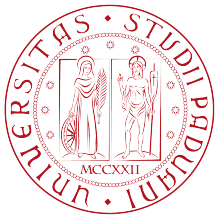

Università di Padova

Laurea Magistrale in Informatica

Corso di Methods and Models for Combinatorial Optimization

Relazione finale sugli esercizi di laboratorio

Marco Romanelli

marco.romanelli.1@studenti.unipd.it

matricola n. 1106706

9 giugno 2017

Indice

1 Introduzione

L'obbiettivo del progetto è quello di implementare il noto problema combinatorio del commesso viaggiatore (Traveler Salesman Problem), d'ora in avanti TSP, in due modalità differenti per poi valutarne le differenze. In particolare, la prima parte consiste nell'implementazione del problema utilizzando le API di CPLEX e trovare la dimensione massima del problema risolvibile entro un certo intervallo di tempo (fino a 1 secondo, fino a 10 secondi, eccetera). Nella seconda parte, invece, si chiede di studiare e implementare un algoritmo di ottimizzazione ad-hoc, utilizzando una qualsiasi meta-euristica vista a lezione. Fatto ciò si richiede di testare l'implementazione, presentarne i costi computazionali e compararne i risultati con il metodo utilizzato nella prima parte.

La presente relazione procederà come segue. Per prima cosa nella sezione 2 verrà presentato il problema della generazione delle istanze. La sezione 3 illustrerà la prima parte del progetto, partendo dalla soluzione con CPLEX e finendo con i risultati ottenuti. La sezione 4, invece, esporrà la seconda parte del progetto. Per concludere, nella sezione 5 verranno presentate alcune considerazioni finali. Nella sezione A dell'appendice si trovano alcune indicazioni sull'utilizzo del codice.

2 Generazione istanze

Come prima cosa è necessario creare delle istanze per il problema che sia andrà ad analizzare. Un'istanza di un problema di foratura (come descritto nella prima parte della consegna) può essere rappresentata da un grafo dove i nodi sono i fori e gli archi con relativo peso sono gli spostamenti e i costi che la trivella deve effettuare per muoversi. Rappresentando questo tramite una matrice dei costi, il problema ora è come generare tali nodi e distanze; i fori nei pannelli perforati, infatti, non sono distribuiti casualmente sullo spazio ma seguono delle logiche intrinseche del problema.

Per cercare, quindi, di riprodurre almeno in parte queste caratteristiche si è utilizzato una collezione di istanze per TSP basate su un data-set di VLSI, fornite da Andre Roh dell'Istituto di Ricerca per la Matematica Discreta dell'Università di Bonn, consultabili gratuitamente sul sito web ufficiale [1]. Un esempio di tale istanze si può vedere in figura 1.

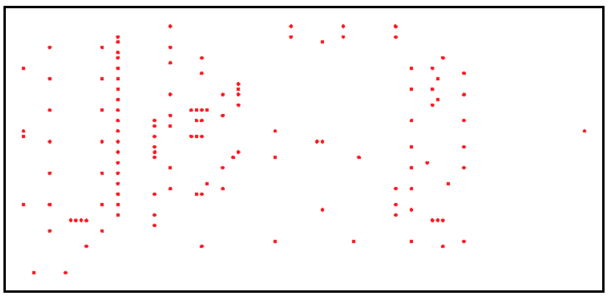

Figura 1: Esempio di problema TSP con 131 città.

Generazione istanze pseudo-casuali Queste istanze hanno però un numero di nodi molto elevato rispetto alle necessità e alle risorse di questa analisi. Inoltre, data la natura probabilistica dell'algoritmo previsto per la soluzione della parte 2 della consegna, è necessario eseguire le soluzioni proposte su molte istanze diverse.

[1] `http://www.math.uwaterloo.ca/tsp/vlsi/index.html`

Per questo motivo è stato implementato un generatore di istanze pseudo-casuali che, a partire dalle istanze TSP descritte in precedenza, generi delle sotto-istanze di dimensioni minori. Dati i punti dell'istanza originale, il generatore prima estrare casualmente un sotto-insieme di questi di dimensione variabile per poi calcolare le distanze (futuri costi) sui nuovi punti.

Partendo da un'istanza originale, il generatore crea delle nuove istanze con dimensione a partire da 5 fino ad arrivare a 40 con un passo di 5. Successivamente ne vengono create altre a partire da 50 con un passo di 10 fino a 80. Infine le ultime due vengono generate con dimensione 100 e 120 (da 100 a 120 con passo 20). Il processo viene ripetuto per quattro istanze originali, generando quindi 57 nuove istanze pseudo-casuali. Il numero del passo va via via aumentando per permettere di eseguire i test in tempi più ragionevoli (basti pensare che per ogni istanza pseudo-casuale l'algoritmo CPLEX ha tempo limite pari a 5 minuti, arrivando quindi a, potenzialmente, quasi 5 ore d'esecuzione per la prima consegna).

