

Università degli Studi di Padova

DIPARTIMENTO DI MATEMATICA

CORSO DI LAUREA MAGISTRALE IN INFORMATICA



Relazione esercitazioni laboratorio

Autore

Giacomo Manzoli 1130822

Indice

1	Introduzione	2
1.1	Generazione delle istanze	2
2	CPLEX	2
2.1	Definizione delle variabili	3
2.2	Definizione dei vincoli	3
2.3	Test del modello	4
3	Algoritmo Genetico	6
3.1	Scelte progettuali	6
3.1.1	Codifica delle soluzioni	6
3.1.2	Generazione della popolazione iniziale	6
3.1.3	Funzione di fitness	7
3.1.4	Operatore di selezione	7
3.1.5	Crossover	7
3.1.6	Mutazione	7
3.1.7	Sostituzione della popolazione	7
3.1.8	Criterio di stop	8
3.2	Parametri dell'algoritmo e processo di ottimizzazione	8
3.2.1	Esperimenti per l'ottimizzazione dei parametri	8
3.2.2	Risultati	9
3.3	Test dell'algoritmo	9
4	Confronto	9
A	Informazioni sul codice e sulla consegna	9
A.1	Possibili problemi di compilazione	10

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 con i relativi test, mentre la sezione §3 contiene la descrizione dell'algoritmo genetico implementato e un confronto delle prestazioni rispetto a quelle ottenute da CPLEX.

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.

In entrambi i casi, una volta generati i punti, le distanze sono state calcolate utilizzando la distanza euclidea.

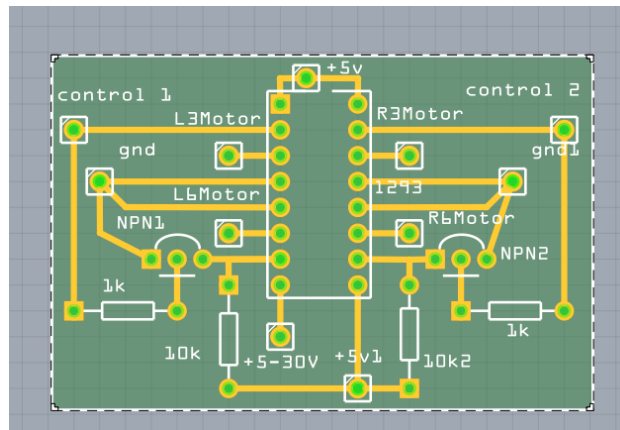


Figura 1: Esempio di scheda perforata presa come riferimento per la generazione delle istanze pseudo-casuali.

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 del 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 corrispondenti 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 variabili
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    }
```

```

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.

2.3 Test del modello

È stato richiesto di osservare qual'è la dimensione massima di un problema che il modello CPLEX riesce a risolvere entro un certo periodo di tempo.

Per fare ciò sono state prima generate 5 istanze casuali e 5 pseudo casuali per ogni dimensione 5, 10, ..., 100 ed è stato eseguito CPLEX con un time limit di 10 minuti. Sono stati poi presi in considerazione 4 livelli di time limit: un secondo, dieci secondi, un minuto e dieci minuti ed è stato osservato qual'è la massima dimensione che può essere risolta entro tali intervalli. I risultati medi delle esecuzioni sulle istanze di varie dimensioni sono riportati nella tabella 1 e riassunti nel grafico 2, mentre la corrispondenza time limit/difficoltà massima è riassunta nella tabella 2.

Dai risultati ottenuti si può osservare che:

- Le istanze generate casualmente sembrano essere mediamente più facili da risolvere rispetto a quelle generate in modo pseudo casuale.
- Per quanto riguarda le istanze pseudo casuali, già con $N = 40$ è stata trovata un'istanza che CPLEX non riesce a risolvere entro 10 minuti d'esecuzione.
- Con $N = 100$ solo due istanze di quelle generate casualmente sono state risolte, mentre di quelle pseudo casuali non ne è stata risolta nessuna. Pertanto sembra che la distribuzione dei punti sia notevolmente influente sulle prestazioni dell'algoritmo.

