Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

Analisi Sperimentale del Costo Computazionale di Problemi su Grafi Formulati in Programmazione Lineare Intera

Relatore:
Chiar.mo Prof.
Ugo Dal Lago

Presentata da:
Luca Orlandello

II Sessione
Anno Accademico 2022/2023

Indice

0 Introduzione **1**

1 I Modelli in Programmazione Lineare Considerati **3**
1.1 Cammini Minimi fra una Singola Coppia 3
1.1.1 Formulazione in Programmazione Lineare con Minimizzazione della Funzione Obbiettivo 4
1.1.2 Formulazione in Programmazione Lineare con Massimizzazione della Funzione Obbiettivo 6
1.2 Cammini Minimi a Sorgente Singola 8
1.2.1 Formulazione in Programmazione Lineare con Minimizzazione della Funzione Obbiettivo 9
1.2.2 Formulazione in Programmazione Lineare con Massimizzazione della Funzione Obbiettivo 10
1.3 Cammini Minimi fra Tutte le Coppie 11
1.3.1 Formulazione in Programmazione Lineare 11
1.4 Albero di Copertura Minimo 12
1.4.1 Formulazione in Programmazione Lineare con Flussi Multipli 13
1.4.2 Formulazione in Programmazione Lineare con Livello Assegnato ai Nodi 14

2 Ambiente di Benchmarking **17**
2.1 Requisiti per Benchmark Affidabili 17
2.1.1 Misurare le Risorse in Modo Accurato 18
2.1.2 Terminare i Processi in Modo Pulito 18
2.1.3 Assegnare i Core della CPU Deliberatamente 18
2.1.4 Disabilitare Swapping 20
2.1.5 Isolare Ogni Esecuzione 20

3 Valutazione Sperimentale del Costo Computazionale **21**
3.1 Generazione dei Grafi 21

3.1.1 Cammini Minimi: Singola Coppia, Singola Sorgente, Tutte le Coppie . . . 21
3.1.2 Albero di Copertura Minimo . . . 22
3.2 Cammini Minimi fra una Singola Coppia . . . 23
3.3 Cammini Minimi a Sorgente Singola . . . 30
3.4 Cammini Minimi fra Tutte le Coppie . . . 37
3.5 Albero di Copertura Minimo . . . 46

A Patch Glpsol, Calcolo Wall Time con clock_gettime e clock_monotonic_raw 51

Elenco delle tabelle

2.1 Requisiti per benchmark affidabili. 17

3.1 Riassunto dei modelli di programmazione lineare e algoritmi implementati per cammini minimi fra una singola coppia. 23
3.2 Riassunto dei modelli di programmazione lineare e algoritmi implementati per cammini minimi a singola sorgente. 30
3.3 Riassunto dei modelli di programmazione lineare e algoritmi per cammini minimi fra tutte le coppie sui quali sono stati effettuati i benchmark. 37
3.4 Coefficiente (c_1) e quota (c_2) della miglior retta che approssima le sezioni del Grafico 3.17 con n fissato. 41
3.5 Riassunto dei modelli di programmazione lineare per il problema dell'albero di copertura minimo sui quali sono stati effettuati i benchmark. 46
3.6 49

Elenco delle figure

3.1 Scatter plot dei tempi computazione misurati nell'esperimento preliminare. 24
3.2 Tempi di computazione di SP-LP-MAX e SP-LP-MIN nell'esperimento preliminare. 25
3.3 Tempi di computazione di BF e SP-LP-MAX nell'esperimento preliminare. 25
3.4 Tempi di computazione di BF e SP-LP-MIN nell'esperimento preliminare. 26
3.5 Tempi di computazione di BF (moltiplicati per 4.5) e SP-LP-MIN (non modificati) nell'esperimento preliminare. 26
3.6 Tempi di computazione di SP-LP-MIN nel secondo esperimento. . . . 28
3.7 Funzione che approssima i tempi di computazione di SP-LP-MIN. . . 28
3.8 Sezioni del grafico della funzione che approssima i tempi di computazione di SP-LP-MIN fissando la variabile n. 29
3.9 Scatter plot dei tempi computazione misurati nell'esperimento preliminare. 31
3.10 Tempi di computazione di SS-LP-MAX e SS-LP-MIN nell'esperimento preliminare. 32
3.11 Tempi di computazione di BF e SS-LP-MAX nell'esperimento preliminare. 32
3.12 Tempi di computazione di BF e SS-LP-MIN nell'esperimento preliminare. 33
3.13 Tempi di computazione di BF (moltiplicati per 3.5) e SS-LP-MIN (non modificati) nell'esperimento preliminare. 33
3.14 Tempi di computazione di SP-LP-MIN nel secondo esperimento. . . . 35
3.15 Funzione che approssima i tempi di computazione di SP-LP-MIN. . . 35
3.16 Sezioni del grafico della funzione che approssima i tempi di computazione di SS-LP-MIN fissando la variabile n. 36
3.17 Tempi computazione di AP-LP. 38
3.18 Tempi computazione di FW. 39
3.19 Tempi computazione di JS. 39

3.20 Tempi di computazione di FW (moltiplicati per $c_1 = 33434.158$) e AP-LP (non modificati). 40
3.21 Tempi di computazione di JS (moltiplicati per $c_2 = 7270.9$) e AP-LP (non modificati). 40
3.22 Sezioni del grafico della funzione che approssima i tempi di computazione di AP-LP fissando la variabile n. 43
3.23 Coefficienti angolari al variare di n, delle rette nelle Figure 3.4. 44
3.24 Quota al variare di n, delle rette nelle Figure 3.4. 44
3.25 Funzione che approssima i tempi di computazione di AP-LP. 45
3.26 Scatter plot dei tempi di computazione di NL e MF nell'esperimento preliminare. 47
3.27 Grafico dei tempi di computazione di NL e MF nell'esperimento preliminare. 47
3.28 Scatter plot dei tempi di computazione di NL nel secondo esperimento. 48

Capitolo 0

Introduzione

La programmazione lineare è una tecnica di modellizzazione che utilizza funzioni lineari per descrivere problemi di ottimizzazione. Un problema di ottimizzazione consiste nel trovare assegnamento di valori a delle variabili in modo da rispettare alcuni vincoli e tra tutti i possibili assegnamenti che rispettino i vincoli siamo interessati agli assegnamenti che minimizzano o massimizzano una qualche funzione chiamata funzione obbiettivo. Un problema di ottimizzazione può essere modellizzato in programmazione lineare se la funzione obbiettivo e i vincoli sono funzioni lineari nelle variabili del problema. Se le variabili del problema sono vincolate ad assumere soltanto numeri interi e non reali allora si parla di programmazione lineare intera. Individuato un modello in programmazione lineare o programmazione lineare intera per un problema, è possibile risolvere il problema fornendo i dati e il modello ad un risolutore automatico. La programmazione lineare è uno strumento molto potente e versatile perché permette in poche righe di codice di scrivere programmi per risolvere problemi complessi, però tutta questa versatilità si paga con il tempo di esecuzione. Gli algoritmi che implementano un risolutore automatico hanno tipicamente un tempo di esecuzione maggiore rispetto ad un algoritmo ad hoc per risolvere un particolare problema di ottimizzazione scritto in un qualsiasi linguaggio di programmazione e nel caso della programmazione lineare intera il tempo di esecuzione impiegato dal risolutore automatico per trovare la soluzione ottima può esplodere esponenzialmente. Inoltre dato un programma lineare non esiste una analisi tipo Big-O per classificarlo in una classe di complessità di tempo e quindi il principale modo per valutare il costo computazionale di un programma lineare, oltre al misurare il numero di variabili e vincoli necessari, rimane quello di valutarlo sperimentalmente. Il lavoro di questa tesi consiste nel valutare sperimentalmente il costo computazionale impiegato per risolvere alcuni problemi di ottimizzazione utilizzando modelli in programmazione lineare intera.

Sono stati considerati i seguenti problemi di ottimizzazione su grafi, tutti noti ammettere algoritmi polinomiali deterministici:

1. cammini minimi fra una singola coppia
2. cammini minimi a sorgente singola
3. cammini minimi fra tutte le coppie
4. albero di copertura minimo

I modelli sono stati scritti con il linguaggio MathProg e risolti con il risolutore `glpsol` versione 5.0. Sia la documentazione e descrizione del linguggio MathProg sia i sorgenti di `glpsol` sono inclusi nel package GLPK (GNU Linear Programming Kit) versione 5.0 (Makhorin, 2020). Una introduzione all'ottimizzazione attraverso MathProg e `glpsol` si può trovare in (András Éles, 2019).

Nel Capitolo 1 vengono introdotti i modelli in programmazione lineare utilizzati per risolvere i problemi di cammini minimi e il problema dell'albero di copertura minimo. Nel Capitolo 2 viene descritta la macchina e la configurazione del sistema sul quale sono stati effettuati i benchmark per valutare sperimentalmente il costo computazionale. Infine nel Capitolo 3 sono riportati e analizzati i tempi di calcolo misurati durante la fase di benchmarking.

Capitolo 1

I Modelli in Programmazione Lineare Considerati

Nelle seguenti sezioni vengono presentati e discussi i modelli in programmazione lineare implementati per risolvere i problemi: cammini minimi fra una singola coppia, cammini minimi a sorgente singola, cammini minimi fra tutte le coppie e albero di copertura minimo.

1.1 Cammini Minimi fra una Singola Coppia

Introduciamo alcune definizioni utili per definire il problema dei cammini minimi fra una singola coppia, ma anche per definire il problema dei cammini minimi a sorgente singola e fra tutte le coppie.

Definizione 1.1. *Un grafo diretto $G = (N, A)$ è una coppia di insiemi tale che $A \subseteq N \times N$ e $N \cap A = \emptyset$. L'insieme N è chiamato insieme dei nodi (o vertici) del grafo mentre l'insieme A è chiamato insieme degli archi del grafo.*

Definizione 1.2. *Un grafo diretto e pesato $G = (N, A, c)$ è una tripla dove (N, A) è un grafo diretto e $c : A \to \mathbb{R}$ è una funzione che associa ad ogni arco del grafo un numero reale chiamato costo o peso dell'arco. Per alleggerire la notazione e usare meno parentesi utilizziamo c_{ij} o $c(i, j)$ al posto di $c((i, j))$ per indicare l'applicazione della funzione c all'arco (i, j)*

Definizione 1.3. *Sia $G = (N, A)$ un grafo diretto, un cammino in G di lunghezza k da un nodo s ad un nodo t è una sequenza di archi $\langle a_1, a_2, \ldots, a_k\rangle$ di lunghezza $k \in \mathbb{N}$ per cui esiste una sequenza di nodi $\langle n_1, n_2, \ldots, n_k, n_{k+1}\rangle$ di lunghezza $k + 1$ tale che $n_1 = s$ e $n_{k+1} = t$ e $\forall i \in \{1, 2, \ldots, k\}$ $a_i = (n_i, n_{i+1})$. Inoltre definiamo un*

cammino in G da un nodo s a un nodo t come un cammino in G di lunghezza k per qualche $k \in \mathbb{N}$

Definizione 1.4. *Sia $G = (N, A)$ un grafo diretto, un ciclo in G è un cammino da un nodo n allo stesso nodo n, con $n \in N$. Inoltre diciamo che un grafo diretto G è aciclico se non esistono cicli in G.*

Definizione 1.5. *Sia $G = (N, A)$ un grafo diretto, un cammino $\langle a_1, a_2, \ldots, a_k \rangle$ in G da un nodo a_1 ad un nodo a_2 si dice semplice se $\forall i, j \in \{1, \ldots, k\}$ $a_i \neq a_j$.*

Definizione 1.6. *Sia $G = (N, A)$ un grafo diretto, G è fortemente connesso se $\forall i, j \in N$ esiste un cammino in G dal nodo i al nodo j*

Definizione 1.7. *Sia $G = (N, A)$ un grafo diretto e pesato, sia $p = \langle (n_1, n_2), (n_2, n_3), \ldots, (n_{k-1}, n_k), (n_k, n_{k+1}) \rangle$ un cammino in G di lunghezza k da un nodo n_1 ad un nodo n_2. Definiamo il costo $c(p)$ del cammino p come la sommatoria di tutti i pesi degli archi che formano il cammino: $c(p) = \sum_{i=1}^{k} c(n_i, n_{i+1})$*

Definizione 1.8. *Sia $G = (N, A)$ un grafo diretto e pesato, sia p^* un cammino in G da un nodo s ad un nodo t. p^* è un cammino da s a t di costo minimo se per ogni cammino p da s a t il costo del cammino p^* è minore o uguale al costo del cammino p: $\forall p \;\; c(p^*) \leq c(p)$*

Dato un grafo diretto e pesato $G = (N, A, c)$ e una coppia di nodi del grafo $s, t \in N$, il problema del cammino minimo fra una singola coppia di nodi consiste nel trovare, se esiste, un cammino in G da s a t di costo minimo. Nelle successive sezioni vengono elencate le formulazioni in programmazione lineare implementate per risolvere il problema dei cammini minimi fra una singola coppia di nodi.

