Problema del Busy Traveller

Pasquale Caporaso 0286761 - Ezio Emanuele Ditella 0286766

1 Descrizione del problema

Siamo un turista in visita in un città, il nostro obiettivo è visitare il maggior numero possibile di monumenti popolari nel tempo a nostra disposizione.
Dati/Input:

1. $\mathcal{S}$: Insieme di monumenti;
2. p_k: Grado di popolarità di un monumento ($k \in [1, |\mathcal{S}|]$);
3. d_{ij}: Distanza tra due monumenti ($i, j \in [1, |\mathcal{S}|]$);
4. T: Tempo a disposizione del turista per la visita;

Output:

1. $\mathcal{I}$:Sottoinsieme ordinato di $\mathcal{S}$ contenente l'itinerario con i monumenti di maggiore popolarità che il turista può visitare con il suo limite di tempo;

Precisazioni:

- Ogni monumento è raggiungibile da ogni altro monumento;
- La distanza tra due monumenti è simmetrica ($d_{ij} = d_{ji}$);
- La posizione iniziale del turista è unica e dalla partenza è in grado di raggiungere ogni monumento;
- Il valore migliore di popolarità è quello con il valore maggiore;

2 Modello PLI

Pensiamo ad ogni monumento come un nodo di un grafo e il loro collegamento come un arco, aggiungiamo in tale grafo anche la posizione iniziale del turista, indiachiamo tale nodo con indice 0. La distanza tra i monumneti sarà il peso di un arco sul grafo, il valore della popolarità sarà associato invece ai nodi, diamo al nodo con la posizione iniziale dell'utente popolarità 0.

Usiamo le seguenti varibili decisionali:

$$x_{ij} = \begin{cases} 1, & \text{se arco } (i,j) \text{ fa parte dell'itinerario} \\ 0, & \text{altrimenti} \end{cases}$$

Questa variabile indica se nell'itinerario il turista farà il percorso dal monumento i al monumento j, precisiamo che, in generale, $x_{ij} \neq x_{ji}$

Usiamo inoltre le sequenti variabili ausiliari:

$$y_i = \begin{cases} k, & k \in [1, |\mathcal{S}|], \text{ se il nodo } i \text{ fa parte dell'itinerario} \\ 0, & \text{altrimenti} \end{cases}$$

$$h_i = \begin{cases} 1, & \text{se } y_i \neq 0 \\ 0, & \text{altrimenti} \end{cases}$$

Le varibili y indicano la posizione di ogni monumneto nella sequenza dell'itinerario, mentre le h ci serviranno per calcolare accuratamente la popolarità nella funzione obiettivo.

Detto ciò il modello PLI è il segeunte ($n = |\mathcal{S}|$):
Funzione obiettivo:

$$\max \sum_{i=0}^{n} h_i p_i$$

Vincoli:

(1) $\sum_{i=0}^{n} \sum_{j=0}^{n} x_{ij} d_{ij} \leq T$ Non possiamo viaggiare più del tempo a disposizione

(2) $\sum_{i=0}^{n} x_{ij} \leq 1 \ (i \neq j) \ \forall j$ Solo un arco entrante attivo per nodo

(3) $\sum_{j=0}^{n} x_{ij} \leq 1 \ (i \neq j) \ \forall i$ Solo un arco uscente attivo per nodo

(4) $\mathrm{y}_0 = 1$ Il primo nodo è quello del turista

(5) $\mathrm{y}_i \leq M \sum_{j=0}^{n} x_{ji} \ \forall i \neq 0$ Nessuna y può attivarsi se non ha almeno un nodo entrante

(6) $\mathrm{x}_{ij} \leq y_i \ \forall (i,j)$ Nessuna y può avere nodi uscenti se è uguale a 0

(7) $\mathrm{y}_j \geq y_i + 1 - M(1 - x_{ij}) \ \forall (i,j)$ Le y devono essere sequenziali se gli archi sono attivi

(8) $\mathrm{h}_i \leq y_i \ \forall i$ $h = 1$ se la y corrispettiva è attiva

Con questa rappresentazione sia il numero di vincoli che il numero di variabili sono un $O(n^2)$.