Per l'implementazione è stato creato appositamente uno script scritto nel linguaggio Python.

3 CPLEX

Il modello è già ampiamente descritto nella consegna, per cui verrà presentanta solamente una panoramica sull'implementazione delle variabili e dei vincoli utilizzando le API di CPLEX ed eventuali modifiche e/o accorgimenti adottati.

3.1 Definizione delle variabili

Nel modello sono presenti due variabili: le x_{ij} e le y_{ij}. A titolo esemplificativo, prendiamo le prime:

$$x_{ij} = \text{ amount of the flow shipped from } i \text{ to } j, \forall (i,j) \in A;$$

L'implementazione corrispondente è:

```
for (unsigned int i = 0; i < N; ++i) {
  for (unsigned int j = 0; j < N; ++j) {
    if (i == j) continue;

    char htype = 'I';
    double obj = 0.0;
    double lb = 0.0;
    double ub = CPX_INFBOUND;
    snprintf(name, NAME_SIZE, "x_%d,%d", nodes[i], nodes[j]);
    char* xname = &name[0];
    CHECKED_CPX_CALL( CPXnewcols, env, lp, 1, &obj, &lb, &ub, &htype, &xname );
    xMap[i][j] = created_vars;
    created_vars++;
  }
}
```

Codice 1: Creazione delle variabili x_{ij}

Una volta create, le variabili sono memorizzate internamente al risolutore e l'unico modo per accedervi è tramite la sua posizione nell'array interno. Per semplificare questo processo, viene creata una matrice $N \times N$ che associa il nome della variabile alla sua posizione interna del risolutore. Mentre viene tenuta traccia del numero di variabili create, l'indice della variabile corrente viene memorizzato nella mappa di supporto (xMap).

Un'altra osservazione che si può fare sul codice è il fatto che non vengono create le variabili x_{ij} quando gli indici sono uguali; infatti, se si pensa al problema reale, non si avrebbe guadagno nel farlo: spostare la trivella lungo l'argo (i,i) equivale a lasciarla ferma.

3.2 Definizione dei vincoli

Sebbene le API di CPLEX permettano la definizione di più vincoli in una sola chiamata, si è scelto di definire un vincolo per chiamata; in questo modo la definizione risulta più chiara e quindi più semplice da modificare in futuro. Il costo computazione derivante da tale scelta è comunque limitato, in quanto la definizione del problema avviene solo in fase di inizializzazione e questa ha un costo molto inferiore rispetto alla fase di ottimizzazione.

```
1  std::vector<int> varIndex(N-1);
2  std::vector<double> coef(N-1);
3
4  int idx = 0;
5  for (unsigned int j = 0; j < N; ++j) {
6    if (j == STARTING_NODE) continue;
7    varIndex[idx] = xMap[STARTING_NODE][j];
8    coef[idx] = 1;
9    idx++;
10 }
11
12 char sense = 'E';
13 double rhs = N;
14 snprintf(name, NAME_SIZE, "flux");
15 char* cname = (char*)(&name[0]);
16
17 int matbeg = 0;
18 CHECKED_CPX_CALL( CPXaddrows, env, lp, 0, 1, varIndex.size(), &rhs, &sense, &
       matbeg, &varIndex[0], &coef[0], NULL, &cname );
19 }
```

Codice 2: Creazione di un vincolo

I parametri della chiamata CPLEX sono (alcuni): il numero di variabili e vincoli da creare, il numero di variabili nel vincolo dove il coefficiente è diverso da zero, la parte destra del vincolo e il verso (in questo caso definisce l'uguale "="), il vettore rmatbeg (posto a zero in quanto creo un solo vincolo [2]), l'inizio dell'array con gli indici della variabili e poi quello con gli indici dei coefficienti e infine il nome del vincolo.

3.3 Verifica del modello

L'obbiettivo della prima parte della consegna consisteva nell'implementare il modello, eseguirlo su delle istanze di prova e osservare fino a che dimensioni il problema può essere risolto entro un certo tempo. Le prove sono state effettuate nel laboratorio LabTA, su una macchina con sistema operativo Ubuntu 64-bit, con un Intel Core i5 a 3,30GHz e 4GB di memoria RAM. Le istanze sono state generate come descritto nella sezione precedente (sezione 2). In tabella 1 si possono trovare i risultati delle esecuzioni, ordinati per dimensione del problema crescente, fra cui si trovano i tempi medi e la percentuale di fallimenti (rispetto alle istanze della stessa dimensione).

Dai risultati si può osservare come:

- il tempo d'esecuzione aumenta in maniera esponenziale, in particolare a partire dalle istanze con $N = 20$.

[2] Non è necessario utilizzare il vettore per tenere traccia delle righe perché definendo un solo vincolo si ha una sola riga che inizierà alla posizione 0.