1.1.1 Formulazione in Programmazione Lineare con Minimizzazione della Funzione Obbiettivo

Dato un grafo diretto e pesato $G = (N, A, c)$ e una coppia di nodi $s, t \in N$, possiamo vedere il problema di trovare il cammino minimo fra una singola coppia di nodi come quello di mandare 1 unità di flusso dal nodo s al nodo t spendendo il meno possibile, dove il costo totale è il costo del cammino per il quale si fa passare la singola unità di flusso. La formulazione in programmazione lineare intera 1.1 sfrutta questa idea. Le variabili x_{ij} $\forall (i, j) \in A$ rappresentano il numero di unità di flusso che si fa passare sull'arco (i, j).

$$
\begin{array}{lll}
\text{variabili:} & & \\
& x_{ij} \in \mathbb{N} & \forall (i,j) \in A \\
\text{minimizza} & \sum_{(i,j)\in A} x_{ij} c_{ij} & \\
\text{vincoli:} & & \\
(1) & \sum_{(s,j)\in A} x_{sj} - \sum_{(i,s)\in A} x_{sj} = 1 & \\
(2) & \sum_{(t,j)\in A} x_{tj} - \sum_{(i,t)\in A} x_{it} = -1 & \\
(3) & \sum_{(k,j)\in A} x_{kj} - \sum_{(i,k)\in A} x_{ik} = 0 & \forall k \in N \setminus \{s,t\}
\end{array}
\tag{1.1}
$$

Di seguito è spiegato il significato di ogni vincolo utilizzato.
Il vincolo (1) dice che la sommatoria del flusso degli archi uscenti da s deve essere uguale a 1 + la sommatoria del flusso degli archi entranti in s, cioè che il nodo s deve essere un nodo che produce esattamente 1 unità di flusso.

Il vincolo (2) dice che la sommatoria del flusso degli archi entranti in t deve essere uguale a 1 + la sommatoria del flusso degli archi uscenti da t, cioè che il nodo t deve essere un nodo che trattiene esattamente 1 unità di flusso.

Il vincolo (3) dice che la sommatoria del flusso degli archi entranti in un nodo k diverso da s e t deve essere uguale alla sommatoria del flusso degli archi uscenti da k, cioè che il nodo k non deve produrre o trattenere unità di flusso ma deve solo inoltrare tutte e sole le unità di flusso degli archi entranti in k.

Infine alcune osservazione sul Modello 1.1. Il modello non ha soluzione se nel grafo G non esiste un cammino dal nodo s al nodo t. Se nel grafo G esiste almeno un cammino da s a t e esiste in G un ciclo di costo negativo allora la soluzione è illimitata. Se nel grafo G esiste almeno un cammino da s a t e non ci sono cicli di costo negativo allora esiste una soluzione ottima e il valore ottimo sarà uguale al costo del cammino di costo minimo da s a t. Inoltre non è sempre vero che il vettore che costituisce la soluzione ottima rappresenti un cammino semplice da s a t poiché possono esistere cicli di costo nullo nel grafo in input.

1.1.2 Formulazione in Programmazione Lineare con Massimizzazione della Funzione Obbiettivo

In questa sezione viene prima presentata una ulteriore formulazione in programmazione lineare per il problema di trovare il cammino minimo fra un singola coppia di nodi, dato un grafo diretto e pesato $G = (N, A, c)$ e una coppia di nodi del grafo $s, t \in N$, e successivamente si discute la correttezza.

$$\begin{aligned}
&\text{variabili:} && \\
&& d_k \in \mathbb{R} \qquad & \forall k \in N \\
&\text{massimizza} & d_t & \\
&\text{vincoli:} && \\
&& d_s = 0 & \\
&& d_j \leq d_i + c_{ij} \qquad & \forall (i,j) \in A
\end{aligned} \tag{1.2}$$

I successivi lemmi servono per dimostrare la correttezza di questa formulazione per il problema dei cammini minimi fra una coppia di nodi.

Lemma 1.1. *Sia G = (N, A, c) un grafo diretto e pesato. Se G ha almeno un ciclo di costo negativo allora la Formulazione 1.2 non ha nessuna soluzione ammissibile.*

Dimostrazione. Sia $C = \langle (1,2), (2,3), \ldots, (k-1,k), (k,1) \rangle$ un ciclo di costo negativo, cioè tale che $c_{12} + c_{23} + \cdots + c_{k-1k} + c_{k1} < 0$. Assumiamo che esista una soluzione ammissibile d e dimostriamo una contraddizione.

1. d è una soluzione ammissibile,
   quindi il vincolo $d_j \leq d_i + c_{ij}\ \ \forall (i,j) \in A$ è rispettato.

2. Gli archi che formano il ciclo C appartengono ad A:
   $(1,2), (2,3), \ldots, (k-1,k), (k,1) \in A$.

Da 1 e 2 otteniamo le seguenti disuguaglianze:

$$\begin{aligned}
d_1 &\leq d_k + c_{k1} \\
d_k &\leq d_{k-1} + c_{k-1k} \\
&\vdots \\
d_3 &\leq d_2 + c_{23} \\
d_2 &\leq d_1 + c_{12}
\end{aligned}$$

Unendo tutte le precedenti disuguaglianze otteniamo che:

$$d_1 \leq d_1 + c_{12} + c_{23} + \cdots + c_{k-1k} + c_{k1}$$

Poiché $c_{12} + c_{23} + \cdots + c_{k-1k} + c_{k1} < 0$ otteniamo anche che:

$$d_1 > d_1 + c_{12} + c_{23} + \cdots + c_{k-1k} + c_{k1}$$

Le due precedenti disuguaglianze formano una contraddizione.

□

Nei successivi lemmi e dimostrazioni si assume che ogni nodo del grafo sia sempre raggiungibile dalla sorgente s. I grafi utilizzati durante la fase benchmarking (vedi Sezione 3.1) sono tutti fortemente connessi, quindi questa assunzione è sempre vera.

Lemma 1.2. *Sia $G = (N, A, c)$ un grafo diretto, pesato e senza cicli di costo negativo. Sia $s \in N$ tale che per ogni $i \in N$ esiste almeno un cammino da s a i. Per ogni $i \in N$ sia p_i un cammino di costo minimo da s a i e sia d_i il costo di p_i. Le seguenti condizioni sono vere:*

$$\forall (i, j) \in A \;\; d_j \leq d_i + c_{ij} \quad e \quad d_s = 0$$

Dimostrazione. Sia $(i, j) \in A$, per dimostrare $d_j \leq d_i + c_{ij}$ procediamo per assurdo assumendo $d_j > d_i + c_{ij}$.

1. Poiché $d_j > d_i + c_{ij}$ il cammino da s a j costruito aggiungendo l'arco (i, j) al cammino p_i ha costo minore del cammino p_j.

2. Da ipotesi p_j è il cammino di costo minimo da s a j.

Da 1 e 2 assurdo.

Dimostriamo ora $d_s = 0$.

1. $d_s \geq 0$: se d_s fosse minore di zero, esisterebbe un cammino da s a s, cioè un ciclo, di costo negativo.

2. Esiste un cammino da s a s di costo zero, costruiamo questo cammino partendo da s e restando in s senza percorrere nessun arco.

Poiché da ipotesi p_s è un cammino minimo da s a s e da 1 e 2 otteniamo che $d_s = 0$.

□

Dal Lemma 1.2 otteniamo che se il grafo in input è senza cicli di costo negativo e ogni nodo e raggiungibile dalla sorgente s, la Formulazione 1.2 ha sempre almeno una soluzione ammissibile. Inoltre la soluzione individuata dal Lemma 1.2 è proprio una soluzione che ci si aspetta di ricevere in output, però potrebbero esistere altre soluzioni ammissibili che rendono maggiore la funzione obbiettivo, il Lemma 1.3 dice che ciò non può mai accadere.

Lemma 1.3. *Sia $G = (N, A, c)$ un grafo diretto, pesato e senza cicli di costo negativo. Sia $s \in N$ tale che per ogni $i \in N$ esiste almeno un cammino da s a i. Siano $\forall i \in N \ \ d_i \in \mathbb{R}$. Per ogni $i \in N$ sia d_i^* il costo minimo tra il costo di tutti i cammini da s a i. Se $d_s = 0$ e $\forall(i,j) \in A \ \ d_j \leq d_i + c_{ij}$ allora $\forall i \in N \setminus \{s\} \ \ d_i \leq d_i^*$*

Dimostrazione. Sia $i \in N \setminus \{s\}$, chiamiamo $p_i = \langle(s,1),(1,2),\ldots,(k-2,k-1),(k-1,i)\rangle$ un cammino da s a i che ha costo d_i^*. Poiché da ipotesi $\forall(i,j) \in A \ \ d_j \leq d_i + c_{ij}$ e tutti gli archi del cammino p_i appartengono ad A, otteniamo se seguenti disuguaglianze:

$$\begin{aligned} d_i &\leq d_{k-1} + c_{k-1i} \\ d_{k-1} &\leq d_{k-2} + c_{k-2k-1} \\ &\vdots \\ d_2 &\leq d_1 + c_{12} \\ d_1 &\leq d_s + c_{s1} = c_{s1} \end{aligned}$$

L'ultima diseguaglianza $d_1 \leq c_{s1}$ deriva dal fatto che per ipotesi $d_s = 0$. Unendo tutte le precedenti disuguaglianze otteniamo che:

$$d_i \leq c_{s1} + c_{12} + \cdots + c_{k-2k-1} + c_{k-1i}$$

Poiché $c_{s1} + c_{12} + \cdots + c_{k-2k-1} + c_{k-1i}$ è il costo del cammino p_i otteniamo che:

$$d_i \leq d_i^*$$

□

1.2 Cammini Minimi a Sorgente Singola

Dato un grafo diretto e pesato $G = (N, A, c)$ e un nodo del grafo $s \in N$, il problema dei cammini minimi a sorgente singola consiste nel trovare, se esiste, $\forall k \in N \setminus \{s\}$ un cammino in G da s a k di costo minimo.
Nelle successive sezioni vengono elencate le formulazioni in programmazione lineare implementate per risolvere il problema dei cammini minimi sorgente singola.

1.2.1 Formulazione in Programmazione Lineare con Minimizzazione della Funzione Obbiettivo

Dato un grafo diretto e pesato $G = (N, A, c)$ e un nodo $s \in N$, possiamo vedere il problema di trovare i cammini minimi dalla sorgente s come quello di mandare 1 unità di flusso dal nodo s al nodo k $\forall k \in N \setminus \{s\}$ spendendo il meno possibile, dove il costo totale è la sommatoria per $k \in N$ del costo del cammino da s a k per il quale si fa passare una unità di flusso. La formulazione in programmazione lineare intera 1.3 sfrutta questa idea. Le variabili x_{ij} $\forall (i, j) \in A$ rappresentano il numero di unità di flusso che si fa passare sull'arco (i, j).

$$
\begin{array}{lll}
\text{variabili:} & & \\
& x_{ij} \in \mathbb{N} & \forall (i,j) \in A \\
\text{minimizza} & \sum_{(i,j)\in A} x_{ij} c_{ij} & \\
\text{vincoli:} & & \\
(1) & \sum_{(s,j)\in A} x_{sj} - \sum_{(i,s)\in A} x_{is} = |N| - 1 & \\
(2) & \sum_{(k,j)\in A} x_{kj} - \sum_{(i,k)\in A} x_{ik} = -1 & \forall k \in N \setminus \{s\}
\end{array}
\tag{1.3}
$$

Di seguito è spiegato il significato di ogni vincolo utilizzato.
Il vincolo (1) dice che la sommatoria del flusso degli archi uscenti da s deve essere uguale a $|N| - 1$ + la sommatoria del flusso degli archi entranti in s, cioè che il nodo s deve essere un nodo che produce esattamente $|N| - 1$ unità di flusso.

Il vincolo (2) dice che la sommatoria del flusso degli archi entranti in un nodo k diverso da s essere uguale a 1 + la sommatoria del flusso degli archi uscenti da k, cioè che il nodo k deve essere un nodo che trattiene esattamente 1 unità di flusso ed eventualmente inoltrare tutte e sole le restanti unita di flusso degli archi entranti in k.

