UNIVERSITÀ DEGLI STUDI DI MILANO
FACOLTÀ DI SCIENZE MATEMATICHE,
FISICHE E NATURALI

Progetto d'esame per il corso di
Algoritmi Euristici

Due euristiche per HTSP a confronto

Prof. **Giovanni Righini**
Alberto Ceselli

PROGETTO di:
Federico Gandellini
Matricola no : 703156

Anno accademico 2012/2013

Indice

Il problema

Lo *Hierarchical Traveling Salesman Problem* è un caso particolare del *Travelling Salesman Problem,* nei paragrafi seguenti vedremo una breve descrizione dei due algoritmi.

Il TSP

Il *Travelling Salesman Problem* (in seguito TSP) può essere definito come segue. Sia $G=(V,A)$ un grafo dove $V=\{v_i,\dots,v_n\}$ è l'insieme dei vertici e $A=\{(v_i,v_j):v_i,v_j\in V\}$. Ad ogni arco (v_i,v_j) è associato un costo c_{ij} non negativo.

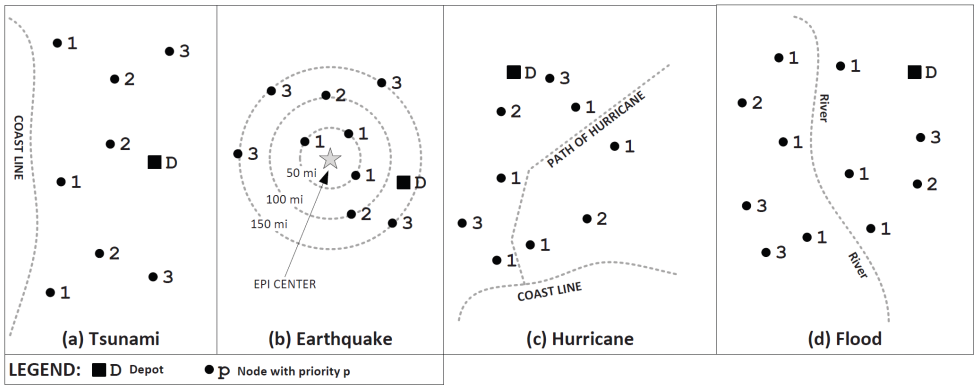
Il problema consiste nel trovare un percorso che visiti tutti i nodi del grafo G esattamente una volta e che minimizzi la distanza totale percorsa. Un ciclo di questo tipo è detto *Hamiltoniano.*

Esistono alcune proprietà dalle quali derivano varianti del problema stesso: esso è definito *simmetrico* se $c_{ij}=c_{ji}$ per ogni i,j e soddisfa la *disuguaglianza triangolare* se $c_{ij}+c_{jk}\geq c_{ik}$ per ogni i,j,k .

L'HTSP

Lo *Hierarchical Traveling Salesman Problem* (HTSP nel seguito) è una variante del TSP. Ad ogni nodo i si associa un livello di priorità $p_i\in\{1..\mathrm{P}\}$ e si fissa un parametro k non negativo che

regola la precedenza dei livelli di priorità da ottenere nella soluzione: prima di visitare un nodo a priorità p_i , dovranno essere visitati tutti i nodi con priorità $p_i - k$. Il problema consiste nel trovare il *tour hamiltoniano* che visiti tutti i nodi del grafo rispettando l'ordine delle priorità dei vari nodi. Possiamo affermare che l'HTSP è un problema *NP-hard*, in quanto caso particolare del TSP, anch'esso *NP-hard*. L'HTSP si applica a problemi reali quali l'organizzazione dei soccorsi in zone colpite da calamità naturali. A seconda del tipo di calamità si possono identificare aree più o meno colpite, con conseguente necessità di aiuti più o meno urgente. Assegnando ad ogni zona (nodo) un valore di priorità si ottengono mappe simili a quella nella figura sottostante.



Nel caso di *tsunami* (a) la zona più colpita sarà quella vicina alla costa, nel caso di *terremoto* (b) sarà quella vicina all'epicentro del sisma, nel caso di *tornado* (c) essa seguirà il percorso del ciclone stesso ed in modo simile, in caso di *esondazione* di un fiume () la zona più colpita sarà quella vicina al letto del corso d'acqua.