Dimensione	Pseudo				Random			
	Tempo medio	Tempo minimo	Tempo massimo	Fallimenti	Tempo medio	Tempo minimo	Tempo massimo	Fallimenti
5	0,02	0,01	0,04	0	0,02	0,01	0,04	0
10	0,08	0,04	0,14	0	0,06	0,03	0,08	0
15	0,14	0,06	0,23	0	0,20	0,09	0,31	0
20	0,82	0,10	1,88	0	0,46	0,28	0,65	0
25	1,38	0,17	3,88	0	0,92	0,43	1,57	0
30	3,51	0,16	6,80	0	2,03	0,88	4,27	0
35	4,75	0,52	11,94	0	3,87	1,47	6,87	0
40	129,16	0,03	600,04	1	6,53	2,32	9,03	0
45	31,50	5,01	94,31	0	10,22	9,11	12,89	0
50	114,99	18,88	350,22	0	18,16	13,72	24,37	0
55	36,53	26,39	47,00	0	24,98	15,83	53,49	0
60	52,91	28,03	77,75	0	42,82	23,03	73,45	0
65	215,51	55,89	600,08	1	81,70	62,26	119,60	0
70	196,50	60,60	600,08	1	62,39	35,62	91,72	0
75	239,86	6,97	411,80	0	90,25	48,40	114,86	0
80	318,38	57,05	600,10	2	203,84	89,87	399,65	0
85	245,64	93,26	600,14	1	268,20	69,19	550,24	0
90	474,56	207,08	600,17	3	311,61	155,87	600,13	1
95	517,15	185,08	600,20	4	445,41	238,19	600,20	1

Tabella 1: Risultati medi dell'esecuzione del risolutore CPLEX su 5 istanze diverse per ogni dimensione. Tutti i tempi riportati nella tabella sono espressi in secondi.

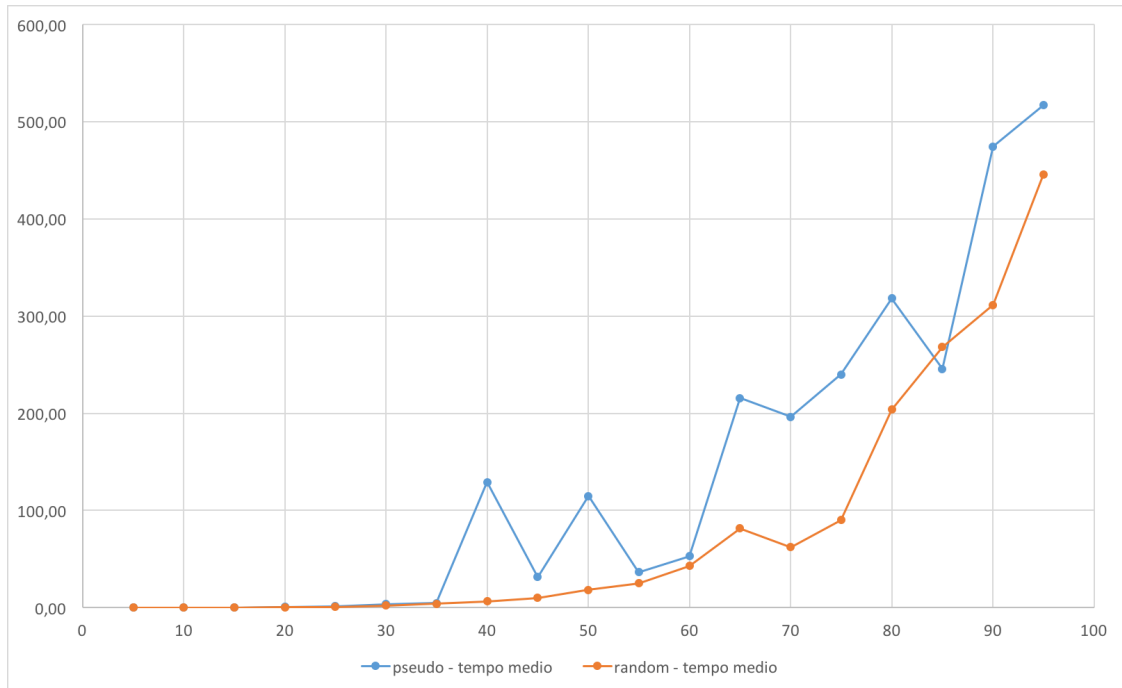


Figura 2: Tempo medio impiegato da CPLEX per risolvere le varie istanze

Time limit (s)	Dimensione	
	Pseudo	Rand
1	20	25
10	35	45
60	60	60
600	85	95

Tabella 2: Massima difficoltà mediamente risolvibile entro i rispettivi time limit. Per determinare la difficoltà è stata usata la media, senza prendere in considerazione i casi limite in cui già per N bassi non viene trovata una soluzione.