Infine alcune osservazione sul Modello 1.3. Il modello non ha soluzione se nel grafo G esiste un nodo $k \in N$ tale che non esiste un cammino dal nodo s al nodo k. Se nel grafo G esiste almeno un cammino da s a k $\forall k \in N$ e esiste in G un ciclo di costo negativo allora la soluzione è illimitata. Se nel grafo G esiste almeno un cammino da s a k $\forall k \in N$ e non ci sono cicli di costo negativo allora esiste una soluzione ottima e il valore ottimo sarà uguale alla sommatoria per $k \in N \setminus \{s\}$ del costo del cammino minimo da s a k. Inoltre non è sempre vero che il vettore che

costituisce la soluzione ottima rappresenti un albero poiché possono esistere cicli di costo nullo nel grafo in input.

1.2.2 Formulazione in Programmazione Lineare con Massimizzazione della Funzione Obbiettivo

In questa sezione viene prima presentata una ulteriore formulazione in programmazione lineare per il problema di trovare i cammini minimi a sorgente singola e successivamente si discute la correttezza.

$$
\begin{aligned}
&\text{variabili:} \\
&\qquad d_k \in \mathbb{R} \qquad \forall k \in N \\
&\text{massimizza} \quad \textstyle\sum_{k \in N} d_k \\
&\text{vincoli:} \\
&\qquad d_s = 0 \\
&\qquad d_j \leq d_i + c_{ij} \qquad \forall (i,j) \in A
\end{aligned}
\tag{1.4}
$$

I successivi lemmi servono per dimostrare la correttezza di questa formulazione per il problema dei cammini minimi a sorgente singola. Il Lemma 1.1 continua a valere anche per la Formulazione 1.4 poiché ha gli stessi vincoli di 1.2. Quindi se il grafo contiene un ciclo di costo negativo la Formulazione 1.4 non ha nessuna soluzione ammissibile.

Nei successivi lemmi e dimostrazioni si assume che ogni nodo del grafo sia sempre raggiungibile dalla sorgente s. I grafi utilizzati durante la fase di benchmarking (vedi Sezione 3.1) sono tutti fortemente connessi, quindi questa assunzione è sempre vera.

Dal Lemma 1.2 otteniamo che se il grafo in input è senza cicli di costo negativo e ogni nodo e raggiungibile dalla sorgente s, la Formulazione 1.4 ha sempre almeno una soluzione ammissibile. Inoltre la soluzione individuata dal Lemma 1.2 è proprio la soluzione che ci si aspetta di ricevere in output, però potrebbero esistere altre soluzioni ammissibili che rendono maggiore la funzione obbiettivo, il Lemma 1.4 dice che ciò non può mai accadere.

Lemma 1.4. *Sia $G = (N, A, c)$ un grafo diretto, pesato e senza cicli di costo negativo. Sia $s \in N$ tale che per ogni $i \in N$ esiste almeno un cammino da s a i. Siano $\forall i \in N\ d_i \in \mathbb{R}$. Per ogni $i \in N$ sia d_i^* il costo minimo tra il costo di tutti i cammini da s a i. Se $d_s = 0$ e $\forall (i,j) \in A\ \ d_j \leq d_i + c_{ij}$ allora $\sum_{i \in N} d_i \leq \sum_{i \in N} d_i^*$*

Dimostrazione. Dal Lemma 1.3 otteniamo che $\forall i \in N\ \ d_i \leq d_i^*$. Da ciò segue che $\sum_{i \in N} d_i \leq \sum_{i \in N} d_i^*$ poiché possiamo ordinare i termini della prima e seconda som-

matoria e fare in modo che il j-esimo termine della prima sommatoria sia minore o uguale al j-esimo termine della secondo sommatoria. □

1.3 Cammini Minimi fra Tutte le Coppie

Dato un grafo diretto e pesato $G = (N, A, c)$, il problema dei cammini minimi fra tutte le coppie consiste nel trovare, se esiste, $\forall i \in N \ \ \forall j \in N$ un cammino in G da i a j di costo minimo. Nella sezione successiva viene presentata la formulazione in programmazione lineare implementata per risolvere il problema dei cammini minimi fra tutte le coppie.

1.3.1 Formulazione in Programmazione Lineare

Possiamo vedere il problema di trovare il cammino minimo fra tutte le coppie di nodi come quello di trovare i cammini minimi a sorgente singola a partire dalla sorgente $k \ \forall k \in N$. Di seguito è presentata una formulazione in programmazione lineare intera che sfrutta questa idea insieme alla formulazione in programmazione lineare 1.3 per risolvere il problema dei cammini minimi a sorgente singola.
Le variabili $x_{ijk} \ \ \forall (i, j) \in A \forall k \in N$ rappresentano il numero di unità di flusso che si fa passare sull'arco (i, j) considerando il nodo k come sorgente.

$$
\begin{array}{lll}
\text{variabili:} & & \\
& x_{ijk} \in \mathbb{N} & \forall (i,j) \in A \ \ \forall k \in N \\
\text{minimizza} & \sum_{k \in N} \sum_{(i,j) \in A} x_{ijk} c_{ij} & \\
\text{vincoli:} & & \\
& \sum_{(k,j) \in A} x_{kjk} - \sum_{(i,k) \in A} x_{ikk} = |N| - 1 & \forall k \in N \\
& \sum_{(h,j) \in A} x_{hjk} - \sum_{(i,h) \in A} x_{ihk} = -1 & \forall k \in N \ \ \forall h \in N \setminus \{k\}
\end{array}
\tag{1.5}
$$

Il Modello 1.5 non ha soluzione se il grafo G in input non è fortemente connesso e ha soluzione illimitata se il grafo ha un ciclo di costo negativo. Se il grafo è fortemente connesso e non ci sono cicli di costo negativi allora esiste una soluzione ottima e il valore ottimo sarà uguale alla sommatoria per $i \in N$ e $j \in N$ del costo del cammino minimo da i a j.

1.4 Albero di Copertura Minimo

Introduciamo alcune definizione utili per definire il problema dell'albero di copertura di costo minimo.

Definizione 1.9. *Un grafo non diretto* $G = (N, A)$ *è una coppia di insiemi tale che* $A \subseteq \{\{i, j\} : i, j \in N \wedge i \neq j\}$ *e* $N \cap A = \emptyset$*. L'insieme* N *è chiamato insieme dei nodi (o vertici) del grafo mentre l'insieme* A *è chiamato insieme degli archi del grafo.*

Definizione 1.10. *Un grafo non diretto e pesato* $G = (N, A, c)$ *è una tripla dove* (N, A) *è un grafo non diretto e* $c : A \to \mathbb{R}$ *è una funzione che associa ad ogni arco del grafo un numero reale chiamato costo o peso dell'arco. Per alleggerire la notazione e usare meno parentesi utilizziamo* c_{ij} *o* $c(i, j)$ *al posto di* $c(\{i, j\})$ *per indicare l'applicazione della funzione* c *all'arco* $\{i, j\}$

Definizione 1.11. *Sia* $G = (N, A)$ *un grafo non diretto, un cammino in* G *di lunghezza* k *da un nodo* s *ad un nodo* t *è una sequenza di archi* $\langle a_1, a_2, \ldots, a_k \rangle$ *di lunghezza* $k \in \mathbb{N}$ *per cui esiste una sequenza di nodi* $\langle n_1, n_2, \ldots, n_k, n_{k+1} \rangle$ *di lunghezza* $k + 1$ *tale che* $n_1 = s$ *e* $n_{k+1} = t$ *e* $\forall i \in \{1, 2, \ldots, k\}$ $a_i = \{n_i, n_{i+1}\}$*. Inoltre definiamo un cammino in* G *da un nodo* s *a un nodo* t *come un cammino in* G *di lunghezza* k *per qualche* $k \in \mathbb{N}$

Definizione 1.12. *Sia* $G = (N, A)$ *un grafo non diretto, un ciclo in* G *è un cammino da un nodo* n *allo stesso nodo* n*, con* $n \in N$*. Inoltre diciamo che un grafo non diretto* G *è aciclico se non esistono cicli in* G

Definizione 1.13. *Sia* $G = (N, A)$ *un grafo non diretto,* G *è connesso se* $\forall i, j \in N$ *esiste un cammino in* G *dal nodo* i *al nodo* j

Definizione 1.14. *Sia* $G = (N, A)$ *un grafo non diretto, un grafo* $H = (N', A')$ *è un sottografo di* G *se* $N' \subseteq N$ *e* $A' \subseteq A$*.*

Definizione 1.15. *Un albero è un grafo* $G = (N, A)$ *non diretto, aciclico e connesso.*

Lemma 1.5. *Sia* $T = (N, A)$ *un grafo non diretto. Le 3 seguenti proposizioni sono equivalenti:*

1. T *è un albero.*
2. T *è connesso e* $|A| = |N| - 1$
3. T *è aciclico e* $|A| = |N| - 1$

Definizione 1.16. *Sia* $G = (N, A)$ *un grafo non diretto, un sottografo* $T = (N', A')$ *di* G *è un albero di copertura se* T *è un albero e* $N' = N$*.*

Definizione 1.17. *Sia $T = (N', A')$ un albero di copertura di un grafo $G = (N, A, c)$, definiamo il costo $c(T)$ dell'albero di copertura T come la sommatoria del costo di tutti gli archi in A': $c(T) = \sum_{\{i,j\} \in A'} c_{ij}$*

Definizione 1.18. *Sia $G = (N, A, c)$ un grafo non diretto e pesato e sia T^* un albero di copertura di G. T^* è un albero di copertura di costo minimo se per ogni albero di copertura T di G il costo di T^* è minore del costo di T: $c(T^*) \leq c(T)$*

Dato un grafo $G = (N, A, c)$ non diretto e pesato, il problema dell'albero di copertura di costo minimo consiste nel trovare un albero di copertura di G di costo minimo.
Nelle sezioni successive vengono elencate le formulazioni in programmazione lineare implementate per risolvere il problema dell'albero di copertura di costo minimo.

1.4.1 Formulazione in Programmazione Lineare con Flussi Multipli

La seguente formulazione, tratta da (Abdelmaguid, 2018), si basa sull'idea che è possibile vedere un albero di copertura di un grafo non diretto $G = (N, A)$ come un insieme di cammini tale che ogni cammino parte dallo stesso nodo s scelto in modo arbitrario e per ogni nodo $k \in N \setminus \{s\}$ esiste uno e un solo cammino che parte da s e termina in k.
Per sfruttare questa idea dal grafo in input $G = (N, A, c)$ non diretto e pesato, costruiamo un grafo $D = (N', A', c)$ diretto e pesato nel seguente modo:

- D ha gli stessi nodi G, cioè $N = N'$.
- Per ogni arco $\{i, j\} \in A$ di costo c_{ij}, aggiungiamo gli archi diretti (i, j) e (j, i) entrambi di costo c_{ij} all'insieme degli archi A' di D.

La Formulazione 1.6 utilizza il grafo costruito D per calcolare l'albero di copertura di costo minimo sul grafo G.

Prima di presentare il modello è spiegato il significato delle variabili. Sia $s \in N'$ un nodo scelto in modo arbitrario, $\forall k \in N' \setminus \{s\}$ $\forall (i, j) \in A'$ la variabile f_{ij}^k rappresenta il numero di unità di flusso sull'arco (i, j) del grafo costruito D, nel flusso dal nodo s al nodo k. $\forall (i, j) \in A'$ la variabile d_{ij} ha valore 1 se esiste un $k \in N' \setminus \{s\}$ tale che la variabile f_{ij}^k contiene un numero di unità di flusso maggiore di zero, altrimenti d_{ij} ha valore 0. $\forall \{i, j\} \in A$ la variabile s_{ij} ha valore 1 se l'arco {i,j} fa parte dell'albero di copertura minimo, altrimenti ha valore 0.

variabili:

$$
\begin{array}{lll}
 & d_{ij} \in \{0,1\} & \forall (i,j) \in A' \\
 & f^k_{ij} \in \mathbb{R} & \forall k \in N' \ \forall (i,j) \in A' \\
 & s_{ij} \in \{0,1\} & \forall \{i,j\} \in A \\
\text{minimizza} & \sum_{\{i,j\} \in A} c_{ij} s_{ij} & \\
\text{vincoli:} & & \\
(1) & \sum_{(s,j) \in A'} f^k_{sj} - \sum_{(i,s) \in A'} f^k_{is} = 1 & \forall k \in N' \setminus \{s\} \\
(2) & \sum_{(i,k) \in A'} f^k_{ik} - \sum_{(k,j) \in A'} f^k_{kj} = 1 & \forall k \in N' \setminus \{s\} \\
(3) & \sum_{(i,v) \in A'} f^k_{iv} - \sum_{(v,j) \in A'} f^k_{vj} = 0 & \forall k \in N' \setminus \{s\} \ \forall v \in N' \setminus \{s,k\} \\
(4) & f^k_{ij} \geq 0 & \forall k \in N' \ \forall (i,j) \in A' \\
(5) & f^k_{ij} \leq d_{ij} & \forall k \in N' \setminus \{s\} \ \forall (i,j) \in A' \\
(6) & s_{ij} = d_{ij} + d_{ji} & \forall \{i,j\} \in A \\
(7) & \sum_{\{i,j\} \in A} s_{ij} = |N| - 1 &
\end{array}
\tag{1.6}
$$

I vincoli dal (1) al (6) del Modello 1.6 garantiscono che il sottografo individuato dalle variabili s_{ij} $\forall \{i,j\} \in A$ sia connesso e con $|N'|$ nodi. Mentre il vincolo (7) garantisce che il sottografo individuato dalle variabili s_{ij} $\forall \{i,j\} \in A$ ha esattamente $|N| - 1$. Per Lemma 1.5 i vincoli dal (1) al (7) garantiscono che il sottografo individuato dalle variabili s_{ij} $\forall \{i,j\} \in A$ sia un albero di copertura.

