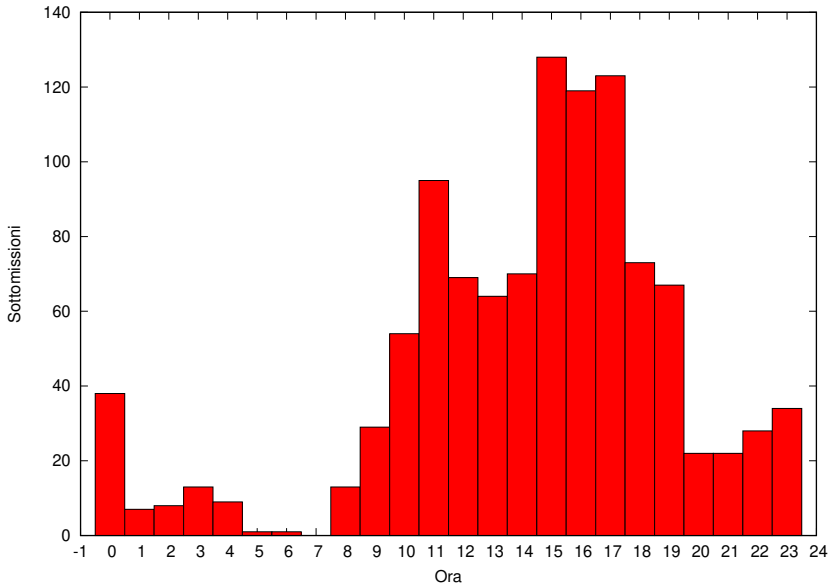
ASD: ENDGAME

Statistiche

Numero sottoposizioni: 1084





- 72 gruppi partecipanti, di cui 46 gruppi hanno ottenuto la sufficienza;
- 173 studenti iscritti, di cui 129 appartenenti a gruppi che hanno ottenuto la sufficienza;

Risultati

Punti bonus (classifica completa sul sito)

- $P < 50 \implies$ Progetto non passato (26 gruppi, 44 studenti)
- $50 \leq P < 80 \implies$ 1 punto bonus (13 gruppi, 32 studenti)
- $80 \leq P < 90 \implies$ 2 punti bonus (11 gruppi, 32 studenti)
- $90 \leq P \leq 92.44 \implies$ 3 punti bonus (6 gruppi, 17 studenti)
- $P > 92.44 \implies$ 3.5 punti bonus (16 gruppi, 45 studenti)

`https://judge.science.unitn.it/slides/asd20/classifica-prog2.pdf`

Il problema I

Riassumiamo brevemente il problema proposto in questo progetto.

- Abbiamo un grafo completo, pesato e indiretto, che rappresenta la nostra mappa di città con le rispettive strade.
- Ogni città custodisce determinate pietre, ognuna rappresentata da un valore di energia e da una massa. In ogni città possiamo raccogliere al massimo una pietra. Ogni tipo diverso di pietra può essere raccolto una ed una sola volta.
- Abbiamo uno zaino in cui raccogliere le pietre. Lo zaino ha una certa capacità che non può essere superata.
- Gli archi del grafo vengono percorsi ad una certa velocità, che decresce proporzionalmente al peso dello zaino.
- Siamo obbligati a visitare tutte le città una ed una sola volta, e a tornare alla base.

Il problema II

L'obiettivo è **massimizzare** il risultato dato da:

$$E(p,t) = G(p) - R \cdot T(p,t)$$

dove $G(p)$ è l'energia raccolta tramite le pietre, R è il tasso di consumo di energia per unità di tempo usato per il trasporto dello zaino e $T(p,t)$ è il tempo impiegato per visitare le città.

Il problema II

Traveling Thief Problem

Il progetto è ispirato ad un problema conosciuto come Traveling Thief Problem. TTP è un problema NP-HARD, quindi trovare la soluzione ottima per questo problema diventa rapidamente intrattabile.