- Il tempo massimo d'esecuzione, pari a 5 minuti (300 s), viene raggiunto per la prima volta nei problemi con $N = 70$.[3]
- Il primo fallimento avviene con l'istanza di dimensione $N = 80$, mentre con istanze con $N = 120$ CPLEX non riesce a trovare una soluzione in nessuna delle istanze (100% di fallimenti).

Dimensione	Tempo minimo [s]	Tempo medio [s]	Tempo massimo [s]	Fallimenti
5	0,02	0,03	0,04	0%
10	0,41	0,77	1,38	0%
15	1,48	3,79	6,91	0%
20	2,83	7,86	16,40	0%
25	28,25	50,95	106,42	0%
30	11,54	112,96	300,01	0%
35	58,54	181,42	300,02	0%
40	214,62	276,03	300,03	0%
50	122,11	255,55	300,03	0%
60	234,13	283,56	300,04	0%
70	300,04	300,05	300,07	0%
80	189,81	263,32	300,11	25%
100	300,09	300,10	300,11	50%
120	-	-	-	100%

Tabella 1: Risultati dell'esecuzione con risolutore CPLEX

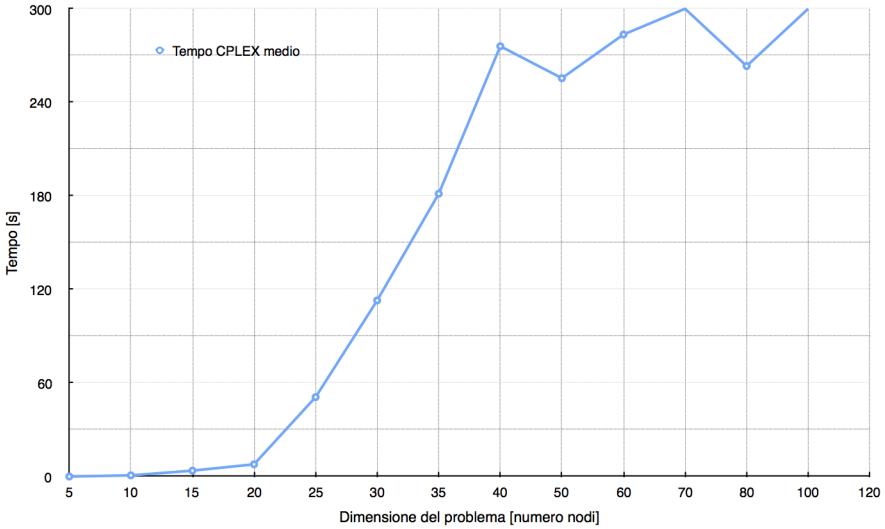

Figura 2: Tempo medio impiegato dal risolutore CPLEX, in secondi.

Infine, come richiesto nella consegna, sono stati definiti quattro intervalli di tempo: 0,1, 1 e 10 secondi, 1 e 5 minuti. Per il limite di 5 minuti sono stati inseriti in tabella due valori; il primo problema per cui CPLEX supera i 5 minuti è quello con dimensione $N = 70$, quindi di conseguenza riesce a risolvere i problemi con $N = 60$ al di sotto del limite dei 5 minuti.

[3] Il tempo è leggermente superiore ai 300 secondi in quanto il conteggio viene fatto quando il risolutore ritorna il risultato. Il valore in eccesso è il tempo che impiega la macchina da quando il risolutore si ferma a quando viene preso il conteggio del tempo.

Tuttavia, anche le istanze con $N = 80$ sono state tutte risolte entro il limite e per questo motivo anche questo è stato indicato anche questo valore.

Tempo limite	**Dimensione**
0,1 secondo	5
1 secondo	10
10 secondi	20
1 minuto	25
5 minuti	60 (80)

Tabella 2: Dimensione massima risolvibile entro un certo tempo.

4 Algoritmo Genetico

La seconda parte prevedeva l'implementazione di un algortimo ad-hoc, scegliendo fra le varie metaeuristiche presenti nelle dispense del corso; fra queste, si è scelta la classe dei metodi basati su popolazione e, in particolare, gli Algoritmi Genetici, molto diffusi grazie alla loro adattabilità e semplicità implementativa.

4.1 Operatori genetici

Trattandosi di una metaeuristica, lo schema principale è generale e lascia molto spazio di manovra. Si inizia con la *codifica delle soluzioni*, per poi definire i diversi *operatori genetici*:

- la generazione di un insieme di soluzioni (popolazione iniziale);
- funzione che valuta la fitness di una soluzione;
- operatori di ricombinazione;
- operatori di passaggio generazionale.

Per la descrizione della metaeuristica si rimanda alle dispsense, mentre qui si evidenzieranno solo le scelte progettuali.

4.1.1 Codifica degli individui

Un individuo della popolazione è una seguenza di $N + 1$ geni, dove N è la dimensione del problema e dove ciascun gene in posizione i indica un nodo (o foro per la trivella) da visitare in posizione i nel ciclo hamiltoniano (il percorso che dovrà fare la trivella). La sequenza ha dimensione maggiore di 1 rispetto alla dimensione del problema per codificare direttamente il vincolo che la trivella ritorni al punto di partenza; l'ultimo gene, infatti, avrà sempre valore uguale al primo e cioè 0.

