RELAZIONE MEMOC

GIULIO LOVISOTTO – 1084847

UNIVERSITA' DEGLI STUDI DI PADOVA

16 Settembre 2015

1 INTRODUZIONE

Questo documento descrive gli approcci sviluppati per l'esercitazione del corso di Metodi e Modelli per l'Ottimizzazione Combinatoria, anno accademico 14/15. L'esercitazione consiste nella risoluzione del Traveling Salesman Problem (TSP) con l'uso di metodi di ottimizzazione. E' stato implementato un modello per il risolutore esatto CPLEX, e una metaeuristica particle swarm optimization PSO. Il seguente documento e' strutturato come segue: in Sezione 2 viene descritto il modello in CPLEX (relativo alla Parte 1), in Sezione 3 viene descritto come e' stato implementato PSO (relativo alla Parte 2), in Sezione 4 vengono descritte le istanze del problema e la metodologia usate per l'esperimento, e presentati i risultati. Il documento termina con alcune conclusioni in Sezione 5.

1.1 FILES CONSEGNATI

- la cartella `cplex`, contiene il codice per il risolutore CPLEX;
- la cartella `pso`, contiene il codice per il risolutore PSO;
- il file `gen.py`, utilizzato per generare le istanze di TSP;
- l'archivio `datasets.zip`, contiene le istanze usate per gli esperimenti;
- l'archivio `results.zip`, contiene i risultati degli esperimenti.

Tutto il codice prodotto e i file usati per questa esercitazione sono disponibili sulla piattaforma GitHub all'indirizzo https://github.com/giuliolovisotto/memoc.

2 MODELLO CPLEX

In questa sezione viene descritto in che modo e' stato realizzato il modello per TSP che viene usato per la risoluzione con CPLEX. Per realizzare tale modello e' stata utilizzata la formalizzazione di TSP fornita dal docente [1]. Tale formalizzazione modella il problema come un problema di ottimizzazione su reti di flusso. E' stato realizzato un unico file `cplex.cpp`. L'esecuzione del file procede secondo i seguenti passi:

1. legge il problema in input, che consiste in un file di testo riportante la matrice contenente i costi degli archi del grafo (separati da virgole), e inizializza le variabili del problema e la funzione obiettivo, impostando i vincoli di tipo e i bounds per ciascuna variabile;

2. per ogni vincolo, lo crea e lo aggiunge alla matrice dei coefficienti dell'istanza del problema. Le variabili relative ai cappi (cioe' x_{ii}, y_{ii}) vengono rimosse;

3. esegue l'ottimizzazione e salva i risultati su file, tra i quali tempo di esecuzione, path ottimo, costo della soluzione ottima trovata.

3 PARTICLE SWARM OPTIMIZATION

In questa sezione vengono descritte come e' stato implementato PSO per la risoluzione di TSP.

PSO e' una metaeuristica basata su popolazione. In esso, gli individui x_k (possibili soluzioni) si muovono nello spazio N-dimensionale secondo le loro rispettive velocita' v_k. In PSO gli individui ricordano la miglior posizione da loro visitata, e la miglior posizione globale visitata dalla popolazione. Tali posizioni influiscono sulla velocita' dell'iterazione successiva.

Per applicare tale metodo a TSP e' necessario renderlo discreto. E' stata scelta la rappresentazione proposta in [2] e ripresa in [3]. In essa, gli individui sono cicli sul grafo che rappresenta il problema, e le velocita' sono sequenze di "Swap Operator". Uno Swap Operator (SO) e' definito da una coppia (i, j), la sua applicazione su una soluzione x_k ha l'effetto di scambiare la componente i-esima con la componente j-esima. E' importante notare che dato che le soluzioni x_k sono cicli sul grafo, l'applicazione di uno SO ad una soluzione x_k genera una nuova soluzione valida. Per formalizzare la rappresentazione e' necessario introdurre 4 operatori:

- $x + v$, dove x e' una possibile soluzione e v e' una sequenza di SO. Ottiene una nuova soluzione applicando gli SO in v alla soluzione x;
- $v_1 \otimes v_2$, dove v_1, v_2 sono sequenze di SO. Produce la piu' corta sequenza di SO il cui effetto e' equivalente all'applicazione di v_1 e v_2 in successione;
- $x - y$, dove x, y sono possibili soluzioni. Produce la piu' corta sequenza di SO che applicata a y (con l'operatore $+$) ritorna x;
- $a * v$, dove a e' uno scalare, e v e' una sequenza di SO. Produce una nuova sequenza di SO a partire da v rimuovendo con probabilita' $1 - a$ ogni suo SO.

La regola di aggiornamento delle velocita' per il generico individuo k all'iterazione i-esima e' la seguente:

$$v_k^{(i+1)} = v_k^{(i)} \otimes \alpha * (p_k^{(i)} - x_k) \otimes \beta * (g^{(i)} - x_k) \quad \alpha, \beta \in [0, 1],$$

dove $p_k^{(i)}$ e' la miglior soluzione visitata dall'individuo k, e $g^{(i)}$ e' la miglior soluzione visitata dalla popolazione, finora. Alla fine dell'iterazione i-esima l'individuo x_k si trova in posizione $x_k + v_k^{(i)}$. Ad ogni iterazione ogni individuo viene valutato e le miglior soluzioni visitate vengono aggiornate.

E' stato realizzato un singolo file `pso.cpp`. Gli individui sono rappresentati da array di interi (`vector<int>`), dove ogni elemento fa riferimento ad un nodo del problema. Gli SO sono coppie di interi, le velocita' (sequenze di SO) sono array di coppie di interi (`vector<pair<int, int> >`). Gli operatori introdotti precedentemente in questo paragrafo sono realizzati tramite funzioni. Il file legge il problema in input (nel formato riportato in Sezione 2). La popolazione e' inizializzata con individui random, le velocita' iniziali sono sequenze di SO di lunghezza random tra 0 ed N (numero di nodi). Gli elementi su cui effettuare swap sono scelti in maniera random. Viene eseguita l'ottimizzazione e vengono salvati i risultati su file.

4 ESPERIMENTI

In questa sezione vengono riportate le tecniche utilizzate per generare le istanze del problema, le metodologie usate per gli esperimenti, e i risultati ottenuti.

4.1 Istanze di TSP

Le istanze di TSP consistono in insiemi di n punti, $n \in \{5, 10, 20, 40, 60\}$, distribuiti un quadrato bidimensionale con lato di dimensione 1. Sono stati usati 3 diversi criteri per la disposizione dei punti:

- **random** i punti sono scelte in maniera casuale;
- **uniform** i punti sono disposti a formare una griglia;
- **clustered** viene creato un grafo, e viene disegnato con l'utility DOT [4]. Le coordinate dei nodi nel layout ottenuto sono le posizioni dei punti.

Per le istanze con criterio **random** e **clustered** vengono generate 20 istanze per ogni dimensione. Figure 1 e 2 riportano degli esempi di istanza per questi layout. Una volta generata l'istanza, la matrice delle adiacenze riportante la distanza tra i punti (costo) viene calcolata e salvata in un file (verra' fatto riferimento a questo file come *dataset*). Viene calcolata sia la distanza **euclidea** che la distanza **manhattan**.