- TTP combina il famoso problema del Commesso Viaggiatore (TSP) con il problema dello zaino 0-1;
- I due problemi sono fortemente interconnessi: la scelta delle pietre da raccogliere condiziona il costo del TSP, in quanto il Commesso Viaggiatore viene rallentato dal peso delle stesse; la scelta del percorso da seguire condiziona la possibilità di poter prendere o meno determinate combinazioni di pietre;
- Non è solo un problema accademico! Nelle applicazioni industriali è molto frequente fronteggiarsi con problemi, già difficili singolarmente, che interagiscono tra di loro.

Complessità

Considerata l'intrattabilità del problema per input abbastanza grandi, siamo obbligati a pensare a soluzioni che riescano ad approssimare al meglio la soluzione ottima, tenendo anche in conto il tempo limitato d'esecuzione (5s per test case).

Verso la soluzione

Prima di passare ad una proposta di soluzione, vediamo qualche idea su come approcciare in generale il problema.

- È sempre cosa buona e giusta partire implementando la soluzione più semplice che vi possa venire in mente: **una baseline**. Ci servirà per capire l'impatto di strategie o tecniche potenzialmente più furbe che andremo ad implementare in futuro, oltre che ad essere sicuri di ottenere i primi punti in classifica.
- **Nonostante i problemi del TSP e dello zaino interagiscano fortemente tra di loro, può essere una buona idea affrontarli separatamente**: definire prima un buon percorso e poi definire che pietre raccogliere lungo il tragitto.
- Questa proposta è ovviamente sub-ottima, ma ricordate che stiamo cercando di approssimare una soluzione ottima, non stiamo puntando a trovarla!

Soluzione I: greedy base

Seguiamo quest'ultima intuizione.

- Costruisco un percorso valido seguendo questa strategia. Partendo dalla base, mi sposto man mano verso la città più vicina alla città in cui mi trovo e che non ho ancora visitato. Una volta visitate tutte le città, torno alla base. Per quanto semplice, questo algoritmo si comporta bene sui casi in cui vale la disuguaglianza triangolare e si comporta discretamente anche in input generali.
- In ogni città posso definire uno score per ogni pietra, ordinare le pietre rispetto a questo score, e prendere la più conveniente. Uno score che combina sia energia sia massa può essere ad esempio il rapporto tra le due.

Questa soluzione garantiva la sufficienza, e ovviamente migliorando le varie euristiche greedy poteva essere portata anche molto in alto a livello di punti.

Osservazioni

Alcune osservazioni:

- Il punteggio viene calcolato in base all'energia finale, che è calcolata tramite Eq. 1.

$$E = G - R \cdot T \tag{1}$$

  Quindi, il punteggio viene deciso fondamentalmente seguendo due metriche: energia totale e tempo trascorso. In alcuni casi, si può trascurare una delle due metriche per favorire quella che contribuisce maggiormente al punteggio.
- Se la capacità del guanto non è molto elevata rispetto alla massa delle pietre, probabilmente si riusciranno a raccogliere poche pietre. Questo significa che possiamo progettare un algoritmo che raccolga pietre solamente verso la fine del percorso, in modo da viaggiare attraverso la maggior parte delle città alla massima velocità.

Soluzione II

Prendiamo in considerazione le osservazioni appena fatte, e seguiamo il seguente algoritmo:

- Decidiamo quali pietre prendere, senza pensare al percorso. Per fare ciò, seguiamo un approccio greedy: ordiniamo le pietre secondo uno **score**, per esempio $\frac{E}{M}$.
- Decidiamo in quali città raccogliere quelle pietre, e creiamo un percorso che **non passa** per queste città.
- Percorriamo il percorso senza raccogliere nessuna pietra, per poi visitare tutte le città in cui vogliamo raccogliere pietre.

Questa soluzione, con alcuni accorgimenti consentiva di arrivare ad oltre **90 punti**.

And now, ladies and gentlemen, the real solutions

Soluzione nome poco creativo

Dividiamo il problema in tre parti distinte, e risolviamo al meglio ciascuna singolarmente.

- Selezione del tour
- Selezione delle pietre
- Abbinamento tra pietre e città

Soluzione nome poco creativo: tour

Il problema è una istanza di TSP asimmetrico.

