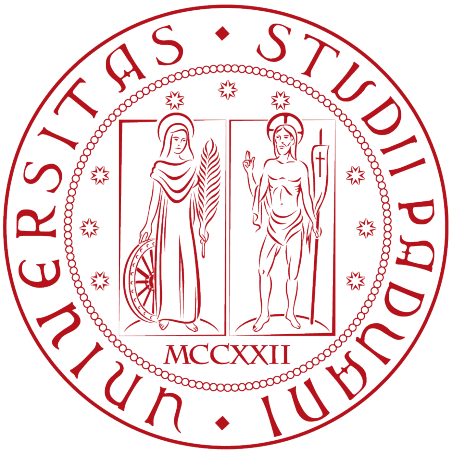
Università degli Studi di Padova

Dipartimento di Matematica

Corso di Laurea Magistrale in Informatica



Relazione riguardo le esercitazioni di laboratorio

Autore

Giacomo Manzoli 1130822

Anno Accademico 2016-2017

Indice

1 Introduzione

L'obiettivo della prima parte del progetto era quello di implementare un modello in CPLEX e di provarlo in modo da trovare la dimensione massima del problema che permette una risoluzione esatta entro un certo limite di tempo, mentre la seconda parte comprendeva l'implementazione di un algoritmo meta-euristico ad-hoc per il problema e di confrontarlo con CPLEX.

La sezione §2 contiene la descrizione dell'implementazione del modello CPLEX, la sezione §3 contiene la descrizione dell'algoritmo genetico implementato, mentre il confronto tra i due approcci è descritto nella sezione §4

1.1 Generazione delle istanze

Prima di implementare i vari algoritmi è stato necessario creare delle istanze per il problema. È stato quindi creato uno script in Python in grado di generare delle istanze casuali a partire da un numero di nodi o fori.

Dato che il problema ha delle caratteristiche specifiche, ovvero visto che si tratta di schede perforate, è ragionevole assumere che i fori seguano un certo pattern. Pertanto lo script è stato sviluppato in modo che possa generare anche delle istanze pseudo-casuali, ovvero delle istanze in cui ci sono blocchi di punti che compaiono vicini tra loro, raggruppati in rettangoli e a coppie.

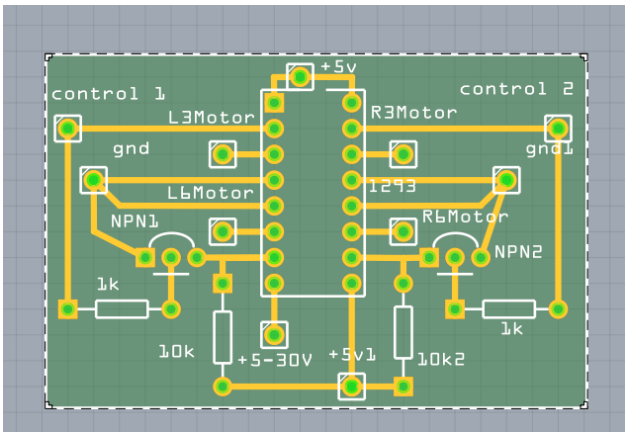


Figura 1: Esempio di scheda perforata presa come riferimento per la generazione delle istanze pseudo-casuali.

2 CPLEX

Il modello CPLEX è stato implementato come specificato nella consegna della prima esercitazione con dei particolari accorgimenti per rendere il codice più comprensibile e mantenibile:

- Le informazioni relative al problema e ad una sua possibile soluzione sono state modellate con due classi `Problem` e `Solution`. Inoltre, tutta la logica di risoluzione è stata incapsulata nella classe `CPLEXSolver`.
- Durante la dichiarazione delle variabili viene costruita una mappa di supporto per rendere più agevole l'utilizzo delle variabili all'interno dei vincoli. Viene inoltre limitato il numero di variabili, evitando di definire le variabili x_{ii} e y_{ii}.
- I vincoli vengono definiti uno alla volta, in modo da semplificare la sintassi di definizione.