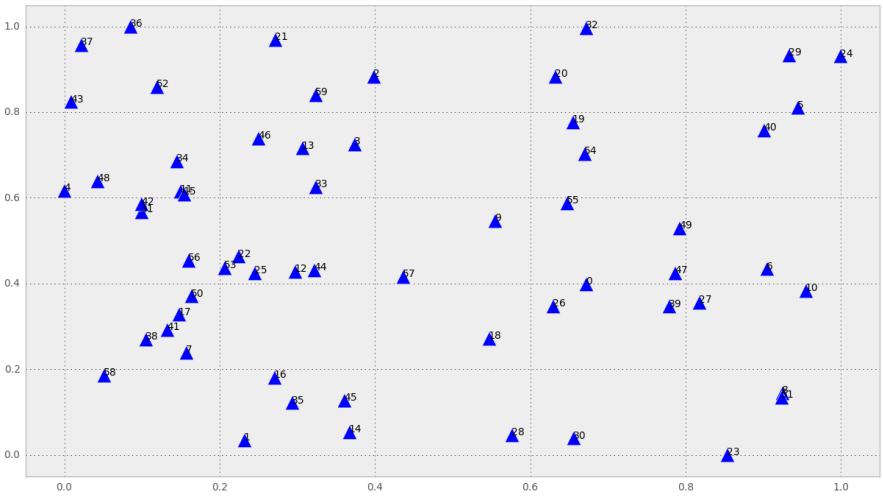

Figure 1: Esempio di istanza con layout random.

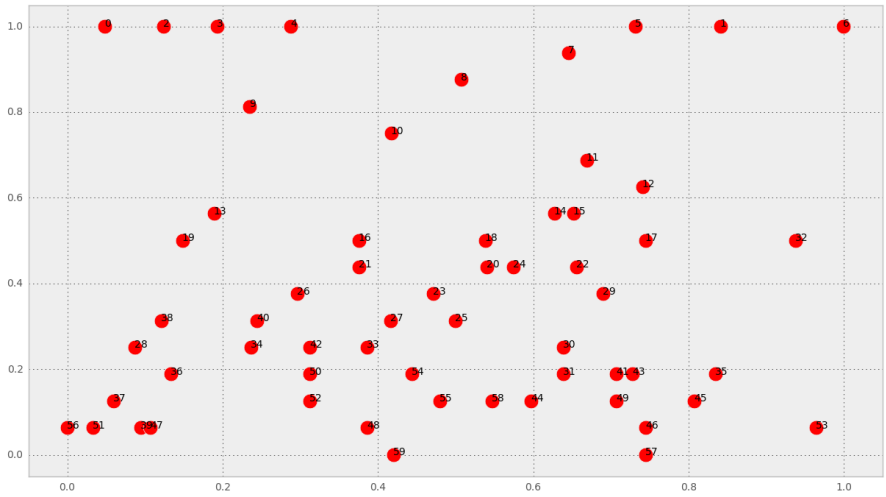

Figure 2: Esempio di istanza con layout clustered.

4.2 Metodologia e Parametri

Gli esperimenti consistono nell'utilizzare le tecniche descritte in Sezione 2 e 3 per risolvere il TSP, sui dataset generati. Per ogni dataset, ogni risolutore viene eseguito 5 volte, e vengono salvate le medie e le standard deviation dei tempi di esecuzione e dei costi delle soluzioni ottime trovate. Gli esperimenti sono stati svolti su macchine con processore Intel Core i5-2500 da 3.30 GHz.

Per PSO e' stata utilizzata una popolazione di 2.000 individui, 500 iterazioni, $\alpha = 0.75$, $\beta = 0.1$ (i parametri sono stati scelti empiricamente, senza un estensivo procedimento di ottimizzazione).

4.3 Risultati

In questa sezione vengon riportati i risultati degli esperimenti. Se non diversamente riportato, i valori nei grafici sono inizialmente mediati sulle diverse esecuzioni sul singolo dataset, e sono poi mediati sui diversi dataset.