L'algoritmo

Come abbiamo anticipato, l'HTSP è una variante del TSP con vincoli aggiuntivi sull'ordine di visita dei nodi. Le euristiche proposte per la risoluzione dell'HTSP utilizzano come sotto procedura un algoritmo noto in letteratura come GENIUS per ottenere la soluzione del problema TSP associato. L'idea alla base di entrambe le nostre euristiche è costruire una soluzione per l'HTSP a partire da una o più soluzioni per TSP. Tali soluzioni risultano inammissibili per HTSP e vengono manipolate per costruirne una ammissibile. L'effetto finale è che si perde ottimalità ma si guadagna ammissibilità.

Iniziamo ora esponendo l'euristica GENIUS e proseguiamo con le euristiche da noi proposte mostrando come GENIUS giochi un ruolo fondamentale nell'ottimizzazione dei vari sotto-problemi generati durante l'esecuzione degli algoritmi.

GENIUS

GENIUS è un algoritmo a due fasi per la risoluzione del TSP. La prima procedura, chiamata *GENI*, si occupa della costruzione di una soluzione ammissibile, successivamente interviene *US,* il cui compito è

eseguire una *post-ottimizzazione* della soluzione trovata al passo precedente con lo scopo di migliorarla. Nei seguenti paragrafi esamineremo nel dettaglio entrambe le procedure.

GENI

Questa euristica costruttiva si applica a problemi simmetrici o asimmetrici, la sua particolarità è il fatto che l'inserimento di un nuovo nodo v non avviene necessariamente tra due nodi inizialmente adiacenti. A fronte dell'operazione di inserimento i due nodi saranno adiacenti a v . Per un dato orientamento del tour siano v_k un vertice sul percorso da v_j a v_i e v_l un vertice sul percorso da v_i a v_j . Per ogni vertice v_h definiamo poi v_{h-1} il suo predecessore e v_{h+1} il suo successore.

L'algoritmo GENI consiste in tre passi principali:

Passo 1: Viene creato un *tour* iniziale di tre vertici scegliendoli casualmente dall'insieme dei nodi N e si inizializzano le *p*-neighborhoods per questi vertici.

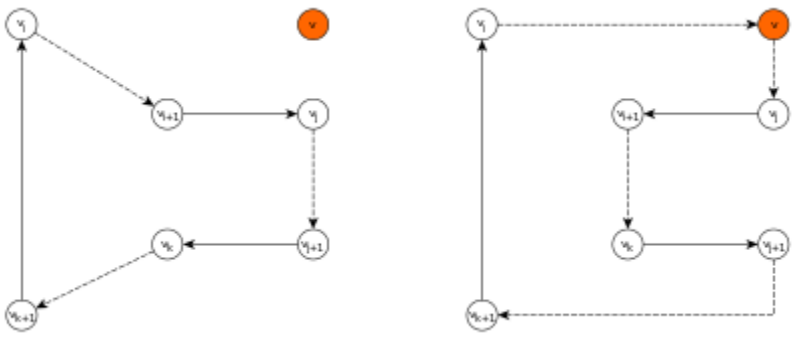
Passo 2: Si sceglie un nodo v non presente nel tour e si utilizza una procedura di inserimento per aggiungervelo. Successivamente si aggiornano le *p*-neighborhoods.

Passo 3: Se il tour in costruzione contiene tutti i nodi di N l'algoritmo termina, altrimenti si itera dal passo 2.

L'operazione di inserimento di un nuovo vertice v tra v_i e v_j può avvenire in uno dei due modi illustrati di seguito.