Generiamo una soluzione iniziale con un algoritmo greedy randomizzato. Scegliamo sempre la città più vicina, e gestiamo gli spareggi casualmente. Ripetiamo più volte e manteniamo la soluzione iniziale migliore.

Ricerca locale: spezziamo il tour in 2 o 3 punti casuali, e consideriamo tutti i modi di ricollegare i pezzi (2-OPT, 3-OPT).

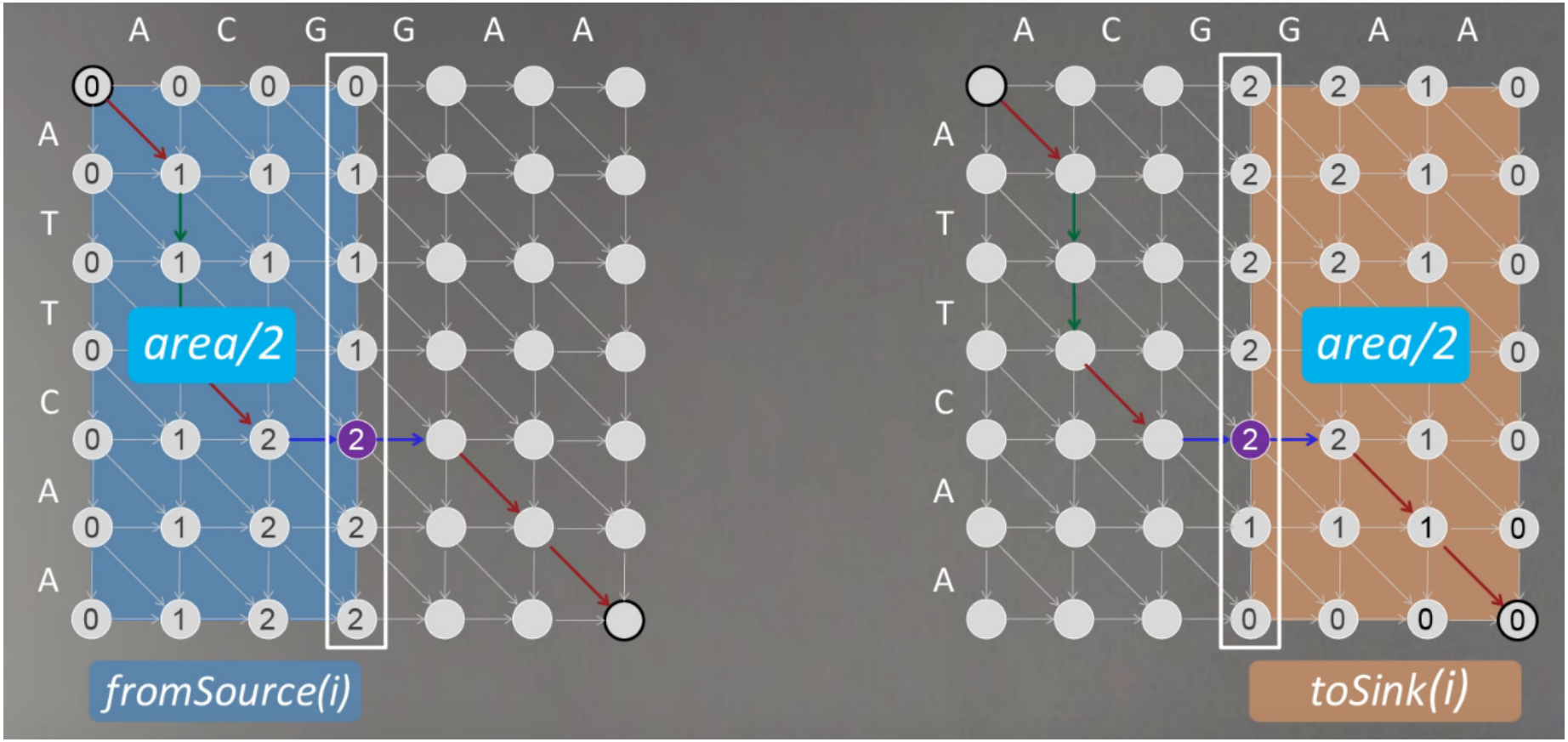
Soluzione nome poco creativo: selezione pietre

Il problema è una istanza di Knapsack binario.