1.4.2 Formulazione in Programmazione Lineare con Livello Assegnato ai Nodi

La seguente formulazione, tratta da (Abdelmaguid, 2018), si basa sull'idea che è possibile vedere un albero di copertura di un grafo non diretto e pesato $G = (N, A)$ come un insieme di cammini tale che ogni cammino termina nello stesso nodo t scelto in modo arbitrario e per ogni nodo $k \in N \setminus \{t\}$ esiste uno e un solo cammino che parte da k e termina in t.

Per sfruttare questa idea dal grafo in input $G = (N, A, c)$ non diretto e pesato, costruiamo un grafo $D = (N', A', c)$ diretto e pesato nel seguente modo:

- D ha gli stessi nodi G, cioè $N = N'$.
- Per ogni arco $\{i, j\} \in A$ di costo c_{ij}, aggiungiamo gli archi diretti (i, j) e (j, i) entrambi di costo c_{ij} all'insieme A'.

La seguente formulazione utilizza il grafo costruito D per calcolare l'albero di copertura di costo minimo sul grafo G.

Sia $t \in N'$ un nodo scelto in modo arbitrario, $\forall (i, j) \in A'$ la variabile d_{ij} ha valore 1 se il corrispettivo arco non diretto $\{i, j\} \in A$ fa parte dell'albero di copertura minimo, altrimenti ha valore 0. Le variabili lvl_k $\forall k \in N'$ rappresentano il livello assegnato ad un nodo e sono utilizzate per eliminare i cicli.

$$
\begin{array}{llll}
\text{variabili:} & & & \\
 & d_{ij} \in \{0, 1\} & \forall (i, j) \in A' & \\
 & lvl_k \in \mathbb{R} & \forall k \in N' & \\
\text{minimizza} & \sum_{(i,j) \in A'} c_{ij} d_{ij} & & \\
\text{vincoli:} & & & \\
(1) & \sum_{(k,j) \in A'} d_{kj} = 1 & \forall k \in N' \setminus \{t\} & \\
(2) & \sum_{(t,j) \in A'} d_{tj} = 0 & & (1.7) \\
(3) & lvl_i \geq lvl_j + d_{ij} - |N'|(1 - d_{ij}) & \forall (i, j) \in A' & \\
(4) & lvl_t = 0 & & \\
(5) & 1 \leq lvl_k \leq |N'| - 1 & \forall k \in N' \setminus \{t\} & \\
(6) & d_{i,j} + d_{j,i} \leq 1 & \forall \{i, j\} \in A & \\
(7) & \sum_{(i,t) \in A'} d_{it} \geq 1 & &
\end{array}
$$

Per evitare di introdurre cicli in una soluzione ammissibile ad ogni nodo è assegnato un livello. Per il vincolo (3) se viene scelto un arco $(i, j) \in A'$, cioè $d_{ij} = 1$, il livello del nodo i deve essere maggiore o uguale del livello del nodo j di almeno 1, altrimenti se $d_{ij} = 0$ il vincolo è ridondante.

Se esistesse un ciclo $C = \langle(1,2),(2,3),\dots,(k-1,k),(k,1)\rangle$ in una soluzione ammissibile allora $d_{12} = d_{23} = \cdots = d_{k-1k} = d_{k1} = 1$ e quindi le seguenti disuguaglianze sarebbero verificate:

$$\begin{aligned} lvl_1 &\geq 1 + lvl_2 \\ lvl_2 &\geq 1 + lvl_3 \\ &\vdots \\ lvl_{k-1} &\geq 1 + lvl_k \\ lvl_k &\geq 1 + lvl_1 \end{aligned}$$

Unendo tutte le precedenti disuguaglianze otteniamo che:

$$lvl_1 \geq 1 + 1 + \cdots + 1 + 1 + lvl_1$$

La precedente disuguaglianza forma una contraddizione.

I vincolo (3) e (4) del Modello 1.7 garantiscono che il sottografo individuato dalla variabili d_{ij} sia aciclico e con $|N'|$ nodi, mentre i vincoli (1) e (2) garantiscono che il sottografo ha esattamente $|N| - 1$ archi. Per il Lemma 1.5 i vincoli dal (1) al (4) garantiscono che il sottografo individuato dalle variabili d_{ij} sia un albero di copertura.

Il vincolo (5) fornisce dei limiti superiori e inferiori alle variabili $lvl_k \;\; \forall k \in N' \setminus \{t\}$. Il vincolo (6) è un vincolo ridondante poiché solo con i vincolo dal (1) al (4) tutti i cicli erano già eliminati e quindi anche i cicli di lunghezza 2. Inoltre dai vincoli dal (1) al (4) segue che esiste almeno un arco entrante nel nodo t, per questo motivo anche il vincolo (7) è ridondante. I vincoli (5), (6) e (7) sono semanticamente ridondanti ma possono ridurre lo spazio di ricerca visitato dal risolutore per trovare la soluzione ottima.

Capitolo 2

Ambiente di Benchmarking

Tutti gli esperimenti e benchmark sono stati condotti su un computer portatile con processore Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz, 8 GB di ram, il sistema operativo Debian 12 (Bookworm) e la release 6.1.0-10-amd64 del kernel Linux.

I modelli in programmazione lineare sono stati risolti con il risolutore `glpsol` incluso nel package GLPK (GNU Linear Programming Kit) version 5.0 (Makhorin, 2020).

Tutti gli altri algoritmi implementati sono stati scritti in C e compilati con il compilatore `gcc` versione 12.2.0 usando l'opzione -02 come livello di ottimizzazione.

2.1 Requisiti per Benchmark Affidabili

In Tabella 2.1 sono riassunti i requisiti per effettuare dei benchmark affidabili descritti in Beyer u. a. (2019) e nelle seguenti sottosezioni è descritto come è stato configurato il sistema utilizzato per rispettare ognuno dei requisiti.

1. Misurare le risorse in modo accurato.
2. Terminare i processi in modo pulito.
3. Assegnare i core della cpu deliberatamente.
4. Disabilitare swapping.
5. Isolare ogni esecuzione.

Tabella 2.1: Requisiti per benchmark affidabili.

2.1.1 Misurare le Risorse in Modo Accurato

Per tutti i modelli risolti con `glpsol` e gli altri algoritmi implementati è stato misurato il wall time (differenza tra il tempo di inizio e il tempo di fine) soltanto dell'effettivo algoritmo non considerando eventuali operazioni di preprocessing (parsing di un file di input, costruzione delle strutture dati per rappresentare un grafo in memoria o la matrice dei vincoli di un modello di programmazione lineare) o postprocessing (stampa in output del wall time e valore della soluzione ottima).

Il wall time per essere accurato deve essere misurato con un clock che non può subire cambiamenti, alcuni clock possono subire piccoli aggiustamenti periodici che introdurrebbero errori nel calcolo del wall time. Il wall time deve essere misurato utilizzando un clock di cui si ha la garanzia che sia strettamente monotono e incrementato con un clock rate costante. Per questo motivo è stata utilizzata la funzione della libreria glibc `clock_gettime` utilizzando il clock `CLOCK_MONOTONIC_RAW`, vedi `$ man clock_gettime`. È stato modificato il codice sorgente di glpsol per utilizzare clock_gettime con clock_monotonic_raw, in Appendice A c'è la patch utilizzata.

Inoltre molte cpu moderne possono modificare dinamicamente la loro frequenza, possono aumentarla o diminuirla di diverse centinaia di megahertz a seconda del carico di lavoro della cpu. Ad esempio, se il carico di lavoro è basso la temperatura della cpu sarà ampiamente sotto il limite massimo consentito e quindi in automatico la cpu aumenta la propria frequenza. Viceversa se il carico di lavoro è alto la temperatura della cpu sarà alta e quindi la cpu riporta la propria frequenza a quella base. Nei processori Intel la funzionalità di modificare dinamicamente la frequenza viene chiamata Intel Turbo Boost. Per diminuire la variabilità nelle misurazioni del wall time il turbo boost è stato disabilitato, utilizzando il processore alla frequenza base di 2.50GHz. È possibile disabilitare il turbo boost con il comando `$ echo 1 | tee /sys/devices/system/cpu/intel_pstate/no_turbo` .

2.1.2 Terminare i Processi in Modo Pulito

Sia glpsol che tutti gli altri algoritmi implementati sono single thread e non creano processi figli. Per questi motivi non si può mai verificare il caso di un processo che termini senza prima terminare i suoi processi figli.

2.1.3 Assegnare i Core della CPU Deliberatamente

La macchina utilizzata ha 1 singola cpu con 2 core fisici e 2 core virtuali per ogni core fisico. Lo scheduler di un kernel normalmente assegna i thread dei processi a i core virtuali seguendo una o più politiche di scheduling. Due thread eseguiti su due core virtuali appartenenti allo stesso core fisico non sono indipendenti, ma condividono alcune risorse hardware che possono essere accedute soltanto in modo

mutualmente esclusivo. A causa di ciò il primo thread può influenzare non deterministicamente il wall time del secondo e viceversa.
La capacità di avere più core virtuali sullo stesso core fisico si chiama simultaneous multithreading (Hyperthreading è l'implementazione proprietaria di Intel) e può essere disabilitato facendo boot del kernel con l'opzione `nosmt`. Disabilitando il simultaneous multitheading la macchina avrà solo un core virtuale per ogni core fisico.
Se si usa grub come boot loader e Debian come distribuzione basta aggiungere `nosmt` alla variabile `GRUB_CMDLINE_LINUX` nel file `/etc/default/grub` e aggiornare la configurazione di grub eseguendo il comando `$ update-grub`. È possibile verificare che il simultaneous multithreading sia disattivato con il comando `$ cat /sys/devices/system/cpu/smt/active`, se è disattivato viene restituito 0 altrimenti 1.

A seconda della politica utilizzata, lo scheduler può sospendere e rimuovere un thread in esecuzione su un core virtuale per sostituirlo con un altro per poi riprendere l'esecuzione del primo in un secondo momento, sporcando il wall time del primo thread. Per questo motivo è necessario togliere alcuni core virtuali dalla gestione dello scheduler e assegnare deliberatamente i thread ai core virtuali della cpu. Per fare ciò è stato utilizzato `cset`: una applicazione python che permette di creare uno "scudo" attorno ad uno o più core virtuali e evitare che thread indesiderati vengano eseguiti sui core virtuali nello scudo. Cset sfrutta una feature del kernel Linux chiamata control groups (cgroups). Poiché è stato disabilitato il simultaneous multithreading la macchina ha a disposizione 2 core virtuali: il core virtuale 0 sul primo core fisico e il core virtuale 1 sul secondo core fisico, attorno al core virtuale 1 è stato creato lo "scudo" con cset con il comando: `$ cset shield --cpu=1 --kthread=on`, mentre il core virtuale 0 è rimasto gestito dallo scheduler. Con il comando `$ cset shield --exec -- <program>` è possibile eseguire un programma all'interno dello "scudo".

Su un sistema correttamente funzionante oltre ai programmi lanciati dall'utente sono presenti anche altri processi, lanciati in automatico durante la fase di boot e eseguiti in background, che implementato le varie funzionalità del sistema. Tutti i thread di questi ulteriori processi vengono gestiti dallo scheduler, poiché è stato attivato uno "scudo" sul core virtuale 1 lo scheduler li eseguirà tutti sul core virtuale 0. Il numero di questi ulteriori processi è stato minimizzato eseguendo tutti i benchmark con il sistema nello stato definito dal target systemd `rescue.target`. È possibile raggiungere questo stato con il comando `$ systemctl isolate rescue.target`. Il target rescue consiste nell'insieme minimo di processi necessari per avere un sistema funzionante.

2.1.4 Disabilitare Swapping

È possibile vedere tutte le zone di swap con il comando `$ cat /proc/swaps`. Sono state disabilitate tutte le zone di swap con il comando `swapoff -a -v`.