2.1 Definizione delle variabili

Le variabili in CPLEX, una volta create, vengono memorizzate in sequenza all'interno di un array interno del risolutore e l'unico modo per riferirsi ad una variabile è mediante la sua posizione nell'array interno.

Per riferirsi più facilmente alle variabili, viene quindi creata una matrice di dimensione $N \times N$ che associa il "*nome della variabile*" alla sua posizione interna nel risolutore.

```
1  // xMap[i][j] è una matrice N x N
2  for (int i = 0; i < N; ++i) {
3    for (int j = 0; j < N; ++j) {
4      if (i == j) continue;
5      char htype = 'I';
6      double obj = 0.0;
7      double lb = 0.0;
8      double ub = CPX_INFBOUND;
9      snprintf(name, NAME_SIZE, "x_%d,%d", nodes[i], nodes[j]);
10     char* xname = &name[0];
11     CHECKED_CPX_CALL( CPXnewcols, env, lp, 1, &obj, &lb, &ub, &htype, &xname );
12     xMap[i][j] = createdVars;
13     createdVars++;
14   }
15 }
```

Codice 1: Creazione delle variabili x_{ij}

La mappatura del nome viene fatta nella riga 12 del frammento: `createdVars` è una variabile che tiene traccia del numero di variabili che sono state create nel risolutore e quindi la prossima variabile aggiunta avrà come indice interno il valore di `createdVars`. Il valore dell'indice viene quindi memorizzato nella mappa e poi incrementato, in modo che alla successiva iterazione dei ciclo, questo sia ancora corretto.

Nello stesso frammento di codice è possibile osservare come **non** vengano create le variabili x_{ii}, questo perché quando i due indici sono uguali, l'`if` in riga 4 blocca l'esecuzione del corpo. Questa scelta è stata fatta perché tale variabile non è significativa per il problema, in quanto scegliere di spostare la trivella lungo l'arco (i, i) equivale al lasciare ferma la trivella.

Quanto riportato è stato effettuato anche per le variabili y_{ij}.

2.2 Definizione dei vincoli

CPLEX permette di definire più vincoli con una sola istruzione, tuttavia la notazione per sfruttare questa possibilità è poco pratica da usare, in quanto richiede che gli indici delle variabili e i corrispettivi coefficienti vengano passati come una matrice sparsa, linearizzata in un vettore. Se invece viene creato un vincolo alla volta, non c'è la necessità di gestire la matrice sparsa, in quanto questa è composta da una sola riga e quindi può essere considerata come vettore. Detto in altre parole, non è necessario utilizzare il vettore `rmatbeg` per tenere traccia dell'inizio delle varie righe, dato che essendoci un'unica riga, questa inizierà alla posizione 0 degli array utilizzati per definire il vincolo.

```
1  // Vincoli sul flusso in ingresso
2  for (int j = 0; j < N; ++j){
3    std::vector<int> varIndex(N-1);
4    std::vector<double> coef(N-1);
5    int idx = 0;
6    // Recupero gli indici dalla mappa delle varibaili
7    for (int i = 0; i < N; ++i) {
8      if (i==j) continue;
9      varIndex[idx] = yMap[i][j];
10     coef[idx] = 1;
11     idx++;
```

```
12    }
13    char sense = 'E';
14    double rhs = 1;
15    snprintf(name, NAME_SIZE, "in_%d",j+1);
16    char* cname = (char*)(&name[0]);
17
18    int matbeg = 0;
19    CHECKED_CPX_CALL( CPXaddrows, env, lp,
20              0, // Numero di variabili da creare
21              1, // Numero di vincoli da creare
22              varIndex.size(), // Numero di variabili nel vincolo con coeff != 0
23              &rhs, // Parte destra del vincolo
24              &sense, // Senso
25              &matbeg, // 0 perché creo un solo vincolo
26              &varIndex[0], // Inizio dell'array con gli indici delle variabili
27              &coef[0], // Inizio dell'array con i coefficienti delle variabili
28              NULL,  // Nomi per le nuove variabili
29              &cname // Nome del vincolo
30            );
31  }
```