4.1.2 Popolazione iniziale

La popolazione iniziale viene creata generando delle soluzioni (individui) in modo pseudo-greedy, ovvero incrementalmente a partire dal nodo inziale scegliendo il successivo casualmente fra quelli ancora da visitare, ma dando più probabilità a quelli con costo minore. Questo per permettere sia una diversificazione degli individui (parte probabilistica) mantentendo tuttavia una convergenza abbastanza veloce (scelte pesate).

La dimensione è specificata tramite un parametro ed è direttamente proporzionale alla dimensione del problema; è ragionevole pensare, infatti, che istanze maggiori abbiano un spazio delle soluzioni maggiore.

4.1.3 Funzione di fitness

La funzione di fitness è una misura quantitiva della bontà di un individuo; in questo è stata scelta la funzione obbiettivo del modello e cioè il costo del ciclo della soluzione.

4.1.4 Operatori di selezione

Gli operatori di selezione scelgono tra la popolazione corrente gli invidui che partecipano ai processi riproduttivi. La scelta dev'essere in parte pesata sugli invidui migliori, in modo da trasmettere le migliori caratteristiche alla progenie, ma dev'esserci anche una componente casuale che permetta di convergere più lentamente e introdurre caratteristiche diverse che potrebbero rivelarsi anch'esse migliori.

Sono stati proposti diversi metodi per combinare questi due aspetti, ma la scelta è ricaduta sul metodo chiamato *Torneo-n*. Vengono quindi prima scelti n individui e poi fra questi viene estratto il miglior candidato; il processo viene ripetuto una seconda volta in modo da ottenere due genitori.

Il valore degli n individui è impostato sul 20% della popolazione.

4.1.5 Operatori di ricombinazione

Gli operatori di ricombinazione, chimati in inglese di *crossover*, generano uno o più figli a partire dai genitori. Come già detto, i genitori in questo caso sono due (come avviene nella maggioranza dei casi) e viene generato un solo individuo figlio utilizzando il metodo noto come *cut-point crossover*. Il metodo nasce dall'ipotesi che geni vicini fra loro controllino caratteristiche tra loro correlate e affinché i figli preservino tali caratteristiche sia necessario trasmettere blocchi di geni contigui. In dettaglio, vengono definiti casualmente 2 punti di taglio (e quindi sarà un 2-cut crossover) e il figlio si ottiene copiano i blocchi dei genitori alternativamente. Per

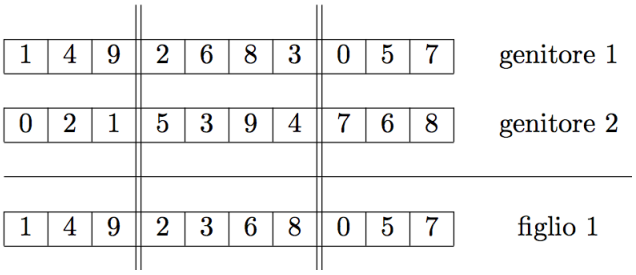

Figura 3: Esempio di 2-cut-point crossover con un figlio.

preservare l'ammissibilità dei figli (e generare quindi soluzioni ammissibili) è stato implementata una variante del metodo chiamata *order crossover*. In questa tecnica, il figlio riporta le parti esterne dal primo genitore mentre i restanti geni (blocco interno del figlio) si ottengono copiando i geni mancanti nell'ordine in cui appiaiono nel secondo genitore. Si noti che questo metodo preserva il vincolo che il primo e l'ultimo gene siano entrambi pari a 0.

4.1.6 Mutazione

Per evitare il fenomeno chiamato assorbimento genetico (convergenza casuale di uno o più geni verso lo stesso valore), l'operatore di mutazinone modifica casualmente il valore di alcuni geni, scelti casualmente anch'essi.

Anche in questo, caso è stato implementato un metodo che preserva l'ammissibilità, chiamato *mutazione per inversione*, dove sono generati casualmente due punti della sequenza e si inverte la sottosequenza tra questi.

La probabilità di mutazione è modificabile tramite un apposito parametro ed è solitamente molto bassa.

4.1.7 Sostituzione della popolazione

A questo punto si ha un nuovo insieme di individui che va sostituito alla generazione corrente. Diverse tecniche sono state proposte e si è scelto di implementare la politica chiamata *selezione dei migliori*. Si mantengono nella popolazione corrente gli N individui migliori tra gli $N + R$ (con N dimensione della popolazione corrente e R i nuovi individui generati), effettuando la selezione utilizzando il metodo Montecarlo con probabilità proporzionale al valore di fitness.