2.1.5 Isolare Ogni Esecuzione

Sia glpsol che tutti gli altri algoritmi implementati non sono stati isolati dal ricevere segnali come sigterm o sigkill da parte di altri processi poiché la macchina utilizzata per i benchmark è una macchina personale e non condivisa. Inoltre sia glpsol che tutti gli altri algoritmi implementati non utilizzano file con un nome hard coded, quindi non c'è il rischio che venga letto un file precedentemente scritto da una precedente esecuzione del programma su cui si sta eseguendo il benchmark.

Capitolo 3

Valutazione Sperimentale del Costo Computazionale

3.1 Generazione dei Grafi

In questa sezione è presentato come sono stati generati i grafi utilizzati per effettuare i benchmark sia per le 3 tipologie del problema dei cammini minimi: singola coppia, singola sorgente e tutte le coppie, sia per il problema dell'albero di copertura di costo minimo.

3.1.1 Cammini Minimi: Singola Coppia, Singola Sorgente, Tutte le Coppie

Per ogni tipologia del problema dei cammini minimi: singola coppia, singola sorgente e tutte le coppie, i grafi utilizzati per effettuare i benchmark sono stati generati con il seguente algoritmo:

1. Prendi in input i parametri n, m, l, u, p: n è il numero di nodi del grafo che si vuole ottenere in output, m è il numero di archi del grafo che si vuole ottenere in output, l, u, p: sono paramenti per la generazione dei pesi degli archi.

2. Genera un grafo G diretto scelto casualmente in modo uniforme tra tutti i grafi diretti di n nodi e m archi. Per ogni arco (i, j) il suo peso $w(i, j)$ è scelto casualmente in modo uniforme nell'intervallo $[l, u] \subset \mathbb{R}$.

3. Verifica che G sia fortemente connesso, se sì continua con 4. altrimenti torna a 2. Se viene superato il numero massimo di tentativi per generare un grafo fortemente connesso l'algoritmo termina.

4. Effettua un ripensamento degli archi di G, per ogni vertice v di G scegli casualmente in modo uniforme $h(v)$ nell'intervallo $[0, p] \subset \mathbb{R}$, poi associa ad ogni arco (i, j) il nuovo peso $\hat{w}(i, j) = w(i, j) - h(i) + h(j)$.
5. Ritorna G e $\hat{w}$.

Per ogni tipologia del problema dei cammini minimi: singola coppia, singola sorgente e tutte le coppie, i parametri l, u, p sono stati fissati nel seguente modo:

$$l = 0$$

$$u = 2 \times 10^6$$

$$p = 2 \times 10^6$$

Scegliendo l, u, p in questo modo, al passo 2. dell'algoritmo di generazione dei grafi possono essere generati soltanto grafi senza cicli di costo negativo. Inoltre la riscrittura dei pesi degli archi al passo 4. lascia invariato il costo di ogni ciclo del grafo su cui viene applicata ma può far diventare negativo il peso di alcuni archi del grafo.
Quindi per ogni tipologia del problema dei cammini minimi i grafi utilizzati per effettuare i benchmark hanno pesi anche negativi ma senza cicli di costo negativo.

3.1.2 Albero di Copertura Minimo

Per il problema dell'albero di copertura di costo minimo: i grafi utilizzati per effettuare i benchmark sono stati generati con il seguente algoritmo:

1. Prendi in input i parametri n, m, l, u: n è il numero di nodi del grafo che si vuole ottenere in output, m è il numero di archi del grafo che si vuole ottenere in output, l, u: sono paramenti per la generazione dei pesi degli archi.
2. Genera un grafo G non diretto scelto casualmente in modo uniforme tra tutti i grafi non diretti di n nodi e m archi. Per ogni arco (i, j) il suo peso $w(i, j)$ è scelto casualmente in modo uniforme nell'intervallo $[l, u] \subset \mathbb{R}$.
3. Verifica che G sia connesso, se sì continua con 4. altrimenti torna a 2. Se viene superato il numero massimo di tentativi per generare un grafo connesso l'algoritmo termina.
4. Ritorna G e w.

Per il problema dell'albero di copertura di costo minimo i parametri l, u sono stati fissati nel seguente modo:

$$l = 0$$

$$u = 2 \times 10^6$$

3.2 Cammini Minimi fra una Singola Coppia

In questa sezione sono stati effettuati alcuni benchmark su modelli in programmazione lineare, elencati in Tabella 3.1, per risolvere il problema dei cammini minimi fra una singola coppia di vertici in un grafo. Sono stati misurati i tempi di calcolo e analizzati i risultati ottenuti. Oltre ai modelli in programmazione lineare, sono stati eseguiti gli stessi benchmark sull'algoritmo Bellman Ford come base per effettuare un confronto con i tempi ottenuti dagli altri modelli.

Nome	**Breve descrizione**
BF	Bellman Ford, implementazione con doppio for. Vedi (Cormen u. a., 2009, p. 651).
SP-LP-MIN	Formulazione in programmazione lineare con minimizzazione della funzione obbiettivo. Vedi Modello 1.1.
SP-LP-MAX	Formulazione in programmazione lineare con massimizzazione della funzione obbiettivo. Vedi Modello 1.2.

Tabella 3.1: Riassunto dei modelli di programmazione lineare e algoritmi implementati per cammini minimi fra una singola coppia.

Inizialmente è stato effettuato un esperimento preliminare dove ogni algoritmo e modello in programmazione lineare in Tabella 3.1 è stato eseguito sullo stesso dataset di 60 grafi. Il dataset è stato costruito generando 3 grafi per ogni coppia (n, m) con $n \in \{200, 300, 400, 500\}$ e per ogni n, m è stato scelto prendendo 5 numeri naturali equispaziati nell'intervallo tra n e $n(n-1)$. Per ogni algoritmo e modello, i tempi di calcolo misurati su grafi con lo stesso numero di nodi e archi sono stati accorpati con una media aritmetica.

In Figura 3.1 è riportato lo scatter plot dei tempi in secondi ottenuti al variare di n e m. In Figura 3.2, 3.3 e 3.4 sono confrontati due a due i grafici dei tempi di ogni algoritmo. Si può notare che:

1. Il grafico SP-LP-MAX cresce molto più rapidamente di BF e SP-LP-MIN e per questo motivo è stato scartato da ulteriori esperimenti.

2. Il grafico di BF e SP-LP-MIN sono molto simili, provando per tentativi a moltiplicare tutti i tempi di BF per una costante si riesce ad ottenere due grafici quasi sovrapponibili. in Figura 3.5 si può osservare che moltiplicando per 4.5 tutti i tempi di BF otteniamo un grafico molto simile a quello di SP-LP-MIN.

Per queste 2 osservazioni è stato effettuato un secondo esperimento con grafi di dimensione maggiore per analizzare se l'andamento asintotico del costo computazionale di SP-LP-MIN sia lo stesso di BF ovvero $\Theta(nm)$.

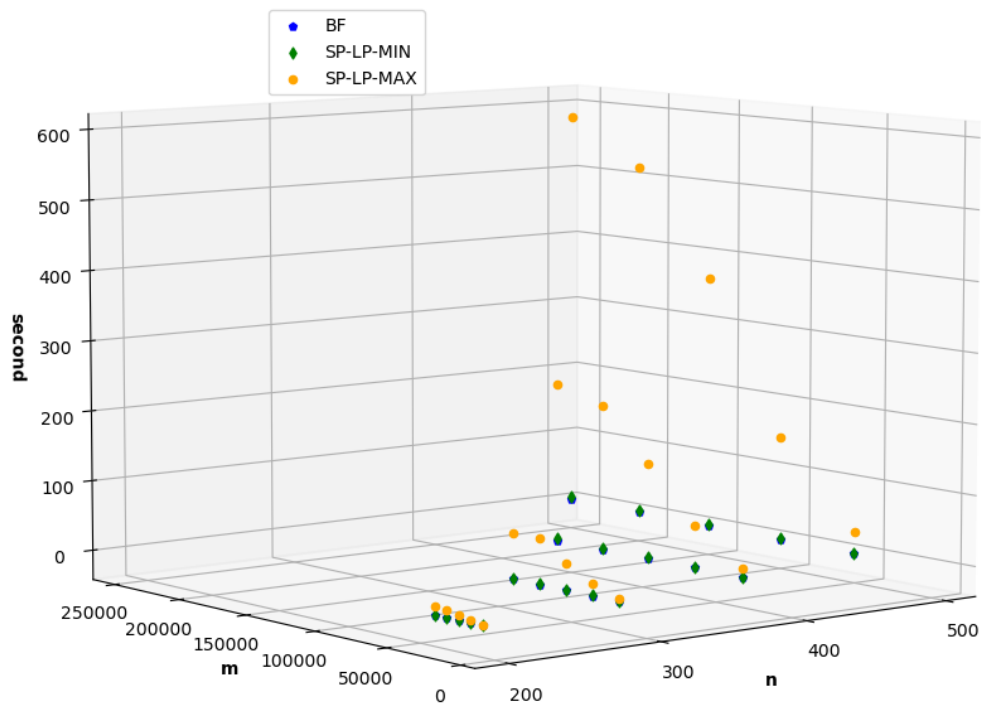


Figura 3.1: Scatter plot dei tempi computazione misurati nell'esperimento preliminare.

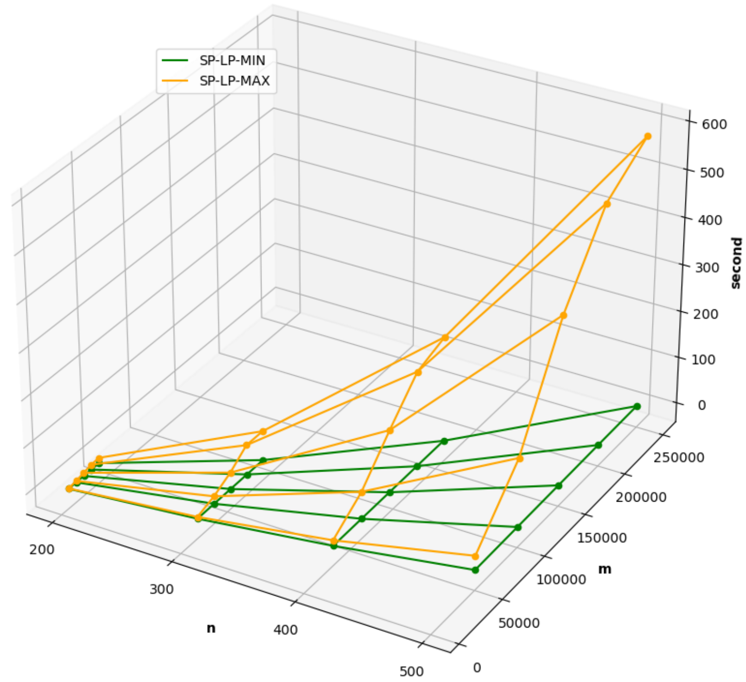


Figura 3.2: Tempi di computazione di SP-LP-MAX e SP-LP-MIN nell'esperimento preliminare.

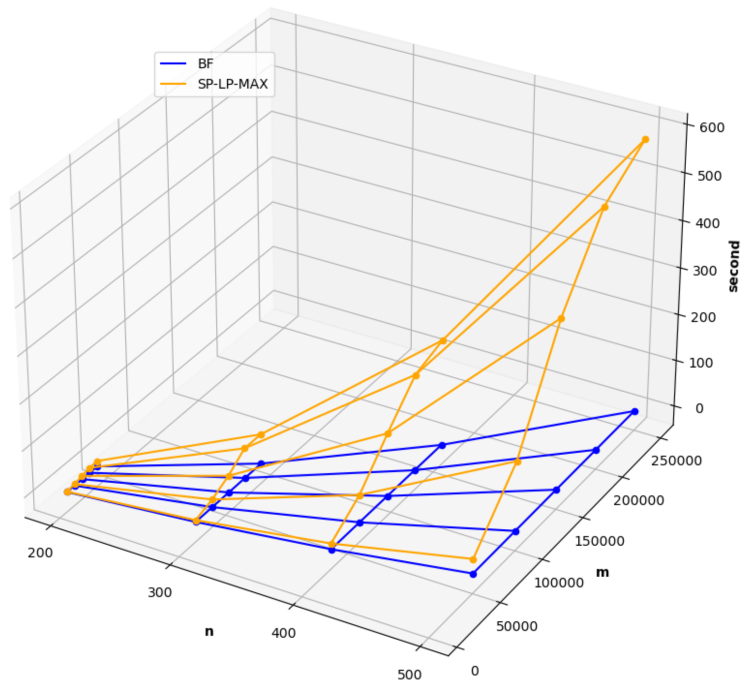


Figura 3.3: Tempi di computazione di BF e SP-LP-MAX nell'esperimento preliminare.