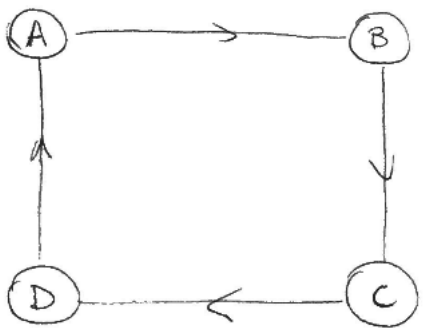
3 Descrizione delle soluzioni ammissibili

Una soluzione ammissibile per questo modello sarà un path che parte dal nodo del turista (indicato dall'indice 0) e si conclude in un qualsiasi altro nodo del grafo.

3.1 Impossibilità dei cicli per una soluzione ammissibile

Proposizione In una soluzione ammissible del problema non è possibile avere cicli

Dimostrazione Consideriamo una soluzione avente un ciclo, come mostrato nella seguente figura (le lettere all'interno dei nodi rapprentano gli indici associati a essi):



Con questa configurazione i valori delle varibiali decisionali sono:

$$x_{AB} = 1 \quad x_{BC} = 1 \quad x_{CD} = 1 \quad x_{DA} = 1$$

E quindi, per il set di vincoli **(7)**, dovranno valere le seguenti relazioni:

$$y_B \geq y_A + 1 \quad \text{perché} \quad x_{AB} = 1$$
$$y_C \geq y_B + 1 \quad \text{perché} \quad x_{BC} = 1$$
$$y_D \geq y_C + 1 \quad \text{perché} \quad x_{CD} = 1$$
$$y_A \geq y_D + 1 \quad \text{perché} \quad x_{DA} = 1$$

Mettendo tutto insieme abbiamo:

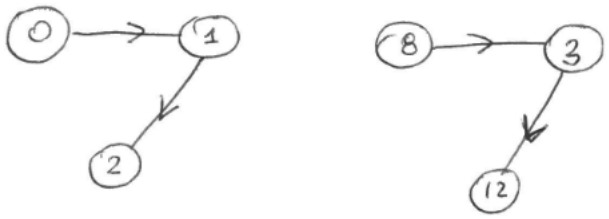
$$y_A \geq y_D + 1 \geq y_C + 2 \geq y_B + 3 \geq y_A + 4$$

Il che è impossibile. Questo ragionamento può essere applicato per cicli di qualunque lunghezza e indipentendemente dai nodi che lo compongono.

3.2 Inammissiblità di soluzioni con due o più path differenti

Proposizione In una soluzione ammissibile del problema deve essere presente un singolo path

Dimostrazione Assumiamo, per assurdo, che esista una soluzione ammissibile con due path come la seguente:



Con questa configurazione i valori delle varibiali decisionali sono:

$$x_{01} = 1 \quad x_{12} = 1 \quad x_{83} = 1 \quad x_{312} = 1$$

Esaminiamo ora i due casi possibili:

Caso 1: $y_8 = 0$
Dal momento che $y_8 = 0$, per il set di vincoli **(6)**, il valore di tutte le varibili decisionali associate agli archi uscenti dal nodo con indice 8 devono essere minori o uguali di y_8, in particolare:

$$x_{83} \leq 0$$

E ciò è in contraddizione con il valore di partenza della soluzione ammissibile

Caso 2: $y_8 \neq 0$
Se $y_8 \neq 0$, diversamente dal Caso 1, il set di vincoli **(6)** è soddisfatto ma, per il set di vincoli **(5)**, il valore di y_8 può essere maggiore di 0 solo se esiste almeno un arco entrante nel nodo di indice 8, e ciò non accade per questa soluzione.
Se aggiungessimo un arco entrante in 8 il problema si trasferirebbe al nodo che abbiamo adesso aggiunto.

