

Corso di Metodi e Modelli per l'Ottimizzazione Combinatoria - Relazione progetto

Università degli studi di Padova

Anno Accademico 2015/2016

Studente: Giacomo Quadrio

Matricola: 1061566

Sommario

Il progetto del corso di Metodi e Modelli per l'Ottimizzazione Combinatoria consiste nel risolvere un problema che vede coinvolta un'azienda metalmeccanica che produce pannelli forati per la costruzione di quadri elettrici. La foratura di questi è eseguita attraverso un macchinario a controllo numerico dotato di una punta diamantata che, muovendosi sul pannello secondo una sequenza programmata, produce i fori nelle posizioni desiderate. L'obiettivo è quindi quello di individuare la sequenza di foratura ottimale che minimizzi i tempi di produzione, tenendo conto che il tempo necessario per la foratura è lo stesso e costante per tutti i punti. Il cosiddetto Problema del Commesso Viaggiatore, più comunemente noto come TSP (*Travelling Salesman Problem*), in cui è richiesto di trovare il *circuito Hamiltoniano* di costo complessivo minimo dato un grafo $G = (V, E)$ pesato con costi c_{ij} assegnati ad ogni arco (i, j) . Un circuito Hamiltoniano non è altro che un ciclo all'interno del grafo tale che ogni vertice $v \in V$ appartenga al ciclo e venga percorso una sola volta.

Nei problemi di tipo TSP abbiamo poi che i costi degli archi rispettano la cosiddetta *disuguaglianza triangolare* ovvero dati tre punti A, B e C, la distanza tra A e C deve essere al massimo la distanza tra A e B più la distanza tra B e C. Più nel dettaglio il TSP in esame in questo progetto è un *TSP euclideo* o planare cioè nel quale si usano le ordinarie distanze euclidee per i costi degli archi. Nonostante il problema rimanga NP-difficile molte euristiche riescono a lavorare meglio.

1. Modello del problema

Il problema oggetto del progetto può essere formulato come un problema di ottimizzazione su reti di flusso partendo quindi da un grafo $G = (N, A)$. Scegliendo arbitrariamente un nodo di partenza $0 \in N$ impostiamo ad $|N|$ il flusso uscente da esso in modo tale che venga spinto verso altri nodi. Tale operazione ha però dei vincoli ovvero ciascun nodo, eccetto l'origine, riceverà una e una sola unità di flusso, ogni nodo sia visitato una e una sola volta e che il costo del cammino, in termini di pesi c_{ij} , sia minimo.

1.1. Il modello nella Programmazione Lineare Intera

Il problema può essere formalizzato con il seguente modello di programmazione lineare intera. Avremo quindi:

Insiemi

- **N**: nodi del grafo, rappresentano le posizioni dei fori da realizzare
- **A**: insieme degli archi (i, j) con i e $j \in N$. Essi rappresentano il tragitto per spostarsi dal nodo i al nodo j

Parametri

- **c_{ij}** : tempo impiegato per spostarsi dal nodo i al nodo j con i e $j \in N$ e l'arco $(i, j) \in A$.
- **0**: nodo di partenza del cammino $\in N$.

Variabili decisionali

- **x_{ij}** : unità di flusso trasportate da i a j con i e $j \in N$ e l'arco $(i, j) \in A$.
- **y_{ij}** : indica l'utilizzo dell'arco (i, j) , 1 se viene utilizzato, 0 altrimenti. Avremo che i e $j \in N$ e l'arco $(i, j) \in A$.

Vincoli

Il modello, a questo punto, prevede un totale di cinque vincoli differenti che possono essere indicati come segue:

1. Il flusso uscente da x_{0j} deve essere massimo, cioè $|N|$
2. Ogni nodo utilizza al massimo una unità di flusso, tranne il nodo di partenza
3. Ogni nodo ha un solo arco in entrata
4. Ogni nodo ha un solo arco in uscita
5. Se vi è un'unità di flusso trasportata da i a j deve di conseguenza esserci un arco che va da i a j

2. Metaeuristiche scelte per il modello e loro implementazione

Il progetto da svolgere richiesto dal corso di Metodi e Modelli per l'Ottimizzazione Combinatoria prevede la risoluzione del modello di programmazione lineare intera tramite due tecniche ovvero usando il risolutore IBM CPLEX ed una o più metaeuristiche a nostra scelta. Una volta fatto ciò si procederà testando i metodi con delle istanze di prova ed i risultati e statistiche confrontati tra di loro per valutarne le prestazioni.