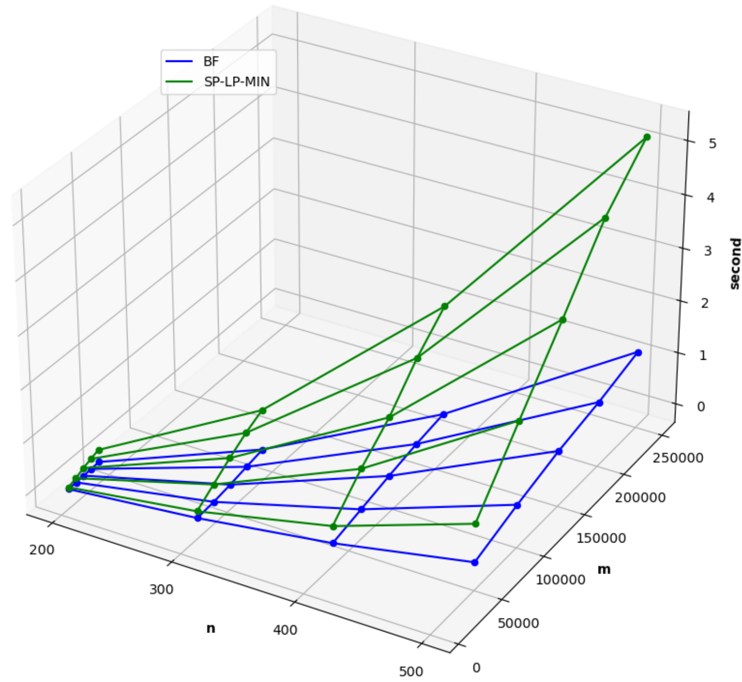


Figura 3.4: Tempi di computazione di BF e SP-LP-MIN nell'esperimento preliminare.

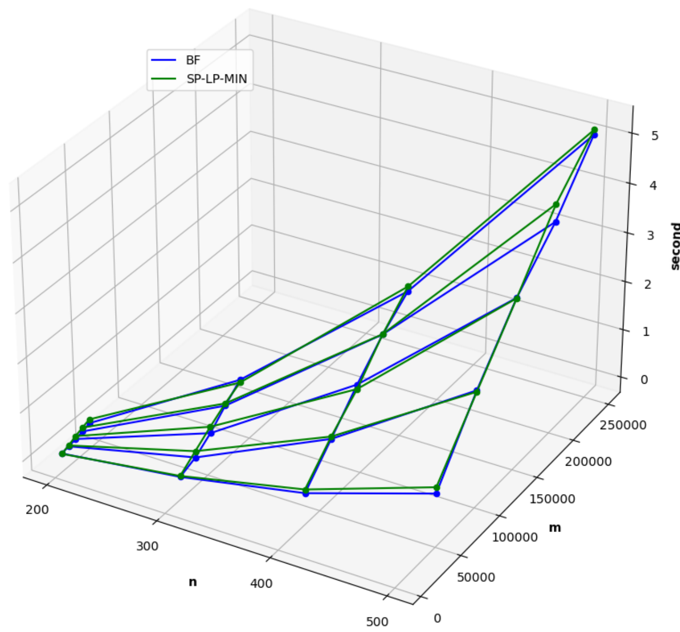


Figura 3.5: Tempi di computazione di BF (moltiplicati per 4.5) e SP-LP-MIN (non modificati) nell'esperimento preliminare.

Il secondo esperimento è stato effettuato eseguendo SP-LP-MIN su un dataset di 75 grafi, diverso dal dataset del precedente esperimento ma generato in modo simile. Il dataset è stato costruito generando 3 grafi per ogni coppia (n, m) con $n \in \{500, 1000, 1500, 2000, 2500\}$ e per ogni n, m è stato scelto prendendo 5 numeri naturali equispaziati nell'intervallo tra n e $n(n-1)$. I tempi misurati su grafi con lo stesso numero di nodi e archi sono stati accorpati con una media aritmetica. In Figura 3.6 è riportato il grafico dei tempi in secondi di SP-LP-MIN.

Per verificare l'ipotesi che il costo computazionale di SP-LP-MIN abbia un comportamento asintotico di $\Theta(nm)$, sui tempi di calcolo di SP-LP-MIN ottenuti nel secondo esperimento è stato fatto fitting di una funzione del tipo $f(n, m) = c_1nm + c_2n + c_3m + c_4$ con $c_1, c_2, c_3, c_4 \in \mathbb{R}$. La miglior funzione che approssima i dati è:

$$\begin{aligned} f(n, m) &= c_1nm + c_2n + c_3m + c_4 \\ c_1 &= +7.01293234 \times 10^{-8} \\ c_2 &= -2.19107384 \times 10^{-5} \\ c_3 &= -8.46227675 \times 10^{-3} \\ c_4 &= +6.54865159 \end{aligned} \tag{3.1}$$

In Figura 3.7 è riportato come la Funzione 3.1 approssima i tempi di calcolo di SP-LP-MIN. Nelle Figure 3.8a 3.8c 3.8e sono riportate delle sezioni del grafico in Figura 3.7 con n fissato: $n = 1500$, $n = 2000$ e $n = 2500$ rispettivamente. Graficamente la Funzione 3.1 approssima bene i dati.

Nelle Figure 3.8b 3.8d 3.8f è riportato il residuo della Funzione 3.1 con n fissato: $n = 1500$, $n = 2000$ e $n = 2500$ rispettivamente. Il residuo è calcolato come la differenza tra il valore del dato osservato e il valore della funzione che approssima i dati. I residui appaiono sparsi in modo casuale attorno allo zero, indicando che la Funzione 3.1 descrive bene i dati.

Dai grafici evince che la Funzione 3.1 approssima bene i tempi di calcolo di SP-LP-MIN e quindi si può concludere che il costo computazionale asintotico di SP-LP-MIN sia $\Theta(nm)$.

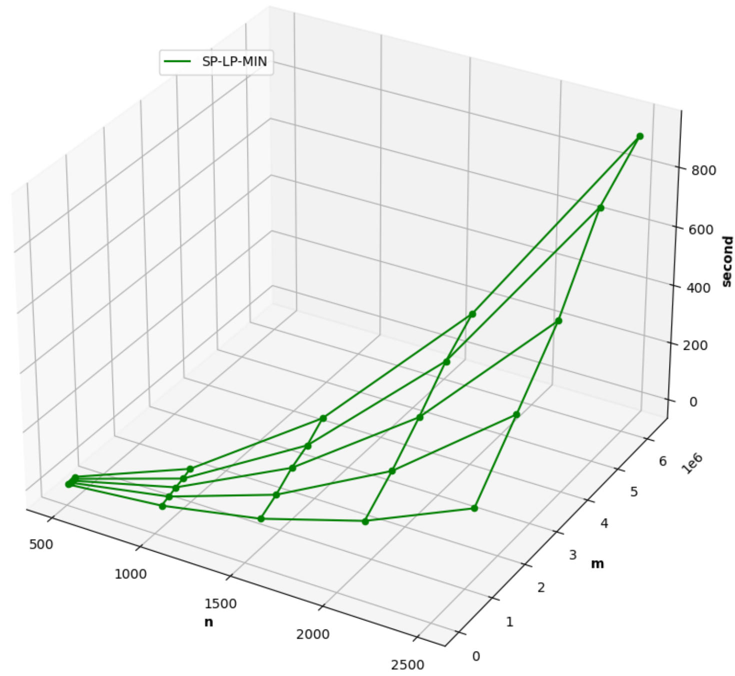


Figura 3.6: Tempi di computazione di SP-LP-MIN nel secondo esperimento.

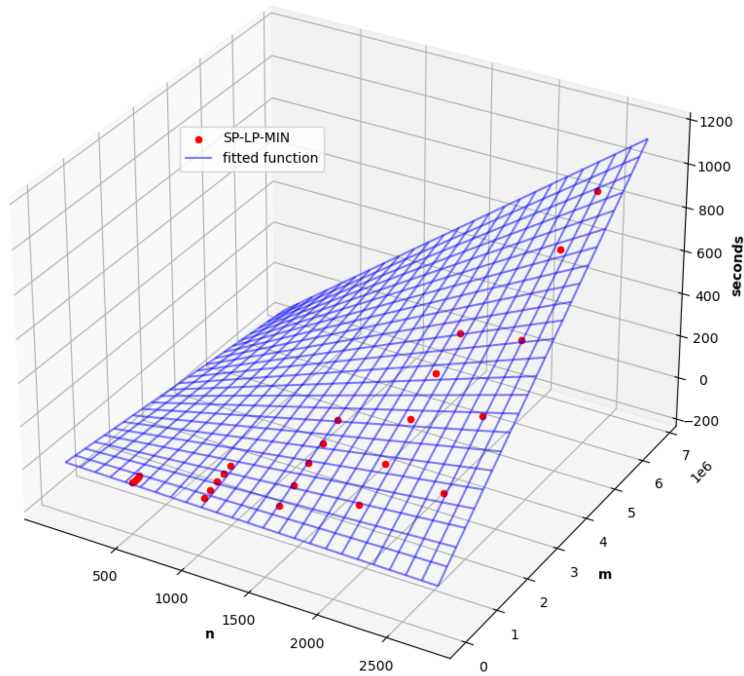
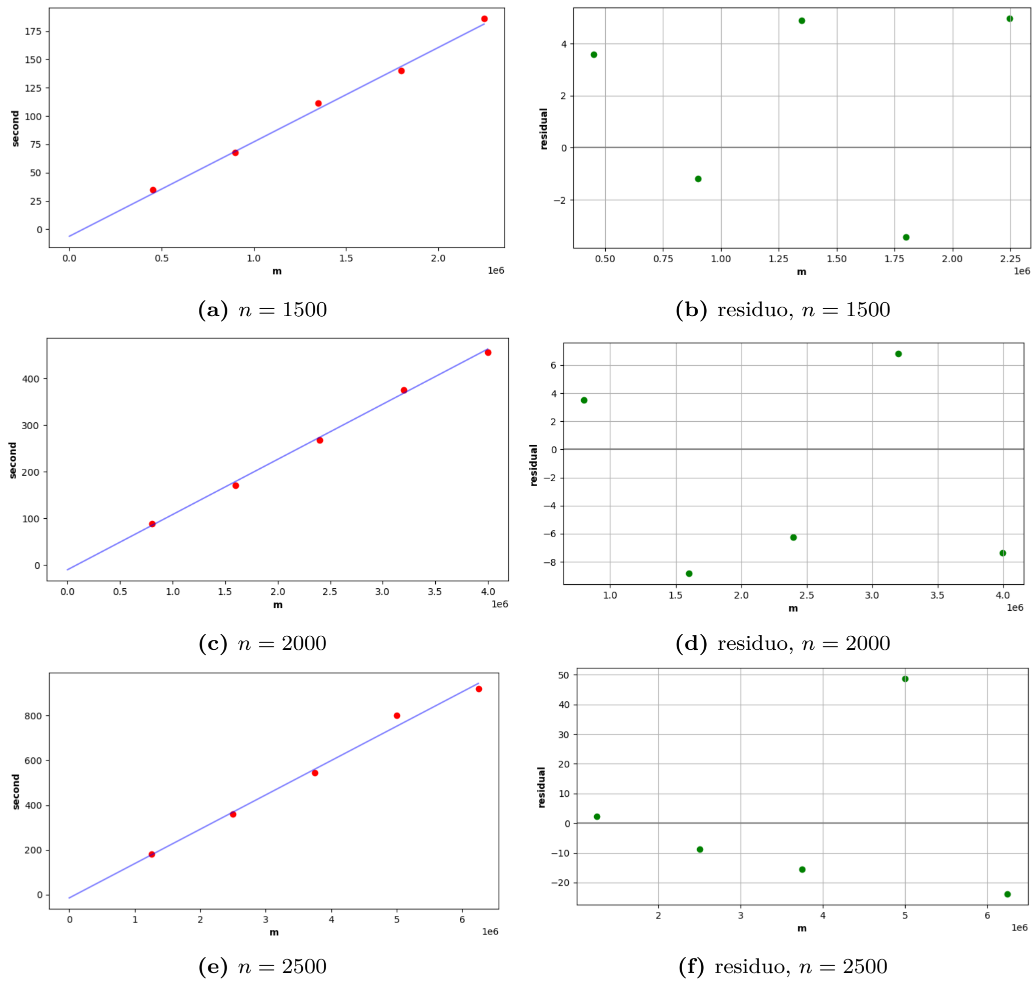


Figura 3.7: Funzione che approssima i tempi di computazione di SP-LP-MIN.



(a) $n = 1500$ **(b)** residuo, $n = 1500$

(c) $n = 2000$ **(d)** residuo, $n = 2000$

(e) $n = 2500$ **(f)** residuo, $n = 2500$

Figura 3.8: Sezioni del grafico della funzione che approssima i tempi di computazione di SP-LP-MIN fissando la variabile n.

