Esercitazione di laboratorio - Parte II

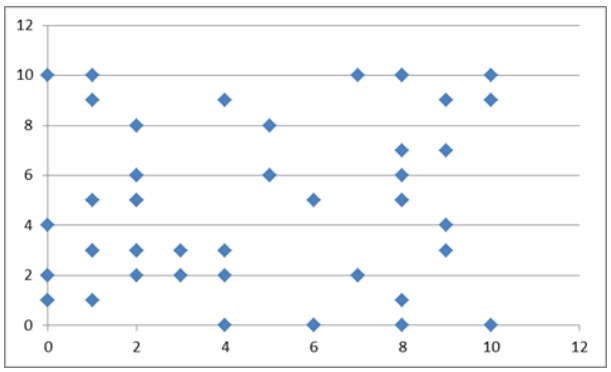
Introduzione

Nella seconda parte dell'esercitazione si vuole costruire un metodo alternativo per il problema presentato nella parte precedente tramite l'uso delle metaeuristiche. Brevemente, il problema consisteva nel trovare il percorso minimo per permettere di perforare una lastra nel più breve tempo possibile. Il problema è dunque riconducibile al trovare il più breve ciclo hamiltoniano di un grafo completo.

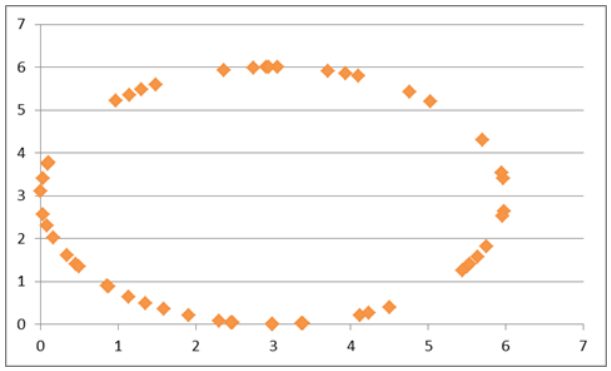
Disposizione dei nodi

L'algoritmo è stato utilizzato con una quantità variabile di punti. Questa quantità varia molto ed è visibile nella variabile *static const int* nel file *TSP.h*. Questo per poter vedere il comportamento del programma sia in presenza di pochi che di tanti nodi. Ovviamente, i nodi sono stati rappresentati come nella precedente parte dell'esercitazione (coordinate x e y).
Anche le disposizioni di tali punti sono rimaste invariate. Questo per poter confrontare tempi e risultati ottenuti con l'esercitazione precedente. Le disposizioni scelte per l'esercitazione sono:

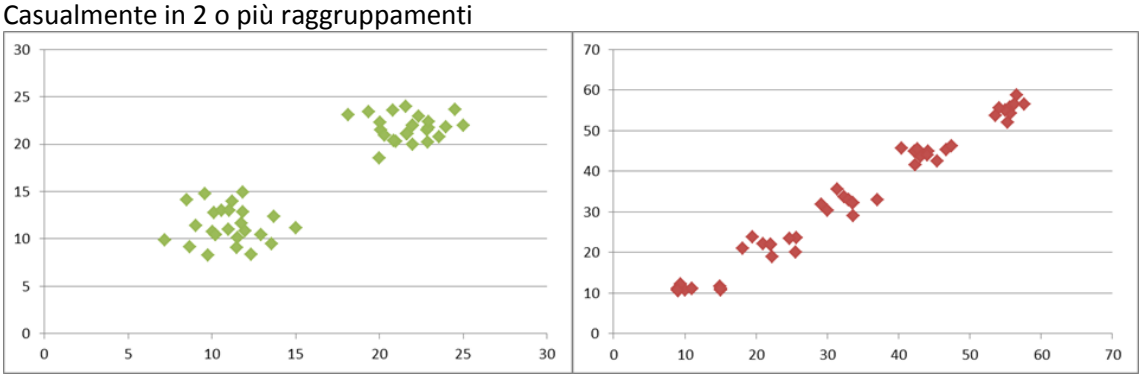
- Completamente casuale



- Casualmente lungo una circonferenza



- Casualmente in 2 o più raggruppamenti



Per le ultime due sono state utilizzate le coordinate polari, quindi la generazione è molto veloce.