L'unica soluzione ammissibile è una soluzione che ha come nodo di partenza il nodo del turista (contrassegnato con 0) perchè è l'unico nodo che non soggetto al set di vincoli **(5)** e che, quindi, può avere un arco uscente senza avere un arco entrante.

4 Implementazione

4.1 Linguaggio e librerie

Per l'implementazione del PLI abbiamo utilizzato *python* come linguaggio di programmazione, in quanto ricco di librerie necessarie per il raggiungimento del nostro scopo. In particolare la libreria da noi utilizzata è *OR-tools*, messa a disposizione da Google gratuitamente. Tale libreria utilizza il Solver di programmazione intera mista *Coin-or branch and cut (CBC)*, solver open source messo a disposizione dalla ***Co****mputational* ***In****frastructure for* ***O****perations* ***R****esearch*.

4.2 Inizializzazione

Abbiamo svilupato il programma in modo che sia possibile analizzare i risultati delle soluzioni offerte dal Solver al variare delle seguenti condizioni iniziali:

- Numero di threads concessi al Solver per la risoluzione
- Tempo massimo concesso al Solver per la risoluzione
- Numero di monumenti visitabili dall'utente
- Tempo a disposizione dell'utente
- Politica di selezione dei monumenti (nel caso in cui il Solver dovesse impiegare più del tempo concesso, l'insieme dei monumenti iniziale potrà essere ristretto usando diverse politiche di selezione. Una spiegazione più dettagliata delle politiche di selezione è presente nel paragrafo delle Sperimantazioni)

4.3 Corpo principale e Output dei risultati

La funzione che si occupa della risoluzione del PLI, *BusyTraveler*, contiene le funzioni della libreria OR-tools per: inizializzare le variabili necessarie per la modellizzazione del PLI, introdurre i vincoli necessari, definire la funzione obiettivo, risolvere il PLI in base alle condizioni iniziali precedentemente descritte. Al termine di ogni esecuzione, vengono salvati su un file in formato *csv* (*BusyTravelerResults.csv*) i valori dell'upper e lower bound della soluzione assieme ai valori delle condizioni iniziali. La rappresentazione "visiva"della soluzione trovata, qualora il solver l'abbia trovata entro il tempo limite imposto, è invece salvata nel file *ResultsBusyTraveler.txt*.
Per ricostruire l'itinerario sottoforma di sequenza ordinata di nomi di monumenti abbiamo sfruttando la relazione d'ordine imposta dai vincoli di tipo **(7)**. Difatti, data una soluzione, è bastato ordinare il vettore di variabili y_i e stampare il nome del monumento i-esimo seguendo l'ordine del vettore ordinato (di seguito un esempio dell'output del risultato "visivo").

Valore della funzione obbiettivo = 828.0
Percorso trovato con t = 10000:
Utente—->Scuderie del Quirinale—->Fontana del Babuino—->Mausoleo di Cecilia Metella—->Vascello (Il)—->Villa Dei Quintili—->Cortile della Pigna—->Galleria Sciarra—->Villa Lante.
Problema risolto in 0.0483 minuti

5 Sperimentazioni

5.1 Dataset utilizzato

Per testare l'efficacia della nostra formulazione abbiamo cercato, ove possibile, di utilizzare dati di monumenti realmente esistenti. La lista dei monumenti utilizzati, con le loro coordinate geografiche, è stata ricavata da una API gratuita che permetteva di richiedere informazioni sui monumneti romani. La distribuzione dei monumenti che abbiamo utilizzato, che in totale sono 100, è rappresentata in **Figura 1**.