4.1.8 Criterio di arresto

I possibili criteri per l'arresto sono diversi e sono possibili anche combinazioni fra questi. In questo caso si è scelto un metodo ibrido che prevede un numero massimo di iterazioni (evoluzioni della popolazione) e tempo massimo d'esecuzione, entrambi specificabili come parametri all'avvio dell'algoritmo.

Il tempo limite permette un confronto più semplice con il metodo CPLEX (i tempi limite sono infatti uguali) ed evita che l'algoritmo richieda troppo tempo quando la dimensione del problema è molto grande; il numero massimo di iterazioni permette invece di limitare il tempo d'esecuzione quando le istanze hanno una bassa dimensione.

4.2 Calibrazione dei parametri

Come detto in precedenza, gli algoritmi genetici dipendono molto dalla loro configurazione, ossia dai valori che si sceglie di dare ai vari parametri. Nel caso in esame, quattro sono i parametri su cui si è andato ad agire:

- tempo limite per l'evoluzione della popolazione;
- numero massimo di iterazioni o, in altre parole, numero massimo di evoluzioni della popolazione;
- fattore di moltiplicazione della popolazione: la popolazione iniziale avrà dimensione pari a questo valore moltiplicato per la dimensione del problema;
- probabilità che un individuo subisca una mutazione.

Limite di tempo Dovendo iterare l'esecuzione dell'algoritmo su diverse istanze e dovendone confrontare i risultati con l'implementazione con CPLEX, è stato necessario introdurre un limite massimo all'esecuzione. Il valore è impostabile secondo le preferenze del momento e al problema che si sta considerando, a seconda del fatto che si voglia una soluzione migliore oppure maggior rapidità d'esecuzione. Per motivi logistici e di tempistiche, il valore scelto è pari a 5 minuti.

Numero massimo di iterazioni Come per il limite di tempo, questo valore è stato deciso sulla base di diverse prove effettuate e cerca di bilanciare una buona evoluzione della popolazione inziale e ovvi costi computazionali. Il valore scelto di 500 iterazioni sembra essere un buon compromesso.

Dimensione della popolazione iniziale In questo caso è stato deciso di non specificare direttamente la dimensione iniziale, bensì di impostare un fattore di moltiplicazione che verrà combinato con la dimensione del problema. Visto che, nelle istanze su cui si andrà poi a eseguire l'algoritmo, questo valore potrà assumere una gamma di valori abbastanza ampia (da 5 a 100),

impostare un numero fisso per la dimensione iniziale sarebbe risultato troppo abbondante per le piccole istanze e troppo riduttivo per quelle grandi.

Per determinare il valore del fattore di moltiplicazione si sono fatte diverse prove e si è visto che, naturalmente, all'aumentare di tale valore l'algoritmo terminava con una soluzione migliore (un numero maggiore di invidui implica una maggior probabilità che la soluzione ottima sia fra questi o che le 'caratteristiche genetiche' di queste siano presenti) ma ovviamente aumentava di molto tempo d'esecuzione. Considerato che la fase di creazione della popolazione e successivamente la sua sostituzione sono le fasi computazionalmente più onerose, si è preferito tenere basso il fattore di moltiplicazione, preferendo un'esecuzione più snella a fronte di una soluzione leggermente meno ottimale.

Il valore finale scelto è pari a 3, producendo quindi una popolazione iniziale di 3 volte superiore alla dimensione del problema.

Probabilità della mutazione Una probabilità di mutazione alta aumenta le probabilità di esplorare più aree dello spazio di ricerca ma rallenta la convergenza all'ottimo globale; d'altra parte, un valore basso potrebbe portare a una prematura convergenza (ottimo locale). Seguendo anche le indicazione fornite sulle dispense, il valore finale scelto è $0,05$.

4.3 Verifica del modello

La seconda parte della consegna prevedeva di eseguire il risolutore proposto, analizzarne le prestazioni e compararle con quelle del risolutore CPLEX. Anche in questo caso le prove sono state eseguite nel laboratorio, utilizzando le istanze generate allo stesso modo (si veda il capitolo 3.3).

I parametri sono stati impostati come descritto precedentemente; si riportano qui i valori per una più facile lettura:

- time_limit: 5 minuti (come con il risolutore CPLEX);
- iteration_limit: 500,
- population_size_factor: N * 3 (con N dimensione del problema),
- mutation_probability: $0,05$.

In tabella 3 si possono trovare i tempi d'esecuzione dell'algoritmo, espressi in secondi, per problemi di dimensione da $N = 5$ a $N = 120$, mostrati in ordine crescente. Per osservare meglio l'andamento del tempo d'esecuzione al variare di N, in figura 4 si può trovare raffigurato il tempo medio. La figura 5 mostra, invece, la differenza fra i tempi d'esecuzione medi del risolutore CPELX e del risolutore GA. Infine, la figura 6 mostra l'andamento della differenza fra le soluzioni trovate dal risolutore GA e quelle del risolutore CPLEX, in media, espressa in percentuale, rispetto al crescere della dimensione del problema.