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 utilizzare un'array di lunghezza $N + 1$, dove N è il numero di nodi da visitare o fori da effettuare, che rappresenta la sequenza in cui vengono visitati i nodi. 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. Un esempio della codifica è disponibile in figura 3.

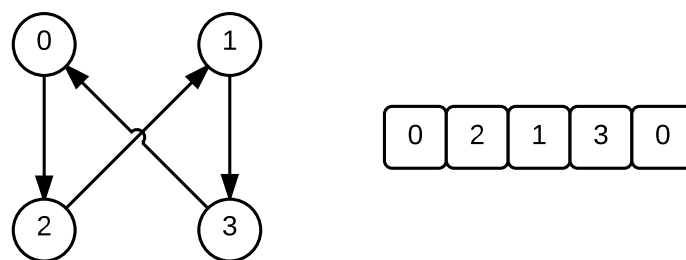


Figura 3: Possibile percorso su un grafo di 4 nodi con la relativa rappresentazione interna.

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.

Per quanto riguarda la dimensione della popolazione, si è scelto di implementarla come parametro dell'algoritmo, in modo che sia possibile specificare una dimensione diversa in base alla dimensione

delle istanze che l'algoritmo si troverà a risolvere. Questo perché è ragionevole assumere che istanze di grandi dimensioni hanno un numero maggiore di possibili soluzioni e quindi poter specificare una dimensione *ad-hoc* permette di avere una popolazione sufficientemente grande da rappresentare un campione significativo delle possibili soluzioni.

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 $K = \text{POPULATION_SIZE}/10$ 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 $\text{succ}(x, G)$ il nodo successivo al nodo x nel cammino della soluzione G , pertanto nella soluzione G sarà presente l'arco $(x, \text{succ}(x, G))$ 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 che il secondo nodo del cammino verrà scelto casualmente tra $\text{succ}(0, G_1)$ e $\text{succ}(0, G_2)$. 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 $\text{succ}(y, G_1)$ o $\text{succ}(y, G_2)$. 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. In questo caso come successore viene scelto il nodo migliore tra quelli disponibili.
- 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 della 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 visitato sarà sempre il nodo 0.

3.1.7 Sostituzione della popolazione

La sostituzione della 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.

Un 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 testare l'algoritmo genetico in modo analogo a quanto fatto per il modello 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) generate sia in modo casuale che pseudo-casuale. Le varie prove sono state poi ripetute 5 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.

Si è scelto di mantenere un `TIME_LIMIT` costante e limitato per rendere più agevole la pianificazione delle esecuzioni e allo stesso tempo per limitare la durata degli esperimenti. Infatti, fissato un N , per provare tutte e 9 le possibili combinazioni dei parametri, ripetendo la prova 5 volte per ogni istanza, sono necessarie 1350 esecuzioni, ovvero circa un giorno d'esecuzione.

Di ogni esecuzione è stata tenuta traccia:

- della soluzione peggiore;
- della soluzione migliore;
- dello stato della popolazione, ovvero il fitness medio;
- del numero di iterazioni effettuate.

mentre per scegliere la configurazione migliore è stata tenuta in considerazione la media di questi valori per le 5 esecuzioni sulla stessa istanza.

3.2.2 Risultati

Dall'esecuzione delle prove è emerso che per tutte e tre le possibili dimensioni i risultati migliori si ottengono con `GROWTH_RATIO= 2`, ovvero andando a generare ad ogni iterazione un numero di individui pari a quello della popolazione. Per quanto riguarda gli altri parametri si è visto che per $N = 100$ e $N = 150$, la configurazione migliore è `POPULATION_SIZE= 500` e `MUTATION_RATE= 0.1` mentre per $N = 50$ è `POPULATION_SIZE= 100` e `MUTATION_RATE= 0.05`.