Type I Insertion

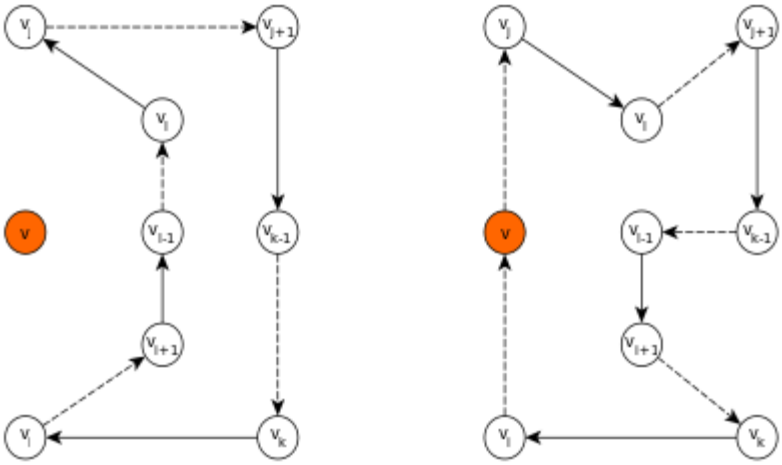
Scelti $v_k \neq v_i$ e $v_k \neq v_j$, l'inserimento del vertice v nel tour consiste nell'eliminazione degli archi (v_i, v_{i+1}), (v_j, v_{j+1}) e (v_k, v_{k+1}) e nell'inserimento degli archi (v_i, v), (v, v_j), (v_{i+1}, v_k) e (v_{j+1}, v_{k+1}) al loro posto. Questo implica che i due *path* $(v_{i+1}, \ldots, v_j)$ e $(v_{j+1}, \ldots, v_k)$ vengano invertiti. La figura sottostante mostra un esempio di tale operazione di inserimento.



Type II Insertion

Scelti $v_k \neq v_j$, $v_k \neq v_{j+1}$, $v_l \neq v_i$ e $v_l \neq v_{i+1}$, l'inserimento del vertice v nel tour consiste nell'eliminazione degli archi (v_i, v_{i+1}),

(v_{l-1}, v_l) , (v_j, v_{j+1}) , (v_{k-1}, v_k) e nell'inserimento degli archi (v_i, v) , (v, v_j) , (v_l, v_{j+1}) , (v_{k-1}, v_{l-1}) e (v_{i+1}, v_k) al loro posto. Anche in questo caso dobbiamo invertire i due *path* $(v_{i+1}, ..., v_{l-1})$ e $(v_l, ..., v_j)$. La figura seguente mostra un esempio di tale operazione di inserimento.



Dovendo scegliere quattro nodi candidati per eseguire l'operazione di inserimento, il costo di questo algoritmo è nell'ordine di n^4 . Per limitare il numero di possibili candidati da valutare, e quindi limitare il tempo richiesto per l'inserimento, usiamo la seguente strategia.

Per ogni vertice $v \in V$ definiamo un insieme $N_p(v)$ chiamato p-neighborhood di v contenente i p vertici sul tour più vicini a v (in termini di costo). Definiti questi insiemi, possiamo scegliere $v_i, v_j \in N_p(v)$, $v_k \in N_p(v_{i+1})$ e $v_l \in N_p(v_{j+1})$. Risultati sperimentali hanno dimostrato che per ottenere un buon rapporto tra

qualità della soluzione e velocità dell'algoritmo, è consigliabile utilizzare $p \in \{4,5,6\}$.

US

La soluzione calcolata con GENI può essere migliorata utilizzando un procedura di post-ottimizzazione chiamata US che si basa su due operazioni fondamentali dalle quali prende il nome: *Unstringing* e *Stringing*. La sotto-procedura di *Stringing* si occupa di aggiungere un nodo ad un tour ed è identica alla procedura di inserimento descritta nell'algoritmo GENI. La contrario, lo scopo della procedura di *Unstringing* è di rimuovere un nodo da un tour.

L'algoritmo US è costituito dalle seguenti operazioni:

Passo 1: Si considera un tour iniziale x di costo z e si impostano $x^* = x, z^* = z$ e $t = 1$.

Passo 2: Partendo dal tour x si applicano in sequenza le procedure di *Unstringing* e *Stringing* sul vertice v_t , in questo modo si ottiene il nuovo tour x' di costo z' . Si impostano quindi $x = x', z = z'$.

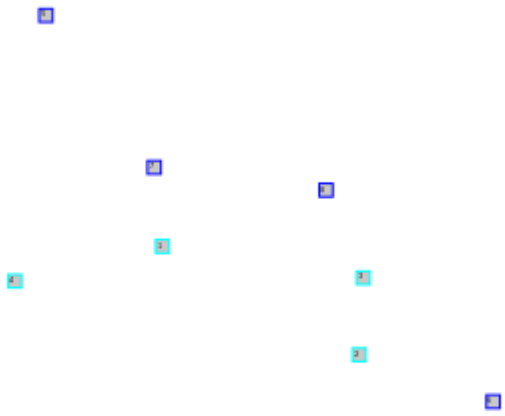
- Se $z < z^*$, si impostano $x^* = x, z^* = z, t = 1$ e si ripete dal Passo 2.