Nello specifico, il problema in esame è un problema di ricerca di vicinato e consiste nel definire una soluzione iniziale e cercare di migliorarla esplorando un intorno di questa soluzione; quindi i metodi scelti ed utilizzati all'interno del progetto sono due, la Local Search ed il Simulated Annealing. Si tratta di metodi metaeuristici ovvero tecniche generali di schemi algoritmici concepiti indipendentemente dal problema specifico. Tali metodi definiscono delle componenti e le loro iterazioni al fine di pervenire ad una buona soluzione ed esse inoltre devono essere specializzate per i singoli problemi.

2.1. Descrizione delle metaeuristiche

Local Search: si definisce una soluzione iniziale e si cerca di migliorarla esplorando un intorno di questa soluzione. Se l'ottimizzazione sull'intorno della soluzione corrente produce una soluzione migliorante il procedimento viene ripetuto partendo, come soluzione corrente, dalla soluzione appena determinata. L'algoritmo termina quando non è più possibile trovare delle soluzioni miglioranti nell'intorno della soluzione corrente o quando è stata determinata una soluzione con valore della funzione obiettivo uguale a qualche bound.

Simulated Annealing: il concetto su cui si basa questa metodologia prende spunto dal mondo reale ovvero consiste nel simulare il comportamento di un processo termodinamico di ricottura di materiali solidi. Se un materiale solido viene riscaldato oltre il proprio punto di fusione e poi viene raffreddato in modo da riportarlo allo stato solido, le sue proprietà strutturali dipenderanno fortemente dal processo di raffreddamento. In sostanza tale algoritmo simula il cambiamento di energia di un sistema sottoposto a raffreddamento, fino a che non converge ad uno stato solido; ciò permette di cercare soluzioni ammissibili di problemi di ottimizzazione cercando di convergere verso soluzioni ottime.

2.2. Implementazione degli algoritmi

Risolutore IBM CPLEX Local Search

Codice 1: Local Search

```
sol = getInitialSol(random);  
  
while (true){  
    vector<int> newSol = findBestN(sol);  
    if (evaluate(newSol) >= evaluate(sol)){  
        return evaluate(sol);  
    }else{  
        sol = newSol;  
    }  
}
```

Come si può vedere, l'algoritmo di Local Search crea innanzitutto una soluzione iniziale da cui partire tramite la funzione `getInitialSol`, dopodiché esso è stato implementato utilizzando un unico ciclo `while` che opera finché non viene restituito in output un valore. Questo significa che ciò avverrà unicamente quando, dopo aver calcolato una nuova soluzione con `newSol = findBestN(sol)`, il suo valore sarà peggiore del valore della soluzione corrente. Se così non è la soluzione corrente viene aggiornata alla soluzione appena calcolata ed il ciclo continua ad operare fino al punto in cui viene trovata

una soluzione peggiorativa.

Simulated Annealing

Codice 2: Simulated Annealing

```
sol = getInitialSol(random);
n_passi = 100000.0 * n / 5;

while (step < n_passi){
    newSol = getNeigh(sol,2,true);
    float de = evaluate(sol) - evaluate(newSol);
    if (de > 0){
        sol = newSol;
    }else{
        temp = 1-(step/n_passi);
        double prob = exp((de)/temp);
        srand(time(NULL)+for_random);
        for_random = for_random + 1;

        if (prob*100 > (rand()%100) ){
            sol = newSol;
        }
    }
    step ++;
}

return(LocalSearch(sol));
```