Implementazione

I metodi utilizzati per l'esplorazione del vicinato sono stati i seguenti:

- Ricerca locale: analizza la funzione obiettivo di ogni vicino e mi sposto su un eventuale vicino migliore. In caso di assenza di vicini migliori, l'algoritmo termina.

- Tabu search: l'algoritmo si ricorda le ultime mosse eseguite. È necessaria in quanto, non terminando più l'algoritmo in assenza di vicini migliori, passo al migliore dei peggioranti. Nell'iterazione successiva però tornerei alla soluzione precedente in quanto sarebbe la migliore, e così via. Ciò si può evitare ricordando le ultime mosse fatte, in modo da non ripeterle. È stata usata una tabu-length pari a 7, in quanto una tabu-length troppo corta non impedisce comunque i cicli, mentre una tabu-length troppo lunga potrebbe bloccare troppi vicini. Da notare che vengono salvate le mosse e non le soluzioni in quanto vorrei poter, se necessario, tornare ad una soluzione precedente per utilizzare un'altra strada.
- Aspirazione: memorizzando mosse e non soluzioni, potrebbe succedere che la soluzione generata da una mossa tabu abbia delle caratteristiche che la rendano comunque interessante. Sarebbe quindi meglio considerare queste soluzioni nonostante siano state proibite.

Per una corretta esecuzione del programma è necessario conoscere alcuni parametri (tutti obbligatori, ma non necessariamente utilizzati):

- Numero dei nodi da utilizzare
- Disposizione dei punti (0=4 punti, 1=random, 2=circonferenza, 3=clusters, 4=leggi da file)
- Euristica (0=LS, 1=TS, 2=ATS)
- Tabu length (serve solo se il terzo parametro equivale a 1 o 2)
- Massimo numero di iterazioni (serve solo se il terzo parametro equivale a 1 o 2)
- Numero di cluster (serve solo se il secondo parametro equivale a 3)
- Tipo di soluzione iniziale (0=random, 1=best choice)

Ricerca della soluzione

Per la ricerca del vicinato è stato utilizzato il seguente schema:

- Soluzione iniziale: la soluzione di partenza è stata generata casualmente. Questo perché non sempre posso raggiungere l'ottimo scegliendo sempre il vicino migliore. In questo modo, posso utilizzare più punti di partenza e avere una maggiore probabilità di avvicinarmi all'ottino non locale. In alternativa si può costruire una soluzione iniziale scegliendo il migliore dei nodi rimanenti.
- Rappresentazione soluzione: data la semplicità, una soluzione è stata rappresentata dalla sequenza del ciclo (0-1-5-6-7-0). I vantaggi offerti sono:
  - Scambi eseguiti in tempo O(1).
  - Ammissibilità di ogni sequenza (ripetendo solamente il nodo iniziale/finale) visto che il grafo è completo.
- Applicazione: la funzione scelta per trovare i membri del vicinato consiste nello scambio di 2 archi del ciclo con 2 esterni . Anche se eseguiti anch'esse in tempo costante, l'aggiunta e la rimozione di archi non sono una scelta sensata per il nostro problema. Si è scelta inoltre una funzione che generasse un numero limitato di vicini, in modo da rendere la loro esplorazione molto veloce.
- Esplorazione del vicinato: come scritto in precedenza, l'esplorazione è stata eseguita tramite ricerca locale, tabu search e aspirazione.
- Valutazione: la valutazione dei vicini viene effettuata tramite confronto della funzione obiettivo (con il metodo scelto non posso avere soluzioni non ammissibili, quindi non è necessaria nessuna "penalizzazione"). In pratica viene c'è la funzione incaricata di controllare l'eventuale miglioramento della funzione obiettivo attuale scegliendo un certo vicino.