- Se $z \geq z^*$ si imposta $t = t + 1$.

- Se $t = n + 1$ l'algoritmo termina restituendo x^* come tour ottimo trovato e z^* come costo. In caso contrario si ripete dal Passo 2.

Due algoritmi per HTSP

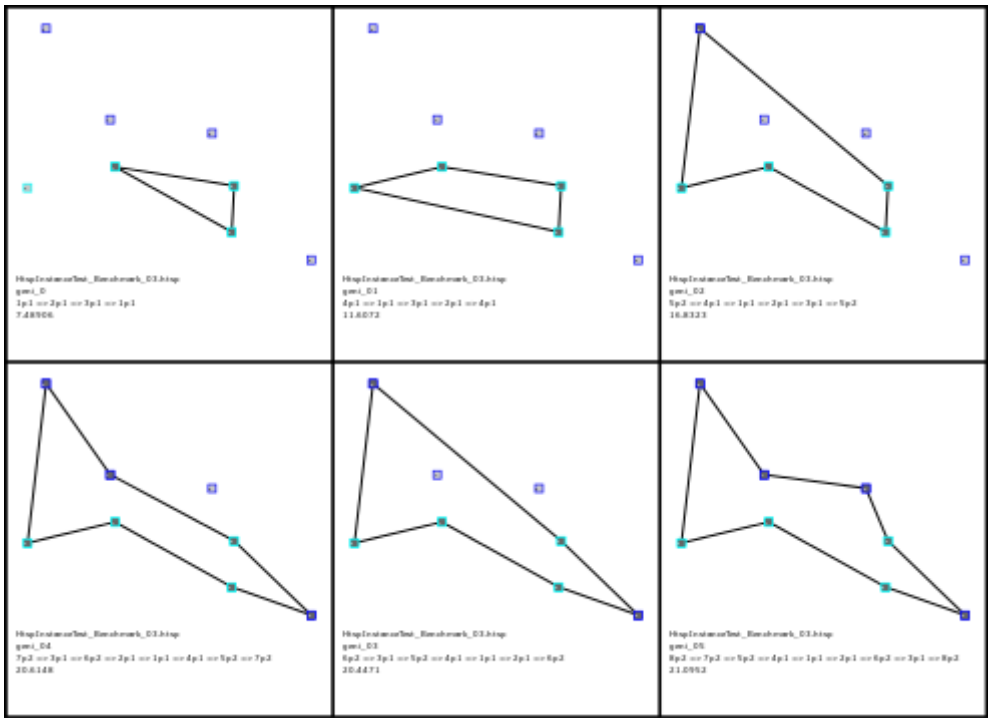
Per la risoluzione dell'HTSP abbiamo realizzato due differenti euristiche che sono accomunate dall'utilizzo dell'algoritmo GENIUS come *solver* per il sotto-problema TSP generato. Le differenze tra i due algoritmi risiedono nel modo di ottenere il sotto-problema e di ricombinare le soluzioni parziali per ottenere una soluzione valida per TSP. L'idea quindi è abbandonare l'ottimalità di una soluzione non ammissibile (ottenuta con GENIUS) e muoversi verso una soluzione *sub-ottima* ma ammissibile. In figura è riportato un grafico di una istanza con otto nodi e due livelli di priorità (azzurro per priorità 1 e blu per priorità 2) utilizzata per i test, nel seguito del capitolo mostreremo i passi compiuti dai due algoritmi proposti su tale istanza. Per il problema di test abbiamo utilizzato un valore $k=0$.



HTSP Solver Type I

Il primo *solver* implementato utilizza GENIUS per risolvere un problema TSP sull'intero insieme N dei nodi del problema.