Codice 2: Esempio di creazione di una serie di vincoli

Dal codice sopra riportato si può osservare come la chiamata della riga 19 definisce solamente un vincolo, pertanto per generare tutti vincoli del tipo

$$\sum_{j:(i,j)\in A} y_{ij} = 1 \quad \forall j \in N$$

è necessario effettuare un iterazione esterna con il ciclo `for` di riga 2.

Questo modo di definire i vincoli potrebbe essere leggermente meno efficiente, dato che effettua più chiamate ai metodi del risolutore, ma l'impatto sulle prestazioni rimane comunque basso, dato che i vincoli vengono creati solamente all'inizializzazione del modello e il tempo necessario è molto inferiore rispetto a quello necessario per l'ottimizzazione.

3 Algoritmo Genetico

Come meta-euristica ad-hoc si è scelto di implementare un algoritmo genetico seguendo le indicazioni presenti nelle dispense del corso.

3.1 Scelte progettuali

Gli algoritmi genetici lasciano molte possibilità di scelta al progettatore e le scelte effettuate influenzano notevolmente l'efficacia dell'algoritmo.

Nel determinare i vari componenti si è cercato di progettare un algoritmo bilanciato, che parta da delle soluzioni buone, ma che converga lentamente grazie alle mutazioni e alla selezione di Montecarlo.

3.1.1 Codifica delle soluzioni

Per la codifica delle soluzioni si è scelto di adottare la stessa utilizzata per CPLEX. Viene quindi utilizzato un array di lunghezza $N+1$, dove N è il numero di nodi da visitare o fori da effettuare, e rappresenta la sequenza di visita. La dimensione dell'array è di $N+1$ perché viene aggiunto un ultimo elemento sempre fisso a 0, per imporre il vincolo che la trivella ritorni al punto di partenza. Allo stesso modo è imposto il vincolo che il primo elemento dell'array sia 0, in modo che nodo di partenza sia sempre quello e che coincida con il nodo finale.

3.1.2 Generazione della popolazione iniziale

La popolazione iniziale viene creata con delle soluzioni generate in modo pseudo-greedy. Ovvero, ogni soluzione viene generata incrementalmente a partire dal nodo di partenza, andando a scegliere come nodo successivo un nodo qualsiasi tra quelli ancora da visitare. La scelta del nodo viene fatta a caso, dando una maggiore probabilità di essere scelti ai nodi migliori

3.1.3 Funzione di fitness

Come funzione di fitness per gli individui è stata utilizzata la funzione obiettivo, ovvero il costo del cammino descritto dalla soluzione.

3.1.4 Operatore di selezione

La selezione delle soluzioni da riprodurre viene fatta secondo un *torneo-K*. Vengono scelti casualmente dalla popolazione K individui, con $K = \text{POP\_SIZE}/10$ e tra questi viene scelto il miglior candidato per partecipare alla riproduzione. Il processo viene quindi eseguito due volte in modo da scegliere i due genitori.

3.1.5 Crossover

Il crossover viene effettuato in modo uniforme utilizzando i due genitori precedentemente scelti, dando maggior probabilità di esser trasmessi ai geni del genitore migliore. La combinazione dei geni viene effettuata costruendo un nuovo cammino a partire dagli archi presenti nei cammini dei due genitori.

Sia $succ(x, G)$ il nodo successivo al nodo x nel cammino della soluzione G, pertanto nella soluzione G sarà presente l'arco $(x, succ(x, G))$ e siano G_1 e G_2 i due genitori della nuova soluzione.