Tempi

Questi sono i tempi ottenuti utilizzando la semplice ricerca locale (tempi misurati in laboratorio). L'algoritmo termina quando non sono più presenti vicini miglioranti.

Numero Nodi	LS random	LS circonferenza	LS 2 cluster	LS 5 cluster
10	0,0001	0,00015	0,0001	0,00015
20	0,00018	0,00022	0,0002	0,00021
30	0,0003	0,00035	0,0003	0,00035

40	0,0045	0,0005	0,00055	0,00055
50	0,0007	0,0009	0,0007	0,0007
60	0,001	0,0013	0,001	0,0011
70	0,002	0,003	0,002	0,002
80	0,002	0,0035	0,003	0,003
90	0,003	0,004	0,0025	0,004
100	0,004	0,005	0,004	0,005
1000	4,3	4,5	3,5	3,2

Di seguito invece vengono riportati i risultati della tabu search. La condizione d'arresto, non più l'assenza do vicini miglioranti, è cambiata. Le possibilità sono ora quindi due, che utilizzano due parametri. La prima consiste semplicemente nel fornire un numero massimo di iterazioni come parametro (100 in questo caso).
La seconda invece consiste nel fermarsi quando non ci sono più vicini "legali" disponibili. Un altro importante parametro è infatti la lunghezza della tabu-list. È stata scelta una lunghezza pari a 7, non troppo lunga da bloccare troppi vicini, e nemmeno troppo corta da consentire comunque di ciclare. I parametri usati sono sempre stati *tabuLength*=8 e *maxiter*=110 per poter fare un migliore confronto (casi a parte sono state le esecuzioni con 10 nodi, per le quali la tabu-length è stata abbassata in quanto troppo alta).

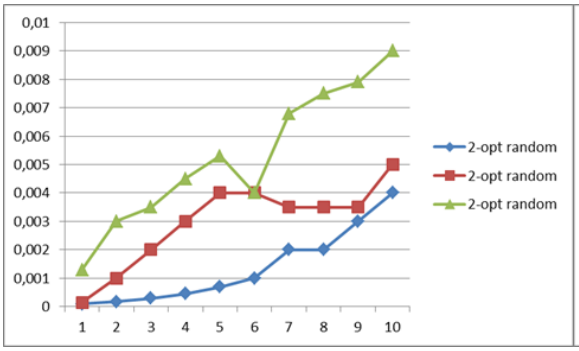
Numero Nodi	**TS random**	**TS circonferenza**	**TS 2 cluster**	**TS 5 cluster**
10	0,00015	0,00015	0,00015	0,00015
20	0,001	0,0015	0,0015	0,0015
30	0,002	0,002	0,002	0,002
40	0,003	0,004	0,003	0,003
50	0,004	0,003	0,004	0,004
60	0,004	0,003	0,004	0,004
70	0,0035	0,0035	0,0035	0,0035
80	0,0035	0,003	0,0035	0,004
90	0,0035	0,004	0,004	0,0045
100	0,005	0,0055	0,005	0,005
1000	0,41	0,413	0,4	0,401

Per finire vengono elencati i tempi utilizzati con la tecnica dell'aspirazione.

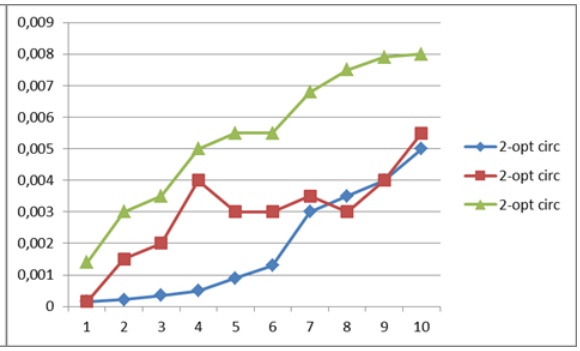
Numero Nodi	**TS random**	**TS circonferenza**	**TS 2 cluster**	**TS 5 cluster**
10	0,0013	0,0014	0,0013	0,0015
20	0,003	0,003	0,003	0,003
30	0,0035	0,0035	0,0035	0,0035
40	0,0045	0,005	0,0045	0,005
50	0,0053	0,0055	0,0055	0,0055
60	0,004	0,0055	0,004	0,006
70	0,0068	0,0068	0,0068	0,0068
80	0,0075	0,0075	0,007	0,007
90	0,0079	0,0079	0,0077	0,0079
100	0,009	0,008	0,0085	0,009
1000	0,4	0,38	0,382	0,36