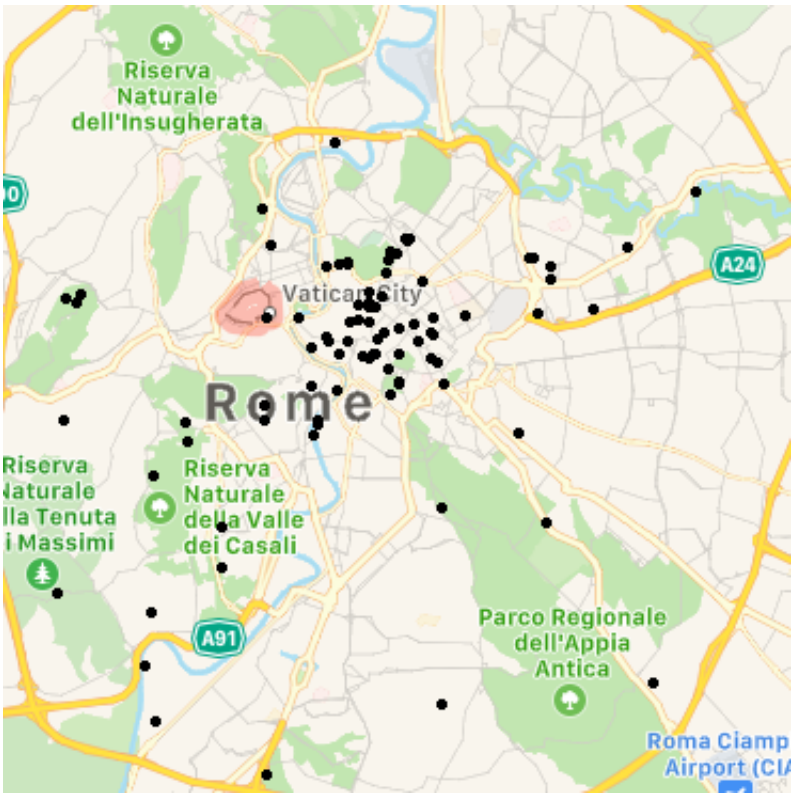


Figura 1: I monumenti utilizzati per il testing

Con la lista delle posizioni di tutti i monumenti abbiamo poi utilizzato le Google Maps API per ottenere tutte le distanze tra di essi, è qui che abbiamo dovuto limitare il numero dei monumenti, in quanto le Google API sono a pagamento e il grant che noi possediamo ci ha permesso di fare le richieste necessarie per trovare le distanze di soli 100 monumenti.

Con queste informazioni ci è stato possibile costruire il grafo dei monumenti e le ultime informazioni da ricavare erano quelle della popolarità dei monumenti e quelle sulla distanza dell'utente da essi. Per quanto riguarda la popolarità non siamo riusciti a trovare un modo affidabile per ottenerla automaticamente con le informazioni su internet, quindi abbiamo deciso di assegnare ad ogni monumento un valore casuale da 1 a 100. La distanza dell'utente, invece, è un dato che dovrebbe cambiare ad ogni risoluzione del problema, tuttavia, per assicurare dei risultati coerenti in fase di testing abbiamo deciso di utilizzare delle distanze casuali fisse, assicurandoci comunque che siano consistenti.

5.2 Ricerca della migliore approssimazione

Abbiamo subito notato che il problema da noi formulato non era risolvibile utilizzando tutti e 100 i nodi, inoltre, il solver che abbiamo utilizzato non era in grado neanche di fornirci dei lower/upper bound nel tempo di computazione che gli avevamo concesso (1 ora).

Il nostro primo obiettivo, quindi, è stato quello di trovare il maggior numero di nodi per cui era posibile trovare dei bound, abbiamo fatto ciò diminuendo progressivamente il numero di nodi finchè non abbiamo ottenuto dei risultati. Abbiamo notato durante questa fase di testing, che le prestazioni del solver erano strettamente collegate allo stato interno del computer su cui veniva utilizzato, per questo motivo, abbiamo eseguito il programma più volte per ogni numero di nodi testato.