Possiamo risolvere tutti gli input di esempio usando programmazione dinamica.
Per stare nei limiti di memoria, manteniamo solo una riga della tabella DP e ricostruiamo la soluzione con una tecnica divide-et-impera chiamata algoritmo di Hirschberg.

Tempo: $\mathcal{O}(\mathcal{N} \cdot \mathcal{C})$
Memoria: $\mathcal{O}(\mathcal{C})$

Soluzione nome poco creativo: Hirschberg



Soluzione nome poco creativo: abbinamento

Il problema è una istanza di bipartite matching.

Cerchiamo un matching che massimizzi una funzione di profitto, ad esempio la somma delle energie delle pietre abbinate.

Per risolvere questi problemi possiamo usare flussi (Edmonds-Karp, Dinic) oppure algoritmi specializzati (Hopcroft-Karp, Munkres) ..., che richiedono tanta memoria.

Soluzione nome poco creativo: sparsification

Come possiamo eseguire algoritmi di matching complicati quando il grafo è denso, e occupa tanta memoria?

Osservazione: grafi bipartiti con archi casuali “non troppo sparsi” hanno un perfect matching con alta probabilità.

Non ci serve fare calcoli complicati, usiamo questa intuizione per cancellare archi a caso negli input più densi. Accettiamo il basso rischio di peggiorare la qualità del matching, in cambio il grafo occupa molta meno memoria.
Per selezionare gli archi usiamo l’algoritmo Reservoir Sampling.

Soluzione nome poco creativo: all together now

- Troviamo un buon tour approssimando TSP
- Selezioniamo le pietre risolvendo Knapsack
- Abbiniamo le pietre e le città con weighted/unweighted matching

E' utile modificare la funzione di profitto del weighted matching in modo da piazzare le pietre più pesanti alla fine del percorso.
Nel tempo che rimane possiamo simulare scambi casuali tra pietre e città, o spostare pietre in città non occupate.

Punteggio: > 96

Soluzione Mr. Algorithm: Idea

Trovare un percorso da commesso viaggiatore, sapendo già quale pietra raccogliere in ogni città

Struttura

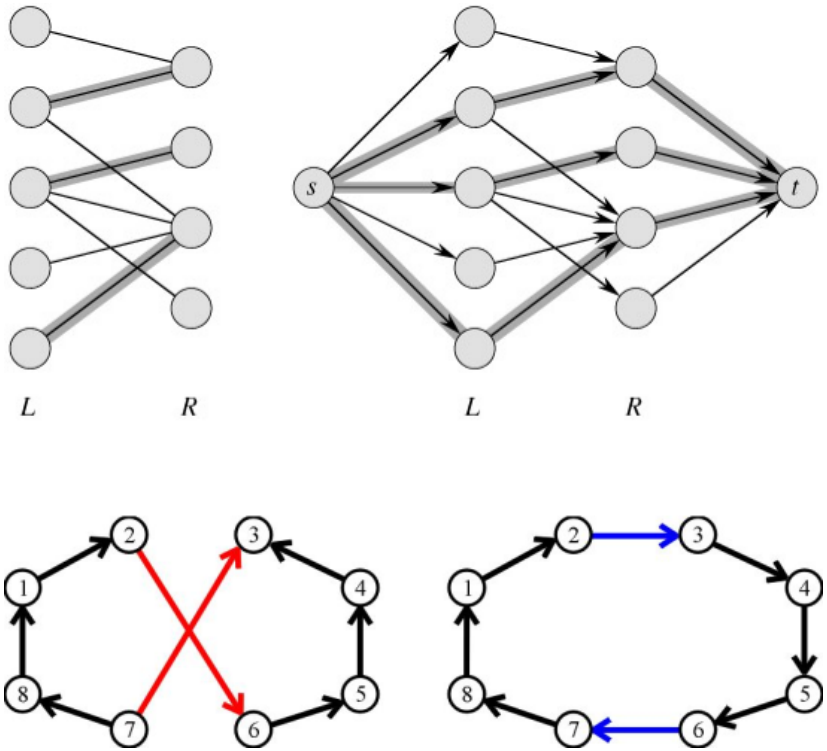
1. Matching bipartito per massimizzare G ed E
   - Zaino approssimato che restituisce le associazioni pietra-città
2. TSP greedy per trovare le prime soluzioni parzialmente ottimizzate
3. TSP con ricerca locale per ottimizzare progressivamente il tempo