In Figura 3 e' riportato il tempo di esecuzione in scala logaritmica per le diverse dimensioni dei dataset, diviso per metodo di risoluzione e per tipo di distanza. Si puo' vedere che CPLEX confrontato con PSO e' piu veloce per dimensioni basse del problema, mentre diventa piu' lento all'aumentare del numero di nodi. Questo e' dovuto al fatto che CPLEX ha tempo di esecuzione esponenziale nella dimensionalita' del problema, ma mostra che per problemi piccoli e' una valida opzione. Dal grafico si puo' inoltre notare che PSO e'

relativamente lento a dimensioni basse. Infatti tale metaeuristica ha una rappresentazione del problema non banale e quindi richiede dell'overhead iniziale per l'inizializzazione della popolazione e il successivo aggiornamento delle velocita'. Inoltre il numero di valutazioni di funzione utilizzate e' molto alto ($2.000 \times 500 = 1.000.000$), e non e' presente un criterio di fermata (il che significa che vengono effettuate tutte le valutazioni di funzione ad ogni esecuzione).

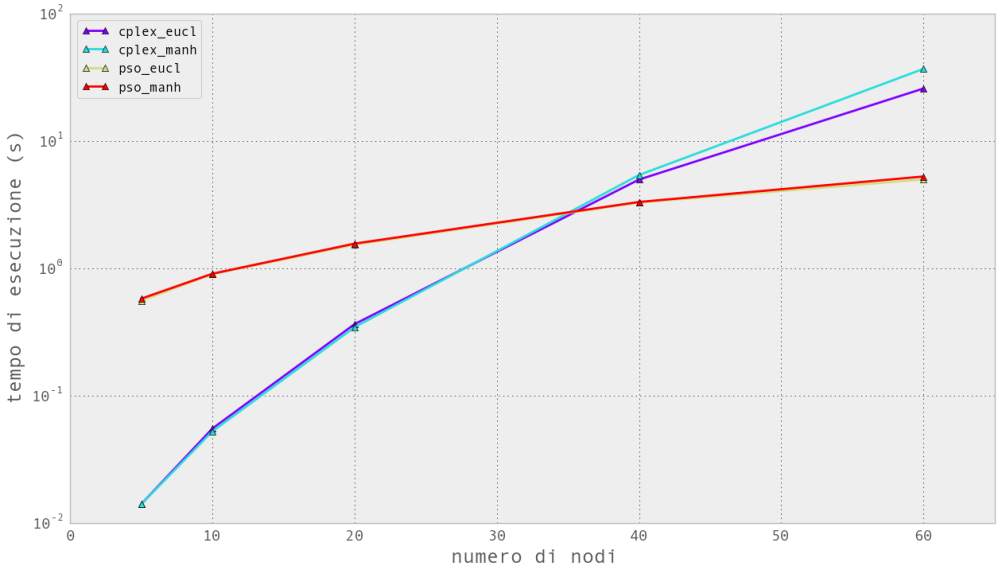

Figure 3: Tempi di esecuzione medi per le diverse dimensioni, divisi per metodo e tipo di distanza.

In Figura 4 e' riportato il tempo di esecuzione in scala logaritmica per le diverse dimensioni dei dataset, diviso per metodo di risoluzione e per criterio di disposizione dei punti. Si puo' vedere che per la disposizione dei punti di tipo **uniform** sia CPLEX che PSO risultano piu' veloci rispetto alle altre disposizioni **clustered** e **random**. Per queste non ci sono significative differenze.

In Figura 5 e' riportato l'errore medio della soluzione trovata da PSO, in percentuale, sul costo della soluzione ottima. Viene inoltre riportato lo scarto quadratico medio di tale errore. Si puo' vedere che PSO e' relativamente efficace a basse dimensioni, questo e' dovuto al fatto che con 1.000.000 valutazioni di funzione e' possibile esplorare buona parte dello spazio delle soluzioni su dataset ridotti. La metaeuristica diventa meno precisa a partire da 20 nodi, fino ad arrivare ad un errore medio del ~45% con 60 nodi.