3.3 Cammini Minimi a Sorgente Singola

In questa sezione sono stati effettuati alcuni benchmark su modelli in programmazione lineare, elencati in Tabella 3.2, per risolvere il problema dei cammini minimi a sorgente singola in un grafo. Sono stati misurati i tempi di calcolo e analizzati i risultati ottenuti. Oltre ai modelli in programmazione lineare, sono stati eseguiti gli stessi benchmark sull'algoritmo Bellman Ford come base per effettuare un confronto con i tempi ottenuti negli altri modelli. Le modalità di analisi utilizzate in questa sezione sono molto simili a quelle utilizzate nella Sezione 3.2.

Nome	**Breve descrizione**
BF	Bellman Ford, implementazione con doppio for. Vedi (Cormen u. a., 2009, p. 651).
SS-LP-MIN	Formulazione in programmazione lineare con minimizzazione della funzione obbiettivo. Vedi Modello 1.3.
SS-LP-MAX	Formulazione in programmazione lineare con massimizzazione della funzione obbiettivo. Vedi Modello 1.4.

Tabella 3.2: Riassunto dei modelli di programmazione lineare e algoritmi implementati per cammini minimi a singola sorgente.

Inizialmente è stato effettuato un esperimento preliminare dove ogni algoritmo e modello in programmazione lineare in Tabella 3.2 è stato eseguito sullo stesso dataset di 60 grafi. Il dataset utilizzato è lo stesso utilizzato per l'esperimento preliminare effettuato nella Sezione 3.2. Per ogni algoritmo e modello, i tempi di calcolo misurati su grafi con lo stesso numero di nodi e archi sono stati accorpati con una media aritmetica.

In Figura 3.9 è riportato lo scatter plot dei tempi in secondi ottenuti al variare di n e m. In Figura 3.10, 3.11 e 3.12 sono confrontati due a due i grafici dei tempi di ogni algoritmo. Si può notare che:

1. Il grafico SS-LP-MAX cresce molto più rapidamente di BF e SS-LP-MIN e per questo motivo è stato scartato da ulteriori esperimenti.
2. Il grafico di BF e SS-LP-MIN sono molto simili, provando per tentativi a moltiplicare tutti i tempi di BF per una costante si riesce ad ottenere due grafici quasi sovrapponibili. in Figura 3.13 si può osservare che moltiplicando per 3.5 tutti i tempi di BF otteniamo un grafico molto simile a quello di SP-LP-MIN.

Per queste 2 osservazioni è stato effettuato un secondo esperimento con grafi di dimensione maggiore per analizzare se l'andamento asintotico del costo computazionale di SS-LP-MIN sia lo stesso di BF ovvero $\Theta(nm)$.

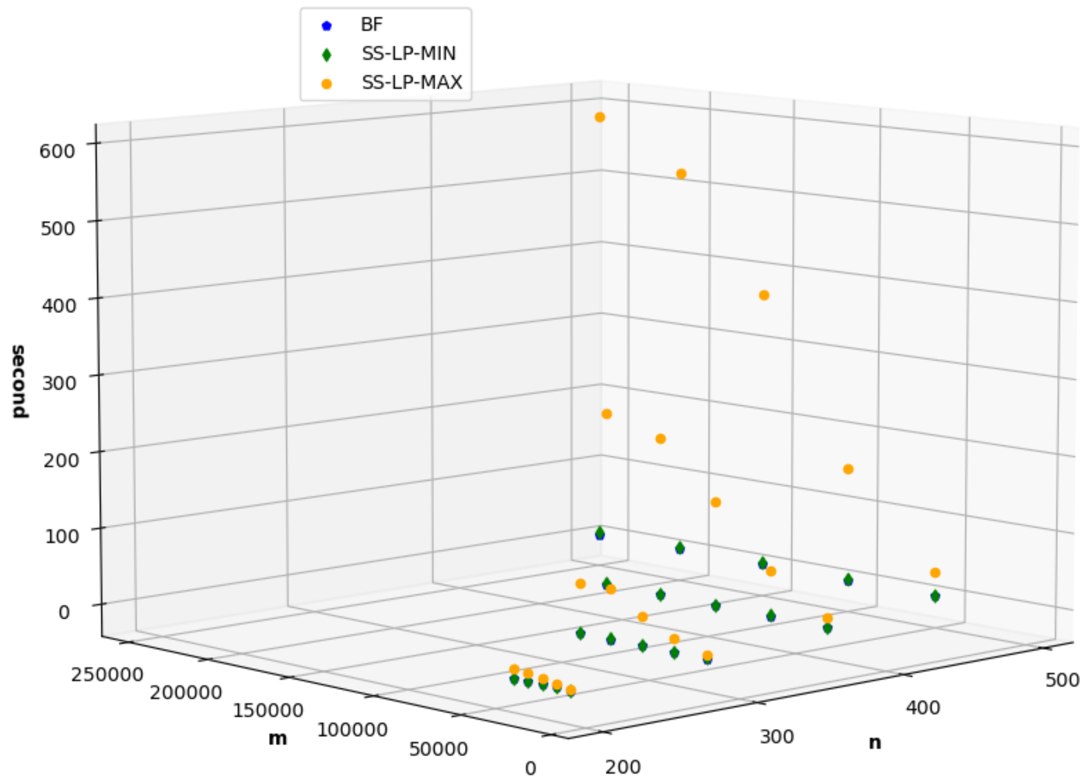


Figura 3.9: Scatter plot dei tempi computazione misurati nell'esperimento preliminare.

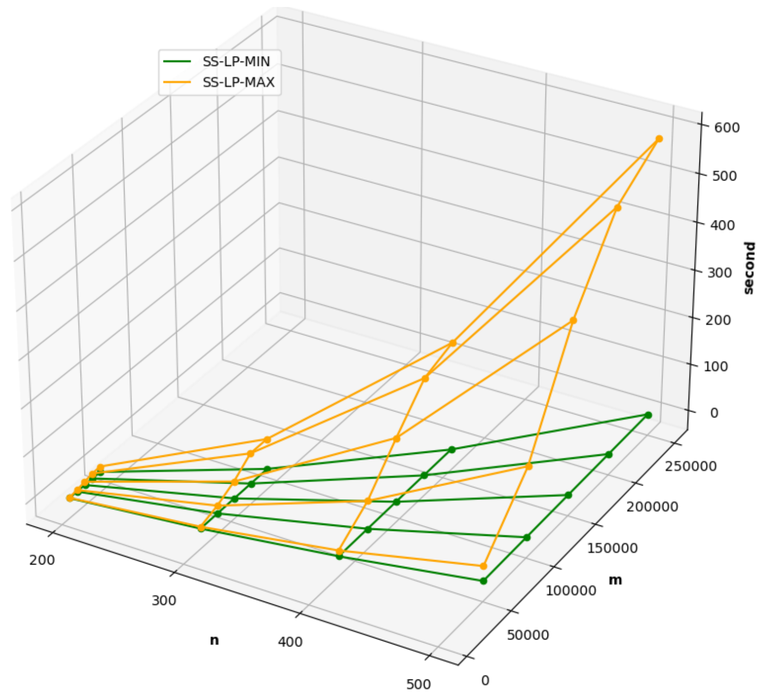


Figura 3.10: Tempi di computazione di SS-LP-MAX e SS-LP-MIN nell'esperimento preliminare.

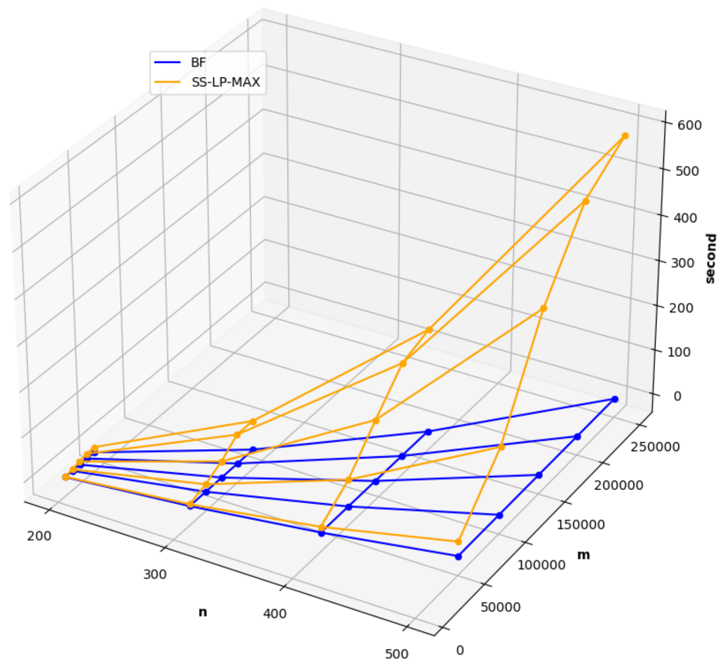


Figura 3.11: Tempi di computazione di BF e SS-LP-MAX nell'esperimento preliminare.

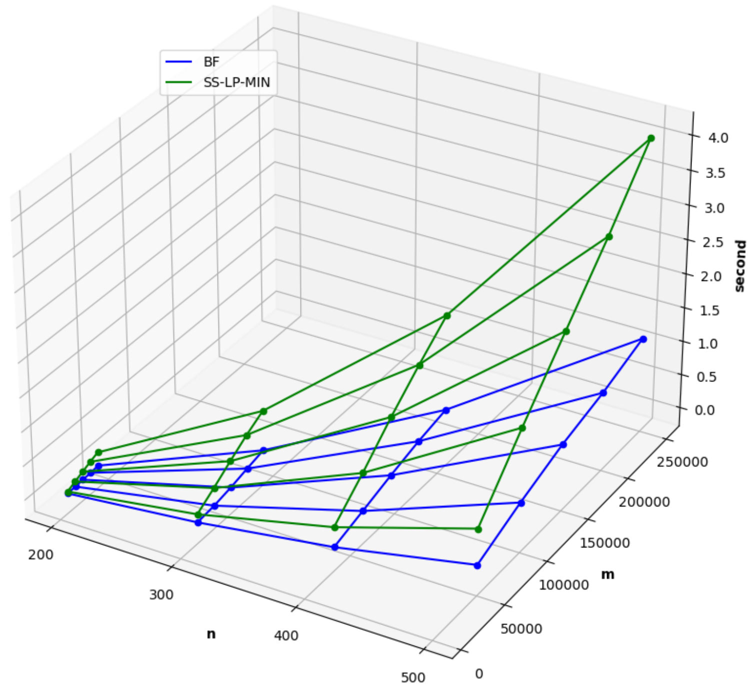


Figura 3.12: Tempi di computazione di BF e SS-LP-MIN nell'esperimento preliminare.

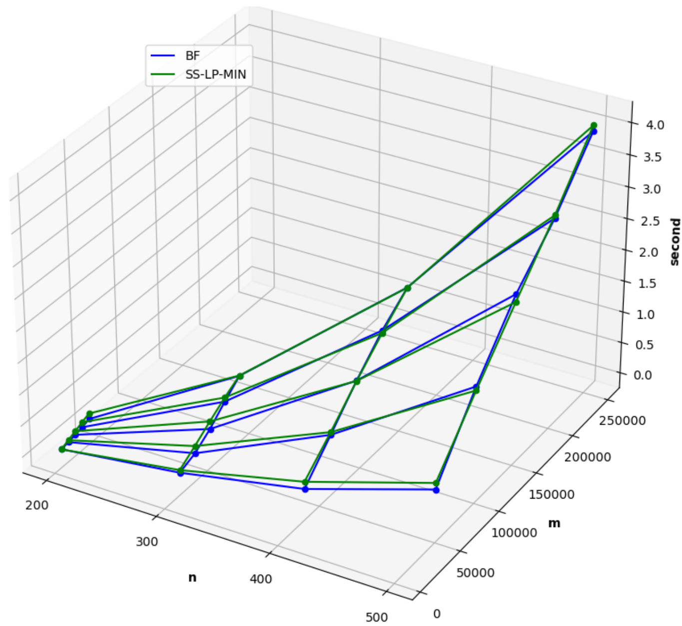


Figura 3.13: Tempi di computazione di BF (moltiplicati per 3.5) e SS-LP-MIN (non modificati) nell'esperimento preliminare.

Il secondo esperimento è stato effettuato eseguendo SS-LP-MIN su un dataset di 75 grafi. Il dataset utilizzato è lo stesso utilizzato per il secondo esperimento effettuato per la Sezione 3.2. I tempi di calcolo misurati su grafi con lo stesso numero di nodi e archi sono stati accorpati con una media aritmetica. In Figura 3.14 è riportato il grafico del tempo di calcolo in secondi di SS-LP-MIN.