Soluzione Mr. Algorithm: Idea

Trovare un percorso da commesso viaggiatore, sapendo già quale pietra raccogliere in ogni città

Algoritmi

- Matching bipartito:
  - Hopcroft-Karp (`matching`)
- Commesso viaggiatore:
  - Nearest neighbour (`TSP`)
  - Ricerca locale (`2-opt`)



Soluzione Mr. Algorithm: Implementazione

1. `matching` con euristiche su e/m per massimizzare G
2. Si esegue la prima `TSP` e si visita il cammino in entrambi i versi
   - Si stampa la prima soluzione
   - Euristica sull'input: termina qui se troppo grande
3. `matching` con euristiche su e/m per massimizzare E
   - Si ripetono `matching` e `TSP` due volte
   - Si stampano progressivamente le soluzioni migliorate
4. Si esegue `2-opt` finché non ci sono più miglioramenti
   - Si stampano progressivamente le soluzioni migliorate
5. Si esegue `drop`
   - Si prova a non prendere alcune pietre per aumentare la velocità

Soluzione Mr. Algorithm: Ottimizzazioni I

Idea: Pre-processare l'input per dare priorità agli oggetti con il rendimento maggiore

Euristiche sull'ordinamento

Matching bipartito ($G^{\uparrow} \vee E^{\uparrow}$)

- Ordina le pietre per valori di e^{ε}/m^{μ} non crescenti, con $(\varepsilon, \mu) \in \{(1,1), (2,1), (1,2), (1,0), (0,1)\}$
- Per massimizzare E si provano solo e/m ed e^2/m

Commesso viaggiatore ($T_{\downarrow}$)

- Ordina le città per masse non decrescenti

Soluzione Mr. Algorithm: Ottimizzazioni II

Euristica sul verso del percorso

Percorso inverso ($T_{\downarrow}$)

- Si prova a percorrere il cammino minimo in senso inverso

Ricerca locale

TSP 2-opt ($T_{\downarrow}$)

- Algoritmo visto a lezione
- Stampa progressivamente le soluzioni con i tempi ridotti

Scartare alcune pietre (`drop`) ($T_{\Downarrow} \wedge G_{\downarrow} \rightarrow E^{\uparrow}$)

- Si ripercorre il cammino, scartando le pietre che consumano più tempo dell'energia che rilasciano

Soluzione Mr. Algorithm: Ottimizzazioni III

Euristica empirica sull'input (spazio)

Se l'input eccede una dimensione specifica ($N \geq 2000, M \geq 6000$):

- Termina prima di eseguire le euristiche e/m per massimizzare E
- In modo da evitare di superare i limiti di memoria

Semplificazione (tempo)

if $(C == 0)$

- Si eseguono solo `TSP` greedy e `2-opt`
- Non si eseguono `matching` (e `drop`)

if $(R == 0)$

- È sufficiente massimizzare G
- Non è necessario massimizzare E ed eseguire `2-opt`

Soluzione IfstreamArrayGlobal

Dividiamo il problema in tre parti:

- Selezione del percorso migliore
- Riflessione sui sassi
- Prendi e porta a casa i sassi

Soluzione IfstreamArrayGlobal: Selezione del percorso migliore

Calcoliamo un TSP del grafo totale.

Generiamo una soluzione iniziale con l'algoritmo greedy che sceglie sempre la città più vicina in base alla distanza.

Cerchiamo di migliorare il percorso utilizzando il metodo di ricerca locale 2-opt: spezziamo la tratta in un punto casuale, la ricolleghiamo e vediamo se migliora.