Anche Figura 6 riporta l'errore medio in percentuale, diviso per tipo di punti del dataset. Le disposizioni di tipo **uniform** hanno un errore minore, mentre le

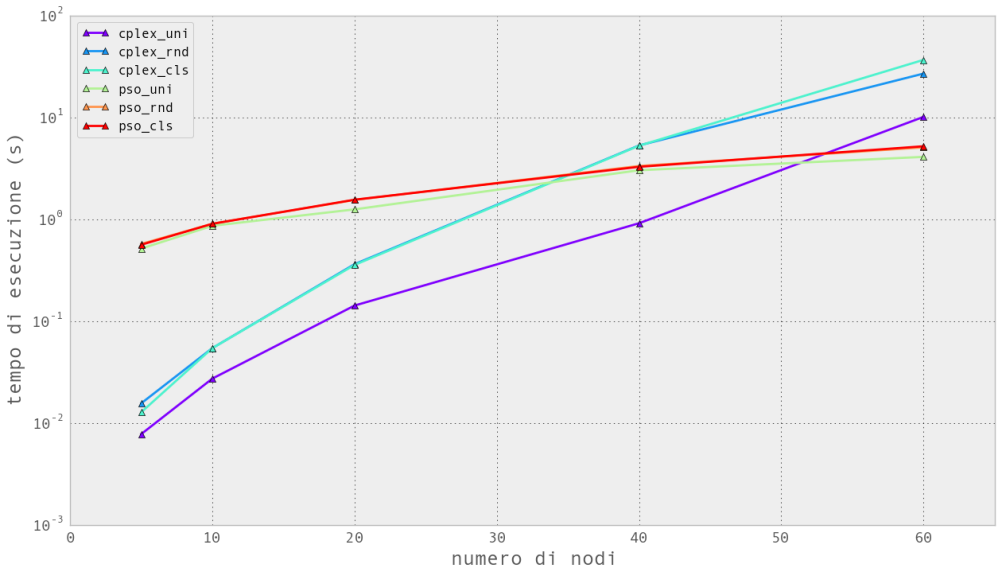

Figure 4: Tempi di esecuzione medi per le diverse dimensioni, divisi per metodo e criterio di disposizione dei punti del problema.

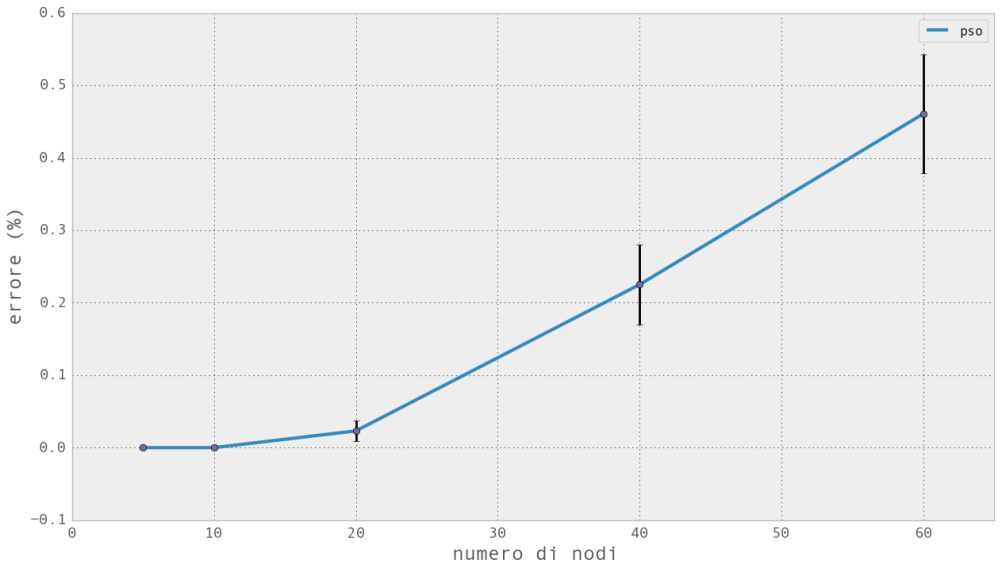

Figure 5: Media e scarto quadratico medio della percentuale di errore di PSO rispetto alla soluzione ottima.

random e le **clustered** commettono errori piu' grandi, e le prime sono in media meno buone rispetto alle seconde. Questo mostra che la maggior parte della varianza riportata in Figura 5 e' dovuta al differente criterio di disposizione dei punti.