Questa discrepanza può essere dovuta al fatto che all'aumentare della dimensione del grafo si ha un'esplosione combinatoria delle possibili soluzioni e quindi la dimensione della popolazione deve aumentare di conseguenza.

Per quanto riguarda il valore di `MUTATION_RATE`, una possibile motivazione per la differenza deriva dal fatto che con $N = 50$ l'algoritmo riesce a fare più iterazioni, mentre con un N maggiore ne vengono fatte meno di 10 per ogni istanza e quindi l'effetto delle mutazioni, combinato alla dimensione della popolazione più grande (500 per $N = 100$ e $N = 150$), risulta più efficace nel produrre soluzioni migliori rispetto all'avanzamento generazionale. Questo perché, considerando una popolazione di 500 individui con probabilità di mutazione del 10%, in un'iterazione l'effetto dell'avanzamento produce una nuova generazione con un fitness leggermente migliore, mentre la mutazione di 50 individui, essendo completamente casuale, può generare fin da subito un super-individuo con un elevato fitness.

Un'ultima osservazione riguarda alcune esecuzioni dell'algoritmo con istanze di dimensione $N = 150$, le quali hanno avuto un comportamento inatteso. Ovvero, mentre la maggior parte delle esecuzioni riusciva a fare solamente un'iterazione a causa del time limit di un minuto, le ultime esecuzioni hanno fatto in media 15 iterazioni. Tale anomalia può essere legata a come il sistema operativo ha assegnato la CPU al programma, anche se tutte le esecuzioni sono state effettuate sullo stesso computer e in condizioni simili. Per motivi di tempo non è stato poi possibile rieseguire l'algoritmo su tali istanze.

Infine, dato la dimensione consistente delle tabelle, queste non sono riportate nella relazione, ma sono contenute in un file Excel presente all'interno dell'archivio consegnato assieme alla relazione.

3.3 Test dell'algoritmo

Per testare l'algoritmo genetico sono state utilizzate le stesse istanze utilizzate per il modello CPLEX, in modo da poter calcolare anche l'optimality gap ottenuto dall'algoritmo genetico.

Come parametri dell'algoritmo sono stati utilizzati:

- `POPULATION_SIZE: 100;`
- `MUTATION_RATE: 0.05;`
- `GROWTH_RATIO: 2.`

Ovvero quelli che sono risultati migliori dal processo di ottimizzazione.

L'algoritmo è stato eseguito quindi 3 volte per ogni istanza e poi è stata presa in considerazione la media dei risultati. Inoltre, per ridurre il tempo necessario è stato assegnato come time limit per la durata dell'esecuzione dell'algoritmo genetico la stessa soglia temporale che ha utilizzato CPLEX. I vari time-limit per le istanze sono riportati in tabella ...

Inserire
tabella
time-
limit

4 Confronto

A Informazioni sul codice e sulla consegna

Il codice consegnato è suddiviso in varie sotto-directory:

- `common`: contiene le classi comuni, utilizzate sia dal modello CPLEX che dall'algoritmo genetico.

- `cplex`: contiene le classi relative all'implementazione del modello CPLEX.
- `ga`: contiene le classi relative all'implementazione dell'algoritmo genetico.

Il codice è stato poi strutturato in modo simile ad una libreria: per entrambe le parti del progetto c'è una classe che incapsula l'algoritmo risolutivo e che espone un metodo `solve()`. Pertanto non è presente un vero e proprio file principale dell'applicazione.

Vengono però forniti vari file `.cpp` che contengono degli esempi di utilizzo della libreria e che corrispondono ai programmi utilizzati per eseguire i test. Viene anche fornito un `main.cpp` che risolve in entrambi i modi un'istanza di un problema generata casualmente. Tale programma può essere compilato utilizzando il comando `make`. All'interno del `Makefile` sono definite anche le regole per compilare gli altri sorgenti.

A.1 Possibili problemi di compilazione

Il programma `main.cpp`, e di conseguenza tutto il codice prodotto, compilano correttamente sui computer del laboratorio con `g++ v5.4`. Tuttavia potrebbero esserci dei problemi nel linking delle librerie di CPLEX. In questo caso è necessario andare ad aggiustare il *filepath* definiti all'interno del `Makefile`.