Soluzione IfstreamArrayGlobal: Riflessione sui sassi: rapporto magico

L'intuizione è di procedere in maniera euristica, introducendo un parametro "furbo".
Per prima cosa percorriamo il percorso nel senso opposto e per ogni nodo calcoliamo la distanza che manca al traguardo, sommando man mano il costo dei vari archi del TSP.

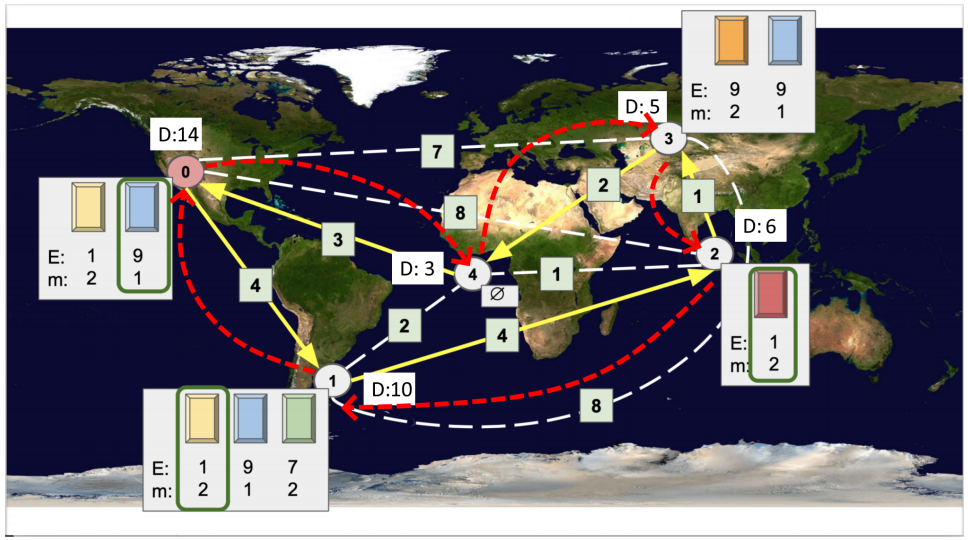


Figura: Calcolo delle distanze su un percorso di esempio

Soluzione IfstreamArrayGlobal: Calcolo del rapporto magico

Calcoliamo poi il parametro...

Rapporto:

$$\frac{E}{m \cdot D}$$

Con

- E: Energia del sasso
- m: Massa del sasso
- D: Distanza tra il sasso e il traguardo

L'idea è la seguente: nel momento in cui raccogliamo un sasso guadagniamo energia E, tuttavia ci tocca "pagare" la sua massa m per tutto il resto del percorso che rimane. Di conseguenza verranno favoriti maggiormente i sassi con molta energia e vicini al traguardo.

Soluzione IfstreamArrayGlobal: Prendi e porta a casa

Un'euristica greedy.

Si procede ordinando tutti i sassi secondo i rapporti appena calcolati e si prova a mettere sul guanto quello con rapporto più alto, se possibile (ovvero non abbiamo nella soluzione un sasso dello stesso tipo o dalla stessa città e la capacità non si è esaurita) e procedendo con i rapporti a seguire. Si prova poi a fare swap randomici sui sassi presi e si vede se c'è un miglioramento.

Il costo di questa operazione è $\mathcal{O}(\mathcal{N} \cdot \mathcal{M})$ nel caso pessimo, perché potremmo dover scorrere ogni sasso per ogni città, anche se nella realtà è più probabile che la selezione termini molto prima.

Soluzione IfstreamArrayGlobal: E per finire... un tocco di magia

A volte i sassi preferiscono essere lasciati in pace e non vogliono che gli si calcoli il rapporto, rimanendo così a -1...

A volte i sassi con più energia (nello specifico quelli con $E > \mathbf{41950}$) tentano di barare e vogliono arrivare primi in classifica. Per farlo moltiplicano il loro rapporto per il numero magico **23**.

Ripetendo euristiche casuali di questo genere, riusciamo ad affinare la soluzione.