Dai risultati si può osservare come:

- i tempi sono di molto infereriori rispetto al risolutore CPELX; basti osservare come per istanze con $N = 25$ il risolutore CPLEX impieghi un tempo medio di quasi 51 secondi, mentre ne bastano meno di 2 per quello GA.
- Anche in questo caso si ha un andamento esponenziale al crescere di N; mentre nel caso precedente si era visto come la curva esponenziale iniziava a "impennarsi" a partire dai problemi con N pari a 20, con l'algoritmo GA questo succede da $N = 80$.
- Dalla figura si può notare come la differenza fra le soluzioni trovate dai due risolutori si mantenga pari a 0 fino a $N = 40$. Questo vuol dire che l'algoritmo GA riesce, fino a questo punto, a trovare la soluzione ottima al problema. Per dimensioni fino a $N = 40$, l'algoritmo

Dimensione	Tempo minimo [s]	Tempo medio [s]	Tempo massimo [s]
5	0,002	0,004	0,008
10	0,18	0,24	0,26
15	0,60	0,60	0,60
20	1,12	1,12	1,13
25	1,73	1,73	1,74
30	2,54	2,54	2,55
35	3,51	3,52	3,53
40	4,76	4,79	4,82
50	9,25	9,31	9,36
60	14,20	14,22	14,26
70	19,84	19,88	19,91
80	27,00	27,11	27,24
100	46,92	47,39	47,96
120	75,00	75,06	75,11

Tabella 3: Risultati dell'esecuzione con risolutore GA

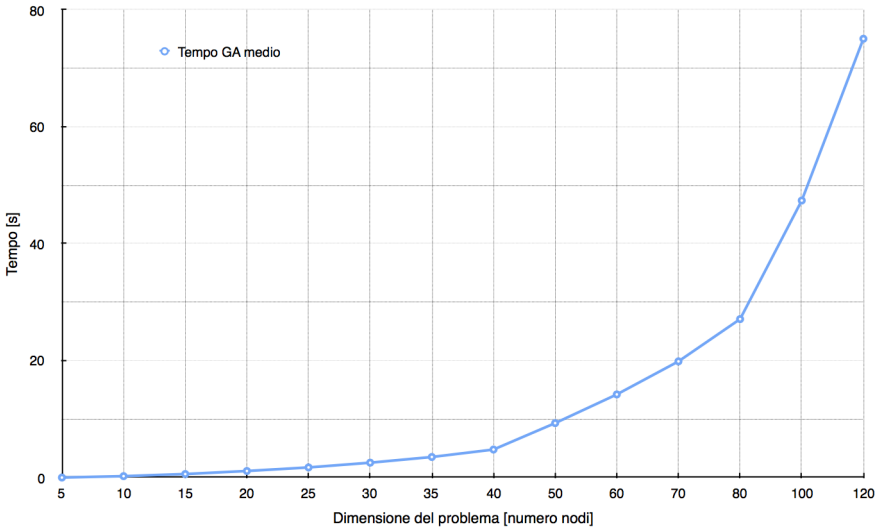

Figura 4: Tempo medio impiegato dal risolutore GA, in secondi.

non riesce a trovare la soluzine ottima, ma riesce tuttavia ad avvicinarsi con uno scarto del $1,32\%$. Successivamente nota invece un comportamento strano: mentre nelle istanze con $N = 60$ lo scarto è del $6,84\%$, in linea con i valori precendenti, sulle istanze con dimensione pari a 50 lo scarto è oltre il 13%. Su questo tipo di istanze sono state fatte diverse prove e tutte riportavano (a parte piccoli discostamenti) lo stesso valore. Probabilmente con queste istanze l'algoritmo generava una popolazione iniziale nettamente peggiore rispetto alle precedenti, oppure non riusciva a migliorla di molto durante l'evoluzione.

Infine, si nota come con istanze di dimensione maggiore a 70 lo scarto è positivo, ossia il risolutore GA trovava una soluzione migliore rispetto a quella trovata da CPLEX. Questo perché con queste dimensioni il risolutore CPLEX eccedeva il limite di tempo imposto e non riusciva, di conseguenza, a trovare la soluzione ottima al problema.