È stata fatta una prova anche con un numero molto elevato di nodi per monitorare il comportamento del programma in queste condizioni. Usando la tabu-search (con eventuale aspirazione), il

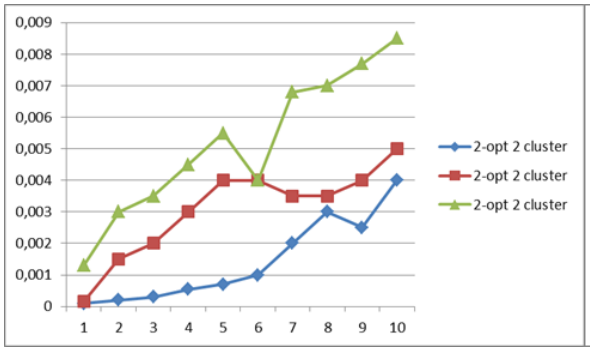
tempo ottenuto è di circa mezzo secondo. Un tempo molto più alto è stato impiegato per la ricerca locale. Questo perché il massimo numero di iterazioni non viene ovviamente usato come criterio di stop, e quindi la ricerca locale continuerà finché non verrà trovato più alcun vicino migliorante.
Sono stati eseguiti test anche con la soluzione iniziale alternativa. I risultati sono molto simili (leggermente più bassi), e sono dovuti al fatto che la soluzione iniziale era probabilmente migliore rispetto a quella casuale e sono stati necessari meno scambi.



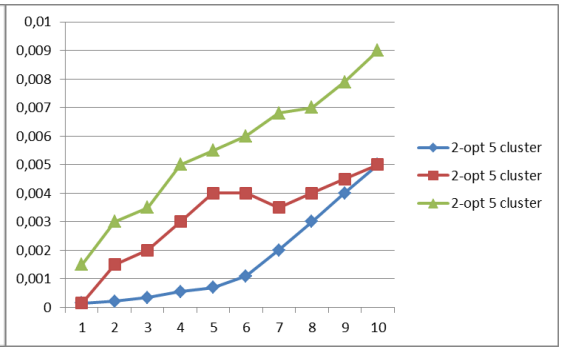
Tempi su random



Tempi su circonferenza



Tempi con 2 cluster



Tempi con 5 cluster

Confronto tempi parte I e parte II

Verranno ora confrontate le tempistiche misurate dalla prima parte con quelle misurate in questa esercitazione. Come si può vedere, le ricerche di questa parte di esercitazione utilizzano un tempo relativamente basso anche con un numero elevato di nodi. Questo dipende principalmente da due motivi. Il primo consiste nel numero limitato di iterazioni nel caso della tabu search. Non dovendo più necessariamente trovare l'ottimo vero e proprio, ma solamente un valore "indicativo", non serve arrivare ad un numero altissimo di iterazioni.
La seconda consiste nel fatto che non è eseguito codice CPLEX per le ottimizzazioni. È usato del semplice codice C++ che esegue degli scambi e calcola delle somme in breve tempo.