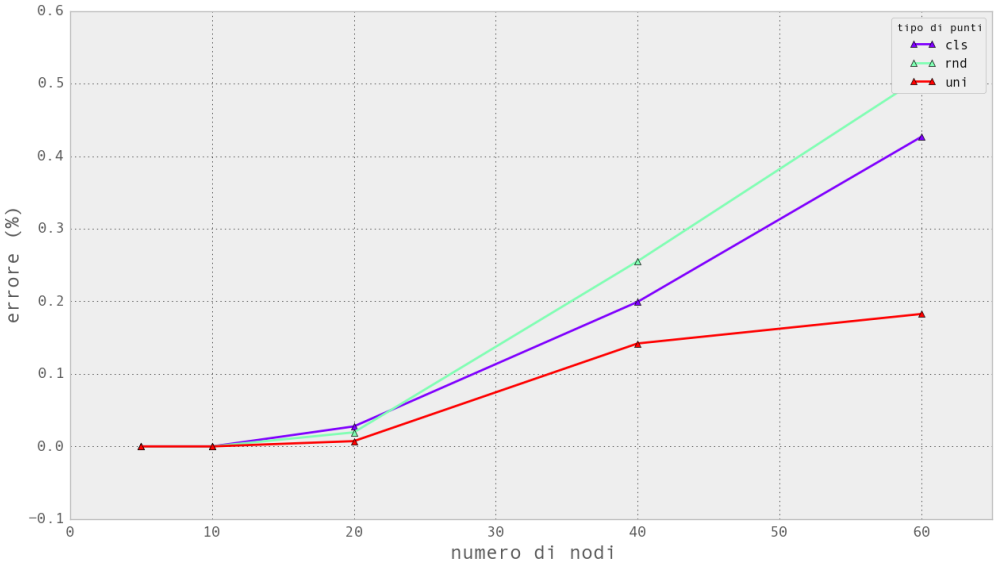

Figure 6: Media della percentuale di errore di PSO rispetto alla soluzione ottima, diviso per criterio di disposizione dei punti del problema.

In Figura 7 e' riportato il miglioramento della miglior soluzione trovata da PSO (in termini di funzione obiettivo) durante l'ottimizzazione. Per ottenere delle medie significative, il valore riportato e' calcolato come la percentuale dell'errore della soluzione corrente (all'iterazione i-esima) rispetto all'ottimo trovato da CPLEX. Si puo' vedere che per dimensionalita' basse PSO converge in fretta alla miglior soluzione (che generalmente e' ottima). Al crescere della dimensione del problema, la convergenza diventa piu' lenta, ma la discesa piu' veloce (anche perche' l'errore di partenza e' piu' alto). Infatti, per esempio per 40 nodi, si passa da un errore iniziale sull'ottimo di piu del 200%, e si arriva alla fine dell'ottimizzazione ad un errore del 20%. Questo e' significativo, mostra che l'ottimizzazione funziona, ma che non e' in grado di arrivare ad una soluzione buona.