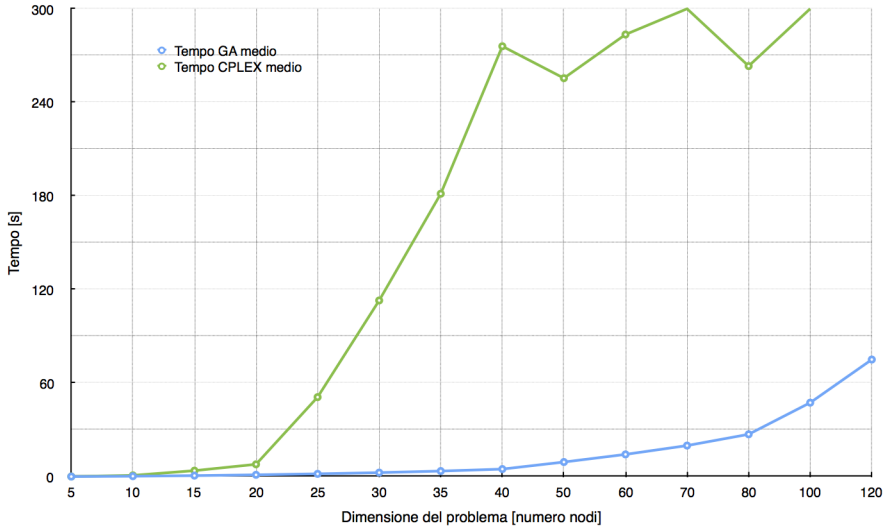

Figura 5: Tempo medio impiegato dai risolutori GA e CPLEX, in secondi.

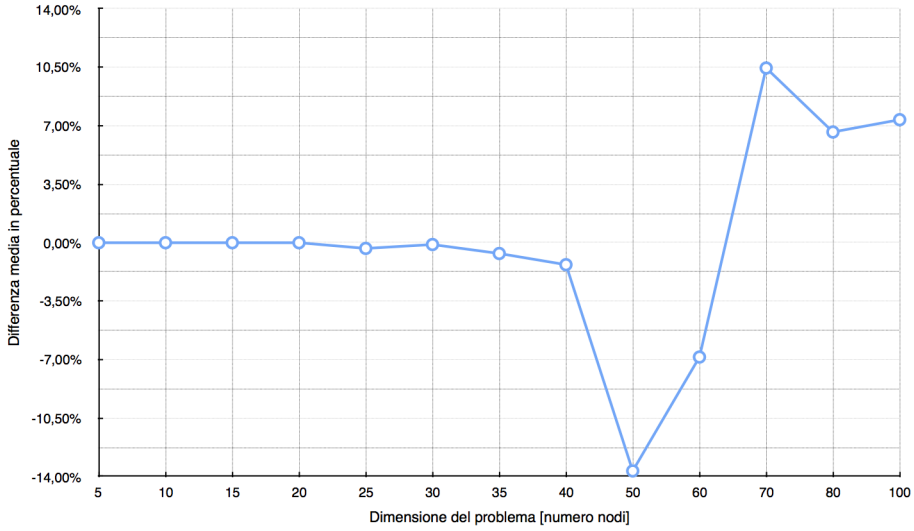

Figura 6: Differenza, in percentuale, fra le soluzioni trovate dai due risolutori.

4.4 Possibili miglioramenti

Come si può notare dai risultati esposti in precedenza, sull'insieme di istanze proposto l'algoritmo GA non ha mai raggiunto il tempo limite imposto di 5 minuti (tempo massimo d'esecuzione: circa 75 secondi) e quindi l'arresto è da attribuirsi esclusivamente al limite sul numero di iterazioni, pari a 500. Tuttavia questo valore è stato deciso dopo prove sperimentali su tutto l'inisieme di istanze. Probabilmente il numero di iterazioni necessarie all'algoritmo per convergere con un problema di $N = 5$ non è lo stesso di uno con $N = 100$. Inoltre, a parte il criterio d'arresto, l'algoritmo non ha alcun controllo sulla convergenza e sulle caratteristiche della popolazione. Diversi miglioramenti sono quindi possibili:

- fasi di intensificazione e diversificazione: l'alternanza di queste fasi durante l'evoluzione potrebbe portare a una convergenza più veloce (per esempip con problemi semplici), ma anche meno "specializzata", con una popolazione che evolve con caratteristiche diverse (meno overfitting).

- Variare la probabilità di mutazione: anche in un contesto come quello descritto sopra.
- misure di convergenza: come, per esempio, la distanza di Hamming; sia come forma di misura di fitness, come criterio d'arresto o per alternare diverse fasi nell'evoluzione.
- criterio d'arresto: oltre a quello appena descritto (distanza di Hamming) e visto in questo caso si aveva a disposizione la soluzione fornita da CPLEX, sarebbe possibile impostare uno scarto accettabile e far terminare l'algoritmo quando si raggiunge questo livello.
- ottimizzazioni del codice: è possibile eseguire fasi dell'algoritmo in maniera parallela; questo aumenterebbe la complessiva implementativa ma se ne guadagnerebbe in tempo d'esecuzione.

5 Conclusioni

Come si è visto, in generale l'algoritmo GA ha trovato soluzioni in un tempo inferiore rispetto al risolutore CPLEX che, tuttavia, ha trovato spesso soluzioni migliori; inoltre le soluzioni trovate (sotto certe condizioni) con CPLEX sono garantite ottime mentre non è così con l'algoritmo genetico. Si tratta quindi di decidere, in base al problma che si vuole risolvere, se si è disposti ad avere soluzioni buone ma non garantite ottime con un costo computazionale inferiore, oppure la sicurezza di soluzioni ottime ma con un tempo d'esecuzione nettamente superiore.