I migliori upper/lower bound ottenuti in questa fase sono in tabella 1.

N. nodi	Lower bound	Upper bound
51	1201.0	1673.99110

Tabela 1: I migliori bound ottenuti durante il testing

5.3 Euristiche

Non essendo riusciti a risolvere il problema per la totalità dei nodi, abbiamo cercato di utilizzare delle euristiche per selezionare quelli che potrebbero essere i nodi più interessanti. Le politiche che abbiamo utilizzato sono 3:

- Random, prendiamo i nodi casualmente
- Minima distanza, abbiamo preso i nodi che erano più vicini all'utente, senza guardare la popolarità
- Massima popolarità, abbiamo preso i nodi con maggiore popolarità, senza guardare la distanza
- Mix popolarità e distanza, abbiamo preso i nodi con la minima somma pesata tra distanza e popolarità (coefficente 1 alla distanza, coefficente -8 alla popolarità)

5.4 Analisi Risultati

Ognuna di queste politiche ci ha permesso di risolvere il problema per un numero di nodi diversi, i migliori risultati con ognuna di esse sono in taballa 2

Politica	N. nodi	F. obiettivo	Tempo impegato
Random	28	979	440s
Massima popolarità	23	1223	43s
Minima distanza	15	621	787s
Mista	30	1487	900s

Tabela 2: I migliori risultati di ogni politica

Dopo aver fatto questo abbiamo cercato di confrontare l'efficacia di ogni politica per lo stesso numero di nodi, i risutati sono in tabella 3.

15 nodi		
Politica	F. obiettivo	Tempo impegato
Random	641	17s
Massima popolarità	980	5s
Minima distanza	621	787s
Mista	797	138s

23 nodi		
Politica	F. obiettivo	Tempo impegato
Random	795	51s
Massima popolarità	1223	45s
Mista	1379	9s

28 nodi		
Politica	F. obiettivo	Tempo impegato
Random	979	550s
Mista	1435	223s

Tabela 3: I risultati limitati per numero di nodi

I risultati ottenuti in forma grafica sono riportati in figura 2 e 3.

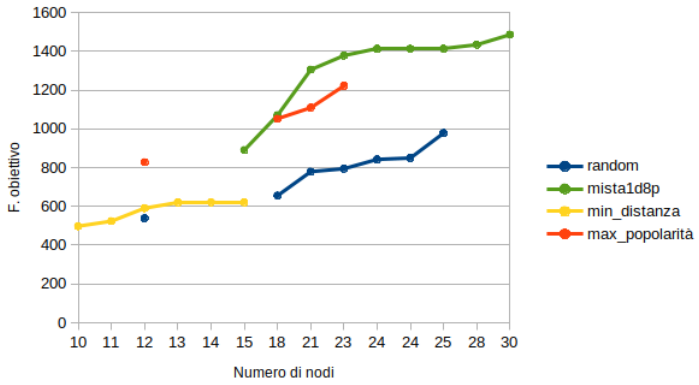


Figura 2: Funzione obiettivo e numero di nodi

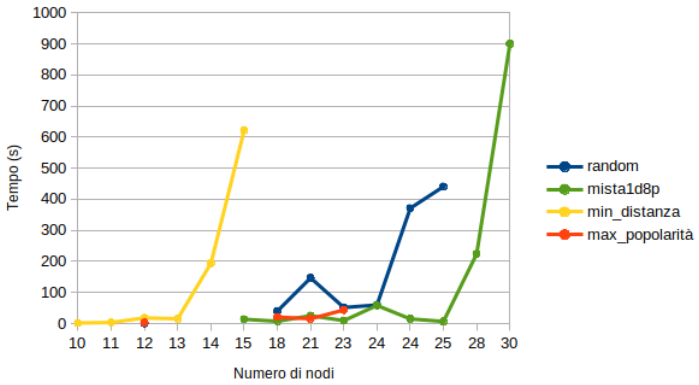


Figura 3: Tempo e numero di nodi

Un altro esperimento che abbiamo effettuato è stato quello di provare a dare ad ogni politica più tempo per risolvere il problema, in particolare abbiamo provato con 2, 3 e 4 ore. Abbiamo continuato ad aumentare il numero di nodi finchè non siamo arrivati ad un problema che non poteva essere risolto nel tempo di un'ora.