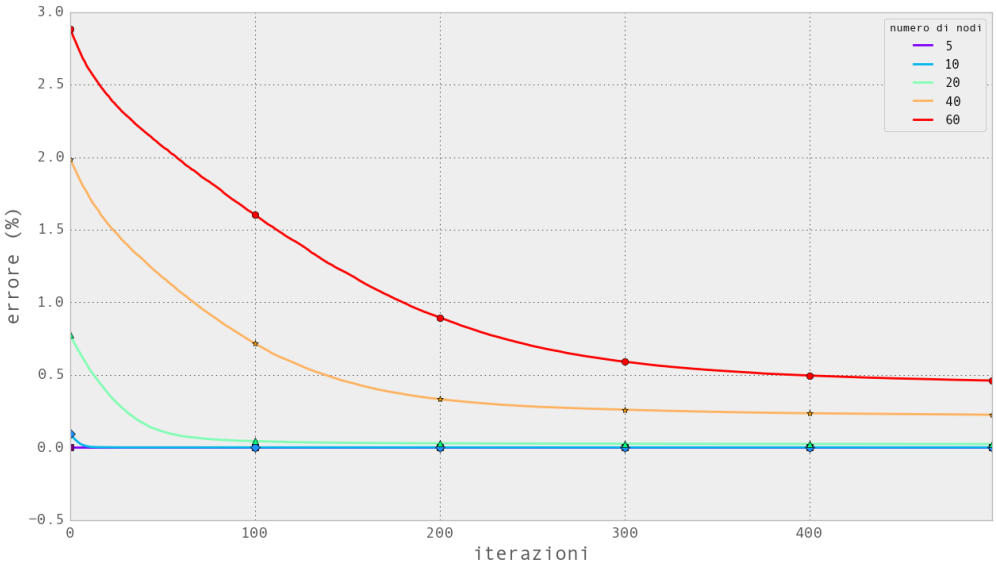

Figure 7: Percentuali di errore della miglior soluzione trovata finora rispetto all'ottimo, durante le iterazioni di PSO, divise per numero di nodi.

5 CONCLUSIONI

Durante l'implementazione di questa versione di PSO ho avuto modo di notare alcuni difetti. Infatti sono dell'opinione che in questo schema manchi una componente che permetta di esplorare la *neighborhood* senza necessariamente muoversi verso una delle soluzioni gia' visitate. Infatti con tale mancanza, una volta generate gli individui iniziali, tutte le modifiche alle velocita' "muovono" gli individui verso uno degli altri, e non c'e alcuna garanzia che l'ottimo stia in queste direzioni. Infatti i risultati mostrano che PSO non riesce a performare in maniera discreta oltre i 20 nodi, commettendo errori notevoli. Un miglioramento per tale metodo potrebbe essere quello di introdurre una componente nella formula di update delle velocita' che permetta di esplorare la *neighborhood* in direzioni promettenti (e.g.: greedy sub tour mutation (GSTM), Albayrak [5]).

Nonostante cio', ho potuto vedere che PSO e' piu' veloce di un metodo esatto quale CPLEX. In particolare, con l'aggiunta di un criterio di terminazione (e.g.: terminare l'ottimizzazione se la miglior soluzione trovata non migliora per 50 iterazioni), la metaeuristica terminerebbe rapidamente per dimensionalita' basse, con risultati discreti. Infatti CPLEX diventa lento al crescere del numero dei nodi, il che lo rende difficilmente applicabile a problemi molto grandi.

REFERENCES

[1] L. Brentegani, L. De Giovanni, and M. Di Summa. Memoc - esercitazione di laboratorio - parte 1. http://goo.gl/H5hiNu.

[2] K.P. Wang, L. Huang, C.G. Zhou, and W. Pang. Particle swarm optimization for traveling salesman problem. In *Machine Learning and Cybernetics*, 2003.

[3] H.X. Shi, Y.C. Liang, H.P. Lee, C. Lu, and Q.X. Wang. Particle swarm optimization-based algorithms for tsp and generalized tsp. *Information Processing Letters*, 2007.

[4] E. Gansner, E. Koutsofios, and S. North. Drawing graphs with dot. http://goo.gl/0TYrXz, 2006.

[5] Murat Albayrak and Novruz Allahverdi. Development a new mutation operator to solve the traveling salesman problem by aid of genetic algorithms. *Expert Systems with Applications*, 2011.