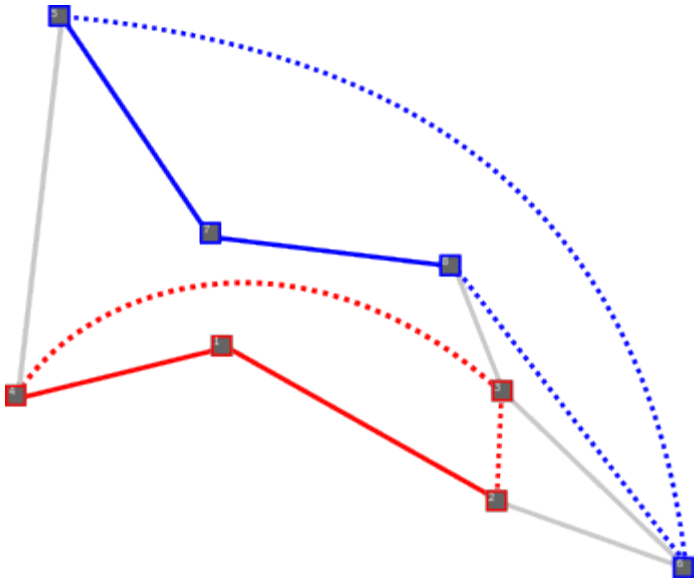
La figura seguente mostra i passi compiuti dall'algoritmo GENI per ottenere la prima parte della soluzione TSP.



In questo esempio, la procedura di ottimizzazione US, ha ottenuto esattamente lo stesso risultato riportato da GENI, quindi non abbiamo incluso alcuna immagine relativa all'elaborazione di US.

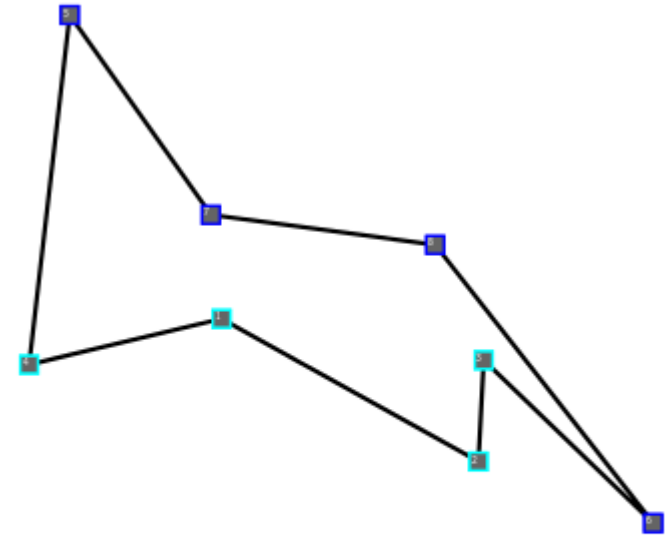
A partire dalla soluzione iniziale mostrata, vengono estratti i *subtour* già ammissibili per il problema finale, quelli cioè che contengono nodi che rispettano l'ordine di priorità previsto dai parametri p_i e k .

La figura seguente mostra i due *subtour* trovati dall'algoritmo rispetto al problema di esempio.



Nello schema sono riportati con linee continue rosse e blu le porzioni di tour già valide e mantenute, con linee tratteggiate gli archi inseriti per chiudere i *subtour* e renderli cicli *Hamiltoniani* validi e in grigio il percorso iniziale trovato da GENIUS.

I *subtour* ottenuti vengono collegati in modo incrementale partendo dal livello di priorità minimo fino ad arrivare al massimo, in modo da preservare la regola di precedenza. La procedura di *join* dei due *subtour* sceglie gli archi da rimuovere e quelli da inserire cercando la doppia coppia che massimizza il rapporto tra costo della coppia di archi rimossa e costo della coppia di archi inserita. Il risultato finale dell'elaborazione è mostrato nella figura seguente.