I risultati sono rappresentati in figura 4

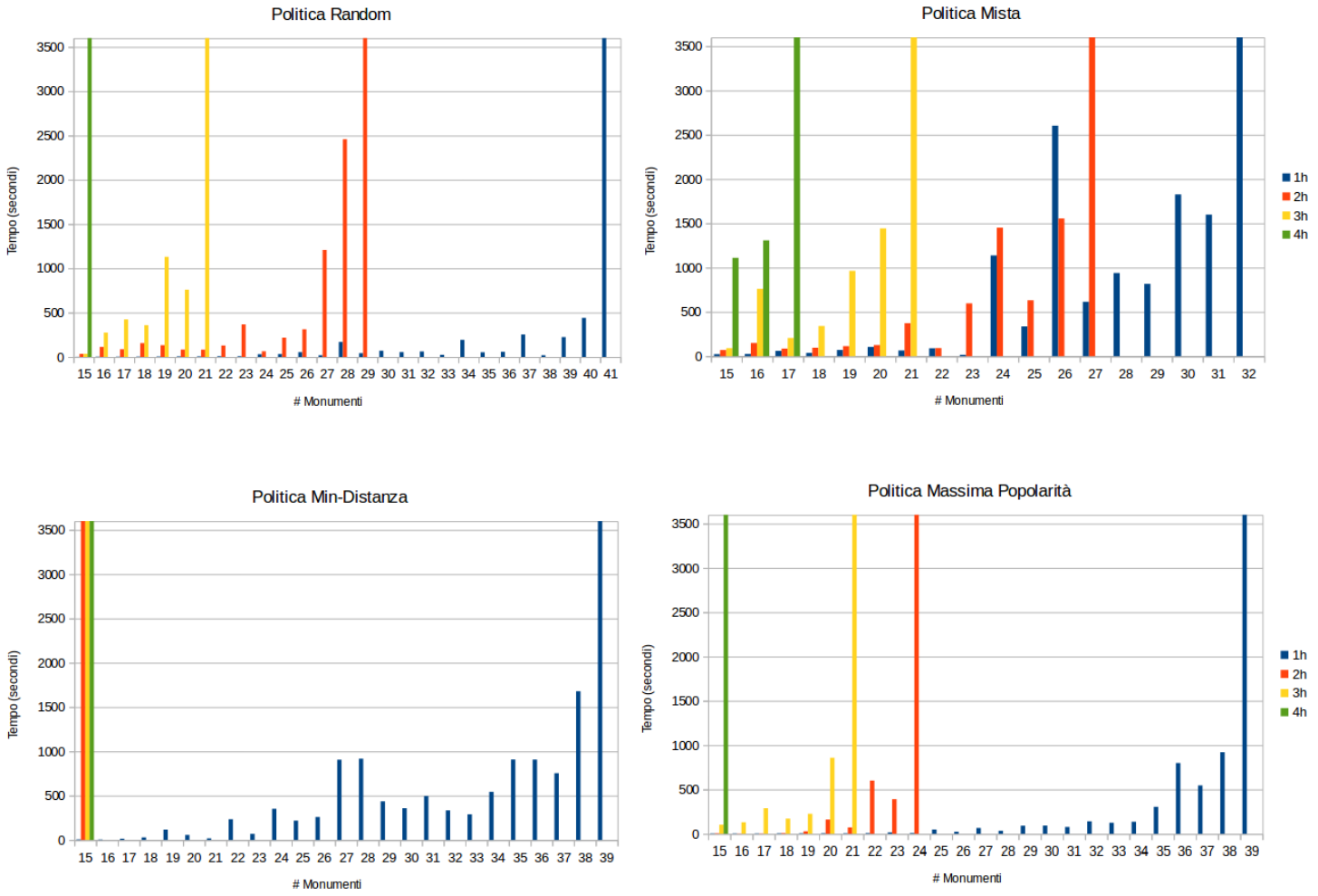
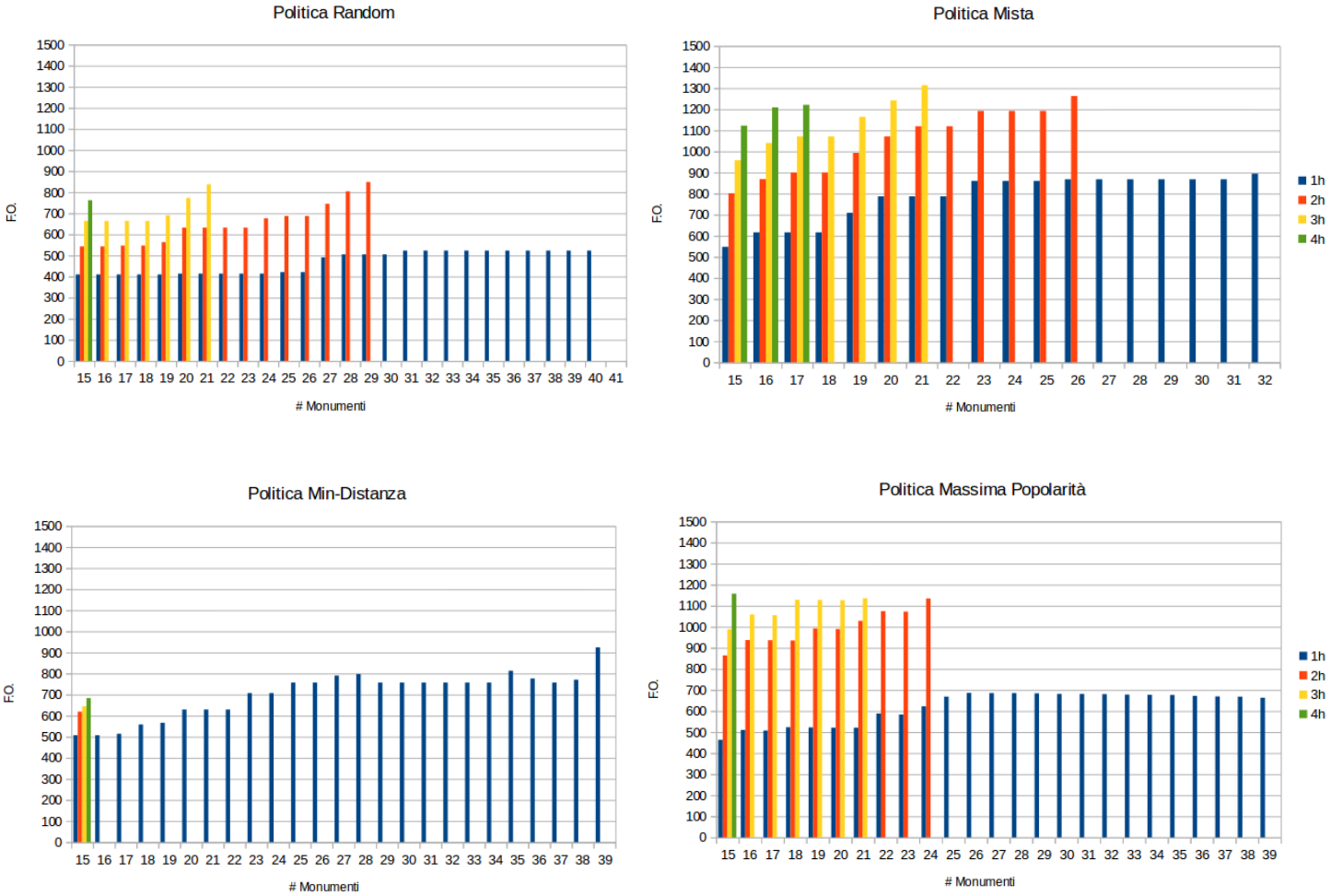