Per quanto concerne invece l'algoritmo di Simulated Annealing, anch'esso crea innanzitutto una soluzione iniziale da cui partire tramite la funzione `getInitialSol`, dopodiché troviamo un ciclo `while` principale che opererà finché `step <= n_passi`; da notare che il numero di passi inoltre è dinamico così che cresca al crescere del numero di nodi coinvolti nel problema. In cosa consiste quindi questo algoritmo? Viene calcolata una nuova soluzione attraverso la funzione `getNeigh`, dopodiché verrà calcolata la differenza tra il valore della soluzione corrente e quello della nuova soluzione. Se il delta ottenuto è maggiore di zero significa che la nuova soluzione è migliorativa e quindi aggiornerò di conseguenza `sol`, se invece così non è procederemo in maniera differente rispetto alla Local Search. Andremo infatti a calcolare per prima

cosa la temperatura di raffreddamento, valore che è coinvolto nel calcolo della probabilità di accettare una mossa peggiorativa. Questa probabilità è calcolata come segue:

$$\text{prob} = \exp(-\delta/t)$$

dove δ è l'entità del peggioramento e t è la temperatura temp di raffreddamento. Nel caso la probabilità prob sia maggiore di un numero random calcolato attraverso la funzione rand ciò comporterà appunto l'accettare la mossa peggiorativa, altrimenti essa verrà scartata. Come si modifica però la probabilità p? Essa diminuisce al crescere del peggioramento indotto dalla mossa stessa e cresce al crescere della temperatura t di processo.

Al termine delle operazioni eseguite tramite il ciclo while andremo ad effettuare infine una **fase di intensificazione** tramite l'operazione di Local Search eseguita sulla soluzione migliore calcolata in precedenza. Il motivo di ciò è che la Local Search è una tecnica relativamente economica in quanto a tempi di esecuzione e permette di raffinare ulteriormente la soluzione trovata.

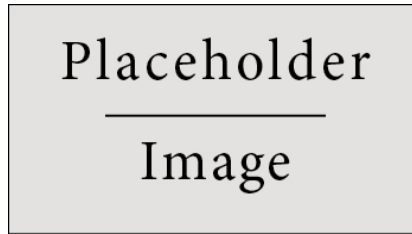


Figura 1: Figure caption

$$E = mc^2 \tag{1}$$

3. Descrizione dei test effettuati

Per verificare le prestazioni degli algoritmi utilizzati nel progetto sono stati condotti diversi test con un numero ben definito di istanze per differenti tipologie di dataset. Nel dettaglio sono stati utilizzati quattro diversi dataset in cui i punti sono disposti come segue:

- **Distribuzione uniforme** ovvero uniformemente disposti all'interno dello spazio

- In **cluster** e disposti in tre raggruppamenti
- In **circoli** e disposti in tre circonferenze
- In **linea** e disposti in tre fasce

Per ciascuna tipologia di dataset sono stati poi selezionati 10 raggruppamenti composti da 4 istanze, ognuno con un numero di nodi che va da 10 a 100 ed in cui la differenza della quantità di elementi tra un gruppo ed il successivo è pari a 10. Le istanze utilizzate sono in realtà matrici delle adiacenze M di dimensioni $n \times n$ tali che nella cella m_{ij} vi è la distanza tra il vertice v_i ed il vertice v_j

I test condotti per i vari algoritmi sono stati effettuati utilizzando, di volta in volta, sempre gli stessi dataset così da poter confrontare meglio i risultati. Ogni istanza di ogni raggruppamento di ogni dataset è stata infine data in pasto agli algoritmi per un totale di 10 volte ed i risultati registrati ritenuti interessanti per la valutazione sono stati vari quali il tempo medio e totale impiegato, il valore dell'ottimo, le soluzioni migliori, la varianza e via dicendo.

4. Descrizione dei risultati ottenuti

I risultati completi ottenuti dall'esecuzione degli algoritmi sono stati riportati all'interno dei file Excel denominati "cplex_results", "local_search_results", "sim_annealing_results" e non quindi direttamente nella relazione a causa della mole di dati che comprendono. In questa sede compariranno quindi solo risultati riassuntivi utili a meglio comprendere le conclusioni a cui si è giunti o, eventualmente, rimandi ai file sopraccitati.