Un appunto va anche fatto alle istanze su cui si sono effettuate le prove. Seppur generate da istanze TSP reali e con particolari accorgimenti per renderle più veritiere, le istanze pseudo-casuali sono tuttavia frutto di una generazione casuale e pertanto potrebbero non rappresentare a pieno le caratteristiche di un problema TSP (o di perforazione) reale. In aggiunta a ciò, la dimensione dei problemi rappresentate da queste, ossia il numero di fori da perforare, è molto limitata considerato che i problemi reali sono composti da migliaia di nodi.

A Note per l'utilizzo del codice

A.1 Compilazione ed esecuzione

La cartella contenente il codice è strutturata in modo tenere separati le varie tipologie di file utili all'esecuzione. Per questo motivo il codice viene fornito con un Makefile che include al suo interno i comandi necessari alla compilazione e all'esecuzione dei test. Per un corretto utilizzo sono necessari alcuni passi da eseguirsi sul terminale di un computer con sistema operativo basato su Linux[4].

Il progetto viene fornito sottoforma di archivio, quindi per prima cosa è necessario estrarre i file contenuti all'interno:

```
tar -zxvf progetto-memoc-romanelli.tar.gz
```

per poi spostarsi all'interno della cartella contenente il codice:

```
cd progetto-memoc-romanelli/codice/
```

Per compilare i sorgenti è sufficiente lanciare il comando Make:

```
make
```

Se la compilazione termina con successo, all'interno della cartella bin (per ulteriori dettagli si rimanda alla sezione A.2) si troverà l'eseguibile principale main.

Per eseguire il programma principale bisogna passare come parametro il percorso dell'istanza di prova desiderata, che si trova nella cartella samples; ad esempio, se si vuole utilizzare l'istanza dcc1911_n025.tsp il comando da utilizzare è:

```
bin/main "samples/dcc1911_n025.tsp"
```

A.1.1 Creazione delle istanze

Se invece si volesse eseguire il programma su tutto l'insieme di istanze, generate come scritto in sezione 2, per riproporre le prove fatte, è necessario utilizzare altri comandi. Per prima cosa vanno generate le istanze:

```
make gen-instances
```

A questo punto è possibile eseguire gli algoritmi sull'insieme di istanze utilizzando il risolutore CPLEX:

```
make run-cplex
```

oppure il risolutore che utilizza l'Algoritmo Genetico:

```
make run-ga
```

[4] I comandi descritti, così come i processi di compilazione ed esecuzione, sono stati provati e verificati sui computer del laboratorio LabTA.

A.2 Struttura della cartella

Come già accennato, la disposizione dei file all'interno della cartella è ben strutturata e definita. Per questo motivo, viene proposta una veloce panoramica dalla disposizione delle cartelle e dei file al suo interno:

- bin: contiene il file binario generato dalla compilazione;
- include: all'interno sono presenti tutti i file header;
- instances: una volta generate, le istanze si troveranno qui dentro;
- samples: contiene alcune istanze di prova;
- scripts: si trova lo script in python che si occupa della generazione delle istanze (per i test completi);
- src: i sorgenti per gli algoritmi risolutivi delle due parti;
- vlsi-dataset: contiente i file delle istanze originali, da cui poi verranno generate tutte le altre.

I sorgenti sono molteplici e una breve descrizione è presente all'interno del file nelle prime righe; Tuttavia, le parti fondamentali degli algoritmi richiesti sono contenute solamente in alcuni, che sono:

- CPLEXSover: contiene la risoluzione del problema posto utilizzando le API di CPLEX (risoluzione parte 1);
- GASolver e GAPopulation: qui si trova la risoluzione mediante l'agoritmo genetico (parte 2);
- Main: principalmente si occupa di iterare gli algoritmi su tutte le istanze, ma può essere visto come esempio su come utilizzare propriatamente le classi e i metodi per la risoluzione di un problema (ad esempio nella funzione single_test);

A.3 CPLEX

Utilizzando il Makefile non è necessario conoscere i nomi dei file o i loro parametri ed è sufficiente utilizzare i comandi descritti in precedenza (sotto-capitolo A.1) per compilare ed eseguire il codice. Tuttavia potrebbe rendersi necessaria una modifica al Makefile nel caso in cui il percorso di installazione di CPLEX sia diverso da quello indicato[5]. In questo caso è necessario aprire il file Makefile con un editor di testo (nano, emacs, gedit, sublime) e modificare il percorso a CPLEX, indicato nelle voci CPX_INCDIR e CPX_LIBDIR.

```
1 CPX_INCDIR := <PERCORSO_CPLEX>/include/
2 CPX_LIBDIR := <PERCORSO_CPLEX>/lib/x86-64_linux/static_pic
```

[5]Il Makefile presente nel codice conseganto al docente è già stato adattato per funzionare nei computer del laboratorio. La modifica è necessaria nel caso si voglia compilare su una macchina diversa.