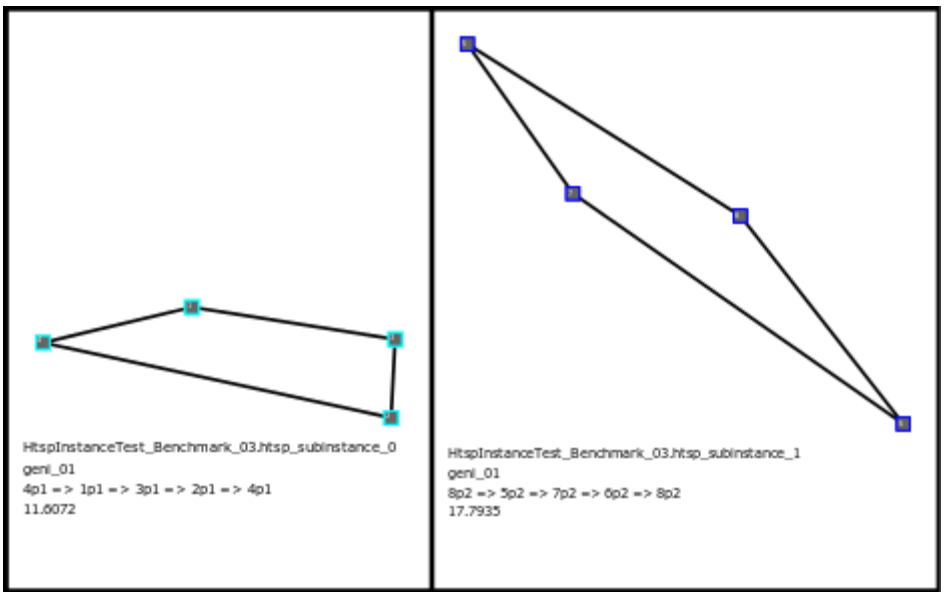


Possiamo notare l'operato dell'algoritmo: l'arco $(v_{4,}v_3)$ è stato sostituito con l'arco $(v_{4,}v_5)$ e l'arco $(v_{6,}v_5)$ è stato sostituito con $(v_{6,}v_3)$. La sostituzione degli archi $(v_{3,}v_2)$ e $(v_{8,}v_6)$ è stata valutata ma è risultata meno vantaggiosa in termini di costo. Il valore totale del tour risultante per il problema trovato dal primo *solver* proposto è 22,73.

HTSP Solver Type II

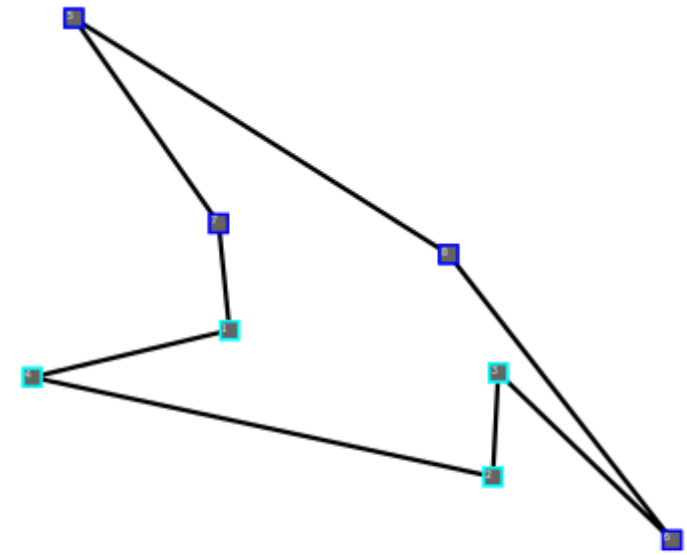
Il secondo algoritmo realizzato partiziona l'insieme iniziale di nodi in sottoinsiemi contenti nodi con priorità *compatibili*, che sono cioè equivalenti in termini di precedenza nella soluzione finale.

Successivamente viene eseguito l'algoritmo GENIUS su ogni sottoinsieme, ottenendo così tante soluzioni per TSP quanti sono i sottoinsiemi, i tour ottenuti sono tutti auto-consistenti. Le figure sottostanti mostrano i risultati per i sotto-problemi TSP per i due sottoinsiemi di nodi trovati dall'algoritmo. Ogni tour collega nodi con lo stesso livello di priorità.



Per ottenere la soluzione finale, i tour vengono collegati tra loro con la stessa procedura illustrata nel primo *solver*. In questo caso, però, gli archi sono tutti candidati ad essere rimossi e sostituiti.

Il risultato che si ottiene è mostrato nella figura seguente.



Come si può notare la soluzione trovata è diversa rispetto a quella trovata dal primo *solver*, ma comunque rispetta il vincolo di priorità del problema. In questo caso il valore del tour risultante è 24.13.

Possiamo quindi affermare che rispetto al piccolo problema preso in considerazione, il primo algoritmo ha trovato una soluzione migliore del secondo.