Si ha quindi che la costruzione del nuovo cammino partirà dal nodo 0 e pertanto il secondo nodo del cammino verrà scelto casualmente tra $succ(0, G_1)$ e $succ(0, G_2)$. Al passo successivo, l'ultimo nodo inserito nel nuovo cammino sarà un nodo y e pertanto la scelta del nodo su cui spostarsi sarà tra $succ(y, G_1)$ o $succ(y, G_2)$. Il procedimento viene ripetuto finché non sarà completato il ciclo, tornando al nodo 0.

Durante la costruzione del figlio possono capitare alcuni casi particolari:

- Uno dei due possibili successori è già presente nel cammino. In questo caso viene scelto l'altro.
- Entrambi i nodi fanno già parte del cammino. Il questo caso come successore viene scelto il nodo migliore.
- Il nodo finale del cammino deve essere il nodo 0. Quindi l'ultimo arco viene scelto forzatamente in modo che sia verso il nodo 0.

Da notare che per come sono gestiti questi casi particolari non possono essere generate soluzioni non valide.

3.1.6 Mutazione

È stata prevista la possibilità che durante l'evoluzione delle popolazione, alcune soluzioni subiscano una mutazione.

Una mutazione consiste nel rimescolare l'ordine di visita dei nodi interni del cammino. In questo modo la mutazione viene fatta velocemente e non invalida la soluzione, perché il primo e l'ultimo nodo vistato sarà sempre il nodo 0.

3.1.7 Sostituzione della popolazione

La sostituzione delle popolazione viene effettuata generando prima un numero R di individui proporzionale alla dimensione della popolazione. Dopodiché la popolazione viene riportata alla dimensione di partenza, selezionando con il metodo di Montecarlo N soluzioni tra le $N + R$ disponibili.

3.1.8 Criterio di stop

Come criterio d'arresto è stato utilizzato un time limit che può essere specificato all'avvio dell'algoritmo.

La scelta è ricaduta su questa condizione d'arresto perché così risulta più semplice effettuare il confronto con CPLEX e perché l'altro criterio provato, ovvero fermare l'algoritmo dopo k iterazioni che non hanno migliorato la miglior soluzione, richiedeva troppo tempo d'esecuzione a causa del metodo di sostituzione della popolazione. Infatti, tra un'iterazione e l'altra può essere scartata anche la soluzione migliore e quindi l'iterazione successiva può risultare migliorativa anche se in realtà non lo è.

L'altro criterio d'arresto preso in considerazione è stato un limite sulle iterazioni, ma facendo le varie prove si è osservato che è preferibile impostare un tempo limite fisso piuttosto che il numero massimo di iterazioni.

3.2 Parametri dell'algoritmo e processo di ottimizzazione

L'algoritmo così implementato richiede che siano specificati i seguenti parametri:

- `POPULATION_SIZE`: dimensione della popolazione;
- `MUTATION_RATE`: probabilità di mutazione:
- `GROWTH_RATIO`: soluzioni generate ad ogni iterazione;
- `TIME_LIMIT`: tempo limite per l'esecuzione dell'algoritmo.

3.2.1 Esperimenti per l'ottimizzazione dei parametri

Prima di confrontare l'approccio genetico con CPLEX è stata eseguita una leggera ottimizzazione dei parametri.

Si sono quindi provate le varie possibili combinazioni per la risoluzione di varie istanze di dimensioni diverse (50, 100, 150 punti) e generate sia in modo casuale che pseudo-casuale. Le varie prove sono state poi ripetute più volte ed è stata effettuata una media dei risultati.

I valori per i parametri che sono stati provati sono:

- `POPULATION_SIZE`: 100, 250, 500;
- `MUTATION_RATE`: 0.01, 0.05, 0.1;
- `GROWTH_RATIO`: 1.1, 1.5, 2;
- `TIME_LIMIT`: 1 minuto.

4 Confronto