Ci ricorderete per sempre come i primi che hanno rotto la barriera del 95.00

Soluzione IfstreamArrayGlobal: Greed is Good

Figura: The real truth

Una Pezza di Pancetta: struttura

Invece di massimizzare E, separiamo il problema in due pezzi distinti: trovare un percorso breve (per abbassare T) e raccogliere pietre che aumentano il più possibile G.

Funzioni rilevanti:

- tspGreedy
- zainoGreedy
- zaino
- output

Una Pezza di Pancetta: tsp e zaino greedy

TSP

Risolviamo il problema del commesso viaggiatore tramite la strategia greedy vista a lezione.
Sperimentalmente l'approssimazione del percorso ottimale data dal tsp si è rivelata molto buona, quindi abbiamo deciso di focalizzarci sull'ottimizzazione della raccolta delle pietre.

ZainoGreedy

Risolviamo il problema della scelta delle pietre nelle varie città due volte, con due diversi criteri greedy:

- Scegliendo quella che massimizza e/m.
- Scegliendo quella che massimizza la differenza tra l'energia della pietra e il costo del suo trasporto fino alla fine del percorso.

Una Pezza di Pancetta: zaino e output

Zaino

Nel caso in cui l'input sia di dimensioni ridotte ($M < 250$), risolviamo il problema dello zaino tramite programmazione dinamica con un dizionario con chiave a tre valori.

Output

La funzione di output riceve il percorso e la lista di pietre e calcola E, G e T.
Ulteriore parametro è *topE*, che contiene il valore migliore ottenuto finora. La stampa avviene solo se si batte il record.

Sfruttando endgame.h, il programma tenta le strategie greedy più volte, percorrendo il percorso in entrambi i versi e stampando la soluzione ogni volta che migliora.

Soluzione "sudoacacia" : Intro

Partiamo col dire che la nostra prima soluzione implementata è analoga a quella descritta da CRICCA.

Separiamo il problema della selezione delle pietre da quella del percorso.

- Percorso: per ogni nodo si seleziona la strada più breve verso un nodo non ancora visitato, fino a tornare alla base.
- Pietre: trovato il percorso lo si percorre raccogliendo per ogni nodo la pietra il cui tipo non è già stato selezionato con il rapporto Energia/Massa.

Di seguito vengono esposte le varie migliorie apportate a questa soluzione di base.

Soluzione "sudoacacia" : Selezione percorso

La scelta del percorso si basa essenzialmente su due tecniche:

- TSP-Greedy: come già spiegato, per ogni nodo si effettua una scelta greedy percorrendo la strada più breve verso un nodo non ancora visitato.
- Random-Path: a partire da S si procede scegliendo una sequenza casuale di nodi nella speranza che tale percorso possa avere un impatto positivo rispetto all'energia finale.

Soluzione "sudoacacia" : Selezione pietre I

Come prima operazione stabiliamo un limite inferiore al valore di E in output decidendo di non raccogliere nessuna pietra sul percorso.

Successivamente utilizziamo Knapsack per ottenere un insieme di pietre ragionevole, eliminando quelle con un rapporto molto basso.

In base al numero di pietre presenti, si opta per una soluzione esatta oppure per una approssimata:

- $M \leq 4500$: sfruttiamo Knapsack 0/1 per ottenere una soluzione esatta, indipendemente dal percorso.
- $M > 4500$: data la lista di pietre ordinata per rapporto Energia/Massa decrescente, si raccolgono pietre finché vi è capacità.

Soluzione "sudoacacia" : Selezione pietre II

Ottenuto da Knapsack l'insieme delle pietre migliori[1] si procede alla raccolta recuperando solo le pietre all'interno di tale lista.

La raccolta viene effettuata seguendo le seguenti strategie:

- Percorso dritto
- Percorso rovescio
- Percorso randomico
- Raccolta con rapporto maggiore

Tutte queste strategie sono eseguite serialmente mantenendo il risultato migliore di volta in volta.

Implementando tutte le strategie appena presentate abbiamo totalizzato **93.93** punti.

Soluzione "sudoacacia" : Ottimizzazioni I

Tra le varie intuizioni avute nel corso del progetto vi è stata anche quella della "scelta posticipata".