Confronto dei risultati ottenuti

Per eseguire i test abbiamo generato le istanze facendo variare i seguenti parametri:

- **Vertici**: 100, 200, 300, 400, 500
- **P**: 5, 10
- **k**: 0, 1, 2, 3

Per ogni combinazione di parametri sono state generare 5 istanze disponendo i vertici in modo casuale.

Gli algoritmi sono stati codificati utilizzando il linguaggio C++, ed i test sono stati eseguiti su una macchina con un processore Intel® Core™ i7-3632QM 64 Bit a 2.20GHz.

Per ogni combinazione di parametri abbia eseguito la media sulle 5 istanze generate, per cercare di avere un valore più attendibile. Al fine di studiare eventuali correlazioni tra i parametri, abbiamo raggruppato questi valori medi in tabelle distinte.

Nelle tabelle identificate con A, per ogni dimensione e livello di priorità abbiamo calcolato la media dei valori ottenuti (per costi e tempi rispettivamente) per tutti i valori di k.

Nelle tabelle identificate con B abbiamo invece riportato la media su tutti i valori di P per ogni dimensione e valore di k.

	P	
Vertici	**5**	**10**
100	155,733	206,281
200	213,697	301,808
300	260,733	372,125
400	302,846	431,794
500	335,389	481,204

Tab. A – Type I
Media costi soluzioni

	P	
Vertici	**5**	**10**
100	141,630	186,927
200	192,504	263,846
300	231,221	318,430
400	267,841	366,097
500	298,566	404,884

Tab. A – Type II
Media costi soluzioni

	P	
Vertici	**5**	**10**
100	0,94	1,08
200	10,59	8,55
300	42,04	45,64
400	95,51	140,87
500	208,25	251,98

Tab. A – Type I
Media tempi

	P	
Vertici	**5**	**10**
100	0,30	0,21
200	2,27	1,60
300	6,99	5,76
400	19,79	13,68
500	40,08	29,28

Tab. A – Type II
Media tempi

	k			
Vertici	**0**	**1**	**2**	**3**
100	255,924	181,726	150,170	136,208
200	367,686	258,864	210,714	193,746
300	456,773	314,159	257,958	236,827
400	533,216	365,526	297,383	273,157
500	582,889	407,438	336,377	306,481

Tab. B – Type I
Media costi soluzioni

	k			
Vertici	**0**	**1**	**2**	**3**
100	1,15	1,13	0,99	0,79
200	10,03	9,26	9,50	9,49
300	45,90	41,74	40,45	47,28
400	121,04	120,02	99,00	132,69
500	207,60	228,69	245,70	238,46

Tab. B – Type I
Media tempi

	k			
Vertici	**0**	**1**	**2**	**3**
100	228,714	163,277	138,714	126,408
200	317,949	228,450	188,907	177,395
300	379,566	274,037	232,259	213,439
400	438,988	316,424	266,966	245,499
500	484,572	350,568	298,058	273,702

Tab. B – Type II
Media costi soluzioni

	k			
Vertici	**0**	**1**	**2**	**3**
100	0,20	0,20	0,27	0,35
200	1,53	1,74	2,34	2,13
300	5,31	5,81	7,03	7,37
400	12,91	13,55	18,44	22,05
500	24,98	31,53	36,50	45,71

Tab. B – Type II
Media tempi

Conclusioni

I risultati ottenuti dai test effettuati mostrano che il *solver Type II* ha fornito risultati migliori su tutte le istanze esaminate, sia in termini di costo della soluzione sia per quanto riguarda i tempi impiegati ad ottenerle. In termini di costo delle soluzioni ottenute il miglioramento varia dall'8% al 18%, mentre in termini di tempo di calcolo il vantaggio è maggiore e si va dal 56% al 90% circa.

Durante i test abbiamo riscontrato in alcuni casi, specialmente in presenza di istanze molto piccole, che l'algoritmo *Type I* ottenesse risultati migliori del *Type II*, ne è un esempio l'istanza da 8 nodi utilizzata per mostrare il funzionamento dell'algoritmo. In ogni caso, per istanze di dimensioni significative, il *solver Type II* si è rivelato costantemente più efficace.

Concludendo possiamo affermare che la variazione dei parametri P e k non influenza il rapporto tra i due algoritmi.