Figura 4: Tempo iimpegato aumentando le ore a disposizione

Come si può osservare dai grafici, all'aumentare del tempo messo a disposizione dall'utente il solver necessita di più tempo per la determinazione della soluzione ottima. Inoltre tanto maggiori sono i monumenti considerati e tanto deve essere minore il tempo messo a disposizione dall'utente per far sì che la soluzione venga fornita entro un ora.



Da questi grafici sono confermati i risultati che ci si aspettava di osservare: tanto maggiore è il tempo messo a disposizione dall'utente e tanto più 'gradimento' (in termini di valore della funzione obiettivo) che percepirà visitando i monumenti. Tuttavia come fatto notare precedentemente, se la soluzione ottima vuole essere calcolata entro un ora e si ha a disposizione molto tempo, il numero di monumenti deve essere ristretto, perdendo così la possibilità di aumentare sostanzialmente il valore della funzione obiettivo.

I valori graficati della funzione obiettivo, sono relativi al valore della soluzione ottima, oppure all'upper bound, nel caso il solver non è riuscito a trovarla nell'ora concessa.

6 Considerazioni finali

I dati raccolti da questo studio mostrano che il problema del Busy Traveller è molto complesso da un punto di vista computazionale anche con un numero relativamente basso di nodi.

Per quanto riguarda le euristiche considerate possiamo dire che le migliori sono quelle che filtrano i nodi in base alla popolarità e sono anche le uniche che ottengono un rislutato migliore del lower bound ottenuto dal Solver utilizzando la totalità dei nodi.
Analizzando più in dettaglio:

- politica Random, permette di risolvere il problema con un numero relativamente elevato di nodi tuttavia, dato che ignora la popolarità dei nodi, il valore della funzione obiettivo rimane comunque basso;
- politica di Massima Popolarità, i risultati di questa politica sono notevoli nonostante il numero inferiore di nodi che essa è in grado di gestire;
- politica di Minima distanza, questa è la politica che ha avuto le performance peggiori, sia per funzione obiettivo, cosa dovuta alla mancanza di filtri sulla popolarità, sia per numero

di nodi, cosa probabilmente dovuta alla vicinanza di tutti nodi e quindi all'impossibilità di eliminare velocemente nodi durante il calcolo della soluzione;

- politica Mista, questa è la politica migliore tra quelle studiate, è in grado di gestire il numero più elevato di nodi e fornisce il valore più alto della funzione obiettivo, questo è probabilemnte dovuto al fatto che abbiamo incluso solo i nodi popolari raggiungibili aumentando l'efficacia dell'algoritmo

Concludendo la politica Mista è quella che garantisce un buon compromesso tra soddisfazione garantita all'utente, intesa come valore della funzione obiettivo, e numero di monumenti considerati.

6.1 Limitazioni

Dal momento in cui le sperimentazioni sono *CPU-intensive* si è scelto di effettuarlee su delle istanze *EC2* offerte dal servizio di *Amazon AWS*, al fronte di risparmiare sui costi energetici (tutte le sperimentazioni sono state eseguite per un periodo ininterrotto di all'incirca 72 ore). Limitati dalla prova gratuita di tale servizio, sono state scelte istanze con le seguenti specifiche: Processore Intel Xeon a core singolo con frequenza di 3.30Ghz, 1 GB di RAM. Bisogna dunque affermare che i risultati ottenuti sono stati fortemente limitati da istanze di una così ridotta capacità computazionale.