Per quanto riguarda la scelta delle pietre, talvolta è conveniente cominciare a raccoglierle più avanti nel percorso, così da diminuire l'energia sprecata in viaggio.

Effettuiamo quindi più scelte posticipate e verifichiamo se si ottengono risultati migliori aumentando di volta in volta il numero di nodi iniziali non considerati.

Tramite questa accortezza siamo riusciti a raggiungere il punteggio di **94.99**.

Soluzione "sudoacacia" : Tuning

Soluzione "sudoacacia" : Tuning

Giunti a questo punto, ad un giorno dalla consegna, con la mente arida di idee ma con grande sete di punti, abbiamo individuato una serie di parametri che ci sembrava potessero influenzare positivamente l'output del nostro programma.

L'idea era buona, tuttavia le varie combinazioni tra i possibili valori si sono rivelate troppe per poter essere testate manualmente da un gruppo di tre persone.

Ed è così che tramite uno script Python multiprocesso e relativamente poche ore di computazioni[2] abbiamo ottenuto la combinazione ottimale di parametri, rispetto al dataset locale, che ci ha permesso poi di arrivare al punteggio di **95.35**.

Soluzione 'TODO: find a good name'

Come già visto, il problema può essere suddiviso in tre sottoproblemi:

1. selezione di un percorso
2. selezione delle pietre
3. assegnazione di tali pietre alle città

Vediamo velocemente le prime due parti, in quanto già ampiamente trattate nelle soluzioni precedenti.

Soluzione 'TODO: find a good name'

Nella prima parte, cerchiamo un percorso di lunghezza minima così da ridurre sicuramente il tempo di percorrenza totale (T). Per far ciò, costruiamo un percorso iniziale partendo dalla sorgente e selezionando gli archi di peso minore. Dopodichè cerchiamo di migliorare la soluzione trovata utilizzando un algoritmo di ricerca locale come `best improving 2-opt move`.

Nella seconda parte, per massimizzare l'energia totale raccolta (G), ordiniamo le pietre per rapporto $\frac{e}{m}$ decrescente e selezioniamo le prime k pietre la cui somma delle masse è $\leq C$. Se $k > n$, ossia se il numero di pietre selezionate è maggiore del numero di città disponibili, selezioniamo soltanto le prime n pietre con maggior energia.

Soluzione 'TODO: find a good name'

Nella parte finale, dobbiamo stabilire a quali città assegnare le pietre selezionate. Poichè aggiungere pietre allo zaino diminuisce la velocità di spostamento, è più efficiente assegnare le pietre più pesanti verso la fine.

Ordiniamo quindi le pietre per masse decrescenti. Per ogni pietra, seguiamo il percorso all'inverso e cerchiamo la prima città libera in cui si trova quella pietra. A questo punto, verifichiamo se sia conveniente o meno prendere la pietra calcolando il delta di energia che si ottiene qualora venga presa e qualora venga scartata. In base a quale dei due Δ è maggiore prendiamo la decisione.

$$\Delta E = G - R \cdot \frac{L}{v_{max} - (M_p + m) \cdot \frac{v_{max} - v_{min}}{C}}$$

Soluzione 'TODO: find a good name'

dove L è la lunghezza del percorso dalla città considerata alla fine. G è l'energia fornita dalla pietra ($G = 0$ se la pietra non viene presa).
Per calcolare la velocità v di spostamento, bisognerebbe sapere quante pietre sono state già state prese fino a quel punto (ossia M_p). Poichè partiamo dalla fine, non abbiamo tale dato e quindi assumiamo che tutte le pietre ancora da prendere verranno prese nelle città ancora da visitare. Ovviamente $m = 0$ per calcolare il Δ nel caso in cui la pietra venga scartata.

Infine, qualora fossero rimaste delle pietre non assegnate (ad esempio, perchè disponibili soltanto in città già occupate), proviamo a riassegnare alcune pietre in modo tale da liberare le città in cui tali pietre sono disponibili. A questo punto, assegniamo a tali città le pietre mancanti.