Numero Nodi	Modello	2-opt	2-opt con tabu-list	2-opt con aspirazione
10	1	0,0001	0,00015	0,0013
20	1	0,00018	0,001	0,003
30	1	0,0003	0,002	0,0035
40	1	0,0045	0,003	0,0045
50	15	0,0007	0,004	0,0053
60	25	0,001	0,004	0,004
70	45	0,002	0,0035	0,0068
80	55	0,002	0,0035	0,0075
90	480	0,003	0,0035	0,0079
100	600	0,004	0,005	0,009

1000	-	4,3	4,5	3,5

Tempistiche con una disposizione dei punti casuale

Numero Nodi	Modello	2-opt	2-opt con tabu-list	2-opt con aspirazione
10	1	0,00015	0,00015	0,0014
20	2	0,00022	0,0015	0,003
30	2	0,00035	0,002	0,0035
40	2	0,0005	0,004	0,005
50	2	0,0009	0,003	0,0055
60	3	0,0013	0,003	0,0055
70	5	0,003	0,0035	0,0068
80	20	0,0035	0,003	0,0075
90	40	0,004	0,004	0,0079
100	55	0,005	0,0055	0,008
1000	-	4,5	3,5	3,2

Tempistiche con una disposizione dei punti casuale lungo una circonferenza

Numero Nodi	Modello	2-opt	2-opt con tabu-list	2-opt con aspirazione
10	1	0,0001	0,00015	0,0013
20	1	0,0002	0,0015	0,003
30	2	0,0003	0,002	0,0035
40	5	0,00055	0,003	0,0045
50	35	0,0007	0,004	0,0055
60	100	0,001	0,004	0,004
70	200	0,002	0,0035	0,0068
80	350	0,003	0,0035	0,007
90	-	0,0025	0,004	0,0077
100	-	0,004	0,005	0,0085
1000	-	3,5	3,2	0,41

Tempistiche con una disposizione dei punti casuale in 2 cluster

Numero Nodi	Modello	2-opt	2-opt con tabu-list	2-opt con aspirazione
10	1	0,00015	0,00015	0,0015
20	1	0,00021	0,0015	0,003
30	2	0,00035	0,002	0,0035
40	10	0,00055	0,003	0,005
50	55	0,0007	0,004	0,0055
60	350	0,0011	0,004	0,006
70	750	0,002	0,0035	0,0068
80	-	0,003	0,004	0,007
90	-	0,004	0,0045	0,0079
100	-	0,005	0,005	0,009
1000	-	3,2	0,41	0,413

Tempistiche con una disposizione dei punti casuale in 5 cluster

Confronto risultati parte I e parte II

Sono stati fatti infine dei test per poter confrontare i risultati ottenuti tramite le due realizzazioni. Per poterli eseguire sono stati generati dei file di testo contenenti le coordinate dei punti da usare, che sono stati quindi letti dai programmi e salvati.
Per i test si è deciso di utilizzare una disposizione casuale di 80 punti e 30 punti suddivisi in 2 cluster. La scelta deriva dal fatto che l'algoritmo esatto termina in un tempo accettabile con la prima disposizione di nodi, mentre nel secondo test, un numero alto di nodi avrebbe richiesto tempi abbastanza.
I parametri utilizzati per la tabu list sono stati calibrati prima dell'esecuzione del programma. Nel primo test vengono usati *tabuLength*=15 e *maxiter*=210. Nel secondo test invece abbiamo *tabuLength*=25 e *maxiter*=110.

	Risultato	Tempo	Errore
Modello	75,4979	84,4011	0
LS	78,6892	0,00258493	4,055575606
Tabu	77,527	0,0123169	2,617281721
Aspirazione	76,763	0,012619	1,648059612

Risultati con una disposizione di 80 punti casuale

	Risultato	Tempo	Errore
Modello	59,8062	8,32829	0
LS	63,6385	0,000218153	6,021983548
Tabu	60,3453	0,000463009	0,893358721
Aspirazione	59,8063	0,00520706	0,000167206

Risultati con una disposizione di 30 punti casuale in 2 cluster

Per ogni test è stato salvato il risultato, il tempo impiegato (in secondi) e l'errore relativo rispetto al valore trovato col modello della prima esercitazione. Come ci si poteva aspettare, il risultato peggiore è stato ottenuto con la ricerca locale, in quanto fermandomi quando non trovo più soluzioni migliori, non potrò raggiungere l'eventuale ottimo. Risultati molto migliori sono stati ottenuti invece con la tabu-search e con l'uso dell'aspirazione.