Per verificare l'ipotesi che il costo computazionale di SS-LP-MIN abbia un comportamento asintotico di $\Theta(nm)$, sui tempi di calcolo di SS-LP-MIN ottenuti nel secondo esperimento è stato fatto fitting di una funzione del tipo $f(n,m) = c_1nm + c_2n + c_3m + c_4$ con $c_1, c_2, c_3, c_4 \in \mathbb{R}$. La miglior funzione che approssima i dati è:

$$\begin{aligned} f(n,m) &= c_1nm + c_2n + c_3m + c_4 \\ c_1 &= +3.45137209 \times 10^{-8} \\ c_2 &= -4.19947833 \times 10^{-6} \\ c_3 &= +9.49602114 \times 10^{-4} \\ c_4 &= -9.82027886 \times 10^{-2} \end{aligned} \tag{3.2}$$

In Figura 3.15 è riportato come la Funzione 3.2 approssima i tempi di calcolo di SS-LP-MIN. Nelle Figure 3.16a 3.16c 3.16e sono riportate delle sezioni del grafico in Figura 3.7 con n fissato: $n = 1500$, $n = 2000$ e $n = 2500$ rispettivamente. Graficamente la Funzione 3.2 approssima bene i dati.

Nelle Figure 3.16b 3.16d 3.16f è riportato il residuo della Funzione 3.2 con n fissato: $n = 1500$, $n = 2000$ e $n = 2500$ rispettivamente. Il residuo è calcolato come la differenza tra il valore del dato osservato e il valore della funzione che approssima i dati. I residui appaiono sparsi in modo casuale attorno allo zero, indicando che la Funzione 3.2 descrive bene i dati.

Dai grafici evince che la Funzione 3.2 approssima bene i tempi di calcolo di SS-LP-MIN e quindi si può concludere che il costo computazionale asintotico di SS-LP-MIN sia $\Theta(nm)$.

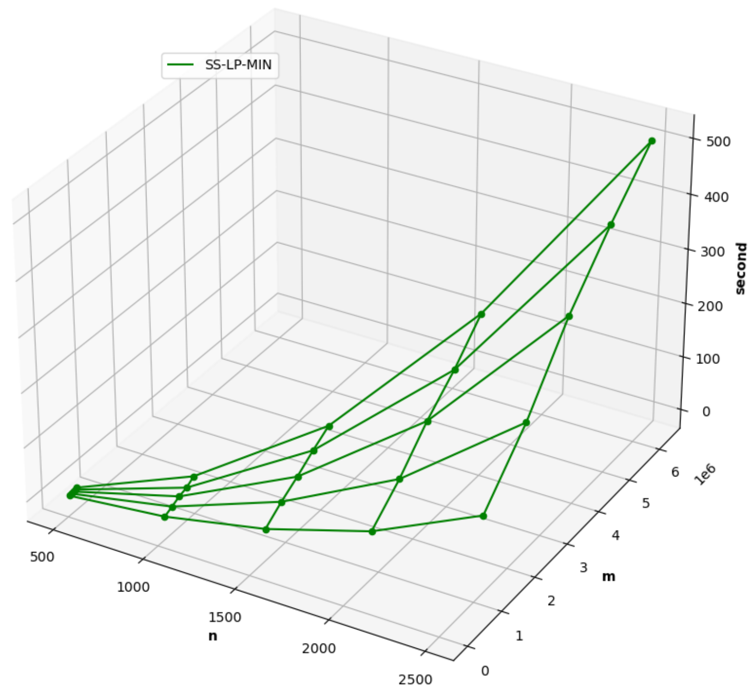


Figura 3.14: Tempi di computazione di SP-LP-MIN nel secondo esperimento.

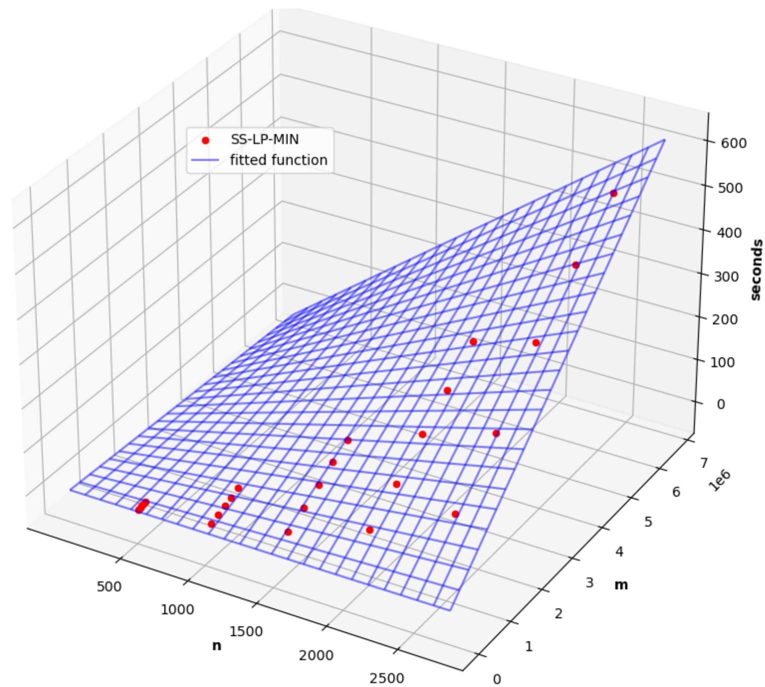
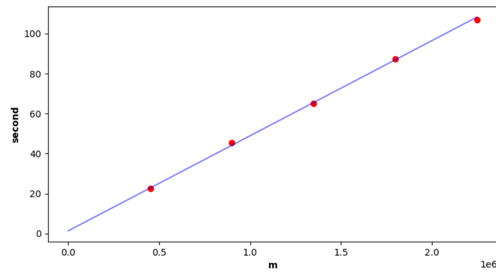
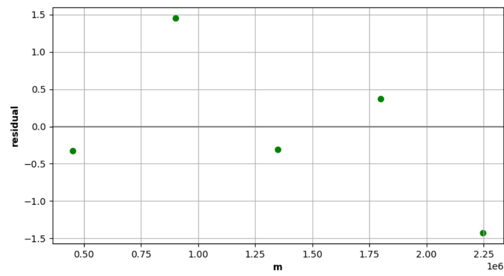


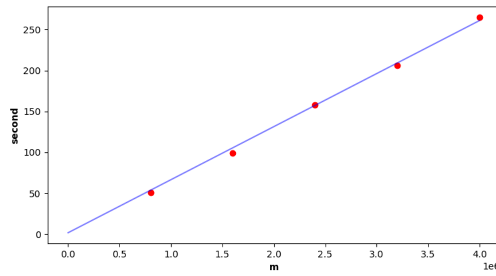
Figura 3.15: Funzione che approssima i tempi di computazione di SP-LP-MIN.



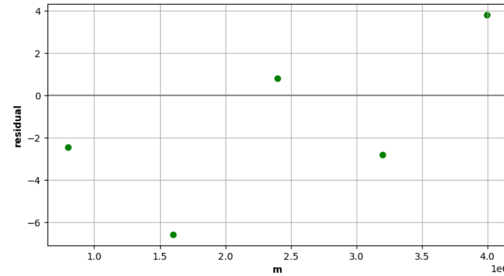
(a) $n = 1500$



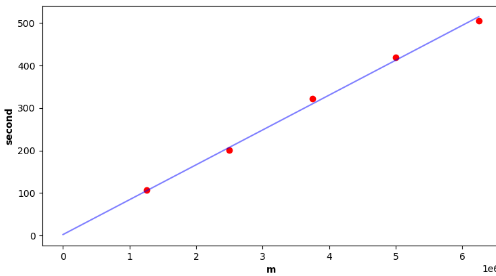
(b) residuo, $n = 1500$



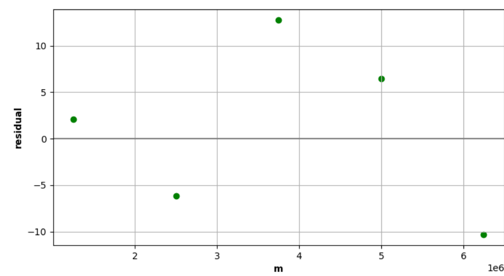
(c) $n = 2000$



(d) residuo, $n = 2000$



(e) $n = 2500$



(f) residuo, $n = 2500$

Figura 3.16: Sezioni del grafico della funzione che approssima i tempi di computazione di SS-LP-MIN fissando la variabile n.

3.4 Cammini Minimi fra Tutte le Coppie

In questa sezione sono stati effettuati dei benchmark sul modello in programmazione lineare 1.5 per risolvere il problema dei cammini minimi fra tutte le coppie di nodi un grafo. Sono stati misurati i tempi di calcolo e analizzati i risultati ottenuti. Sono stati eseguiti gli stessi benchmark anche sugli algoritmi Floyd-Warshall e Johnson, i tempi ottenuti sono stati utilizzati per effettuare un confronto con i tempi ottenuti dal Modello 1.5. In Tabella 3.3 sono riassunti il modello in programmazione lineare e gli algoritmi sui quali sono stati effettuati i benchmark.

Nome	**Breve descrizione**
AP-LP	Formulazione in programmazione lineare con minimizzazione della funzione obbiettivo. Vedi Modello 1.5.
FW	Floyd Warshall. Vedi (Cormen u. a., 2009, p. 693), (Floyd, 1962).
JS	Johnson: Bellman Ford per riscrittura pesi poi Dijkstra (con heap binario) a partire da ogni vertice. Vedi (Cormen u. a., 2009, p. 700), (Johnson, 1977).

Tabella 3.3: Riassunto dei modelli di programmazione lineare e algoritmi per cammini minimi fra tutte le coppie sui quali sono stati effettuati i benchmark.

È stato effettuato un solo esperimento dove ogni algoritmo e modello in programmazione lineare in Tabella 3.3 è stato eseguito sullo stesso dataset di 84 grafi. Il dataset è stato costruito generando 3 grafi per ogni coppia (n, m) con $n \in \{40, 70, 100, 130, 165, 200\}$ e m scelto nel seguente modo: per ogni n, se n è minore o uguale a 130, m è stato scelto prendendo 5 numeri naturali equispaziati nell'intervallo tra n e $n(n-1)$, invece se n maggiore o uguale a 165, m è stato scelto prendendo 4 numeri naturali equispaziati nell'intervallo tra n e $n(n-1)$. Per ogni algoritmo e modello, i tempi di calcolo misurati su grafi con lo stesso numero di nodi e archi sono stati accorpati con una media aritmetica.

In Figura 3.17, 3.18 e 3.19 sono riportati i grafici dei tempi di calcolo in secondi al variare di n e m, rispettivamente di AP-LP, FW e JS. AP-LP raggiunge molto più velocemente tempi di calcolo più elevati, il grafico di AP-LP raggiunge un valore massimo di 6705.22 secondi mentre i grafici di FW e JS raggiungono rispettivamente soltanto un valore massimo di 0.01813 e 0.1117 secondi. Nonostante ciò per capire se l'andamento asintotico del grafico di AP-LP sia lo stesso di FW o JS, si è verificato se il grafico di FW o JS siano uguali a grafico di AP-LP a meno di una costante moltiplicativa, cioè si sono cercate due costante $c_1, c_2 \in \mathbb{R}$ tali che moltiplicando tutti

i tempi di calcolo di FW per c_1 otteniamo un grafico quasi sovrapponibile a quello di AP-LP e moltiplicando tutti tempi di calcolo di JS per c_2 otteniamo un grafico quasi sovrapponibile a quello di AP-LP. Le costanti c_1 e c_2 sono state ottenute nel seguente modo:

$$\forall i \in \{1..84\} \;\; z_i = \text{tempo di calcolo di AP-LP sull'i-esimo grafo nel dataset}$$
$$\forall i \in \{1..84\} \;\; \hat{z}_i = \text{tempo di calcolo di FW sull'i-esimo grafo nel dataset}$$
$$\forall i \in \{1..84\} \;\; \bar{z}_i = \text{tempo di calcolo di JS sull'i-esimo grafo nel dataset}$$

$$c_1 = \operatorname*{argmin}_{c \in \mathbb{R}} \sum_{i=1}^{84} (z_i - c\hat{z}_i)^2$$
$$c_2 = \operatorname*{argmin}_{c \in \mathbb{R}} \sum_{i=1}^{84} (z_i - c\bar{z}_i)^2$$

In Figura 3.20 c'è il grafico di FW moltiplicato per $c_1 = 33434.158$ confrontato con il grafico di AP-LP e in Figura 3.21 c'è il grafico di JS moltiplicato per $c_2 = 7270.9$ confrontato con il grafico di AP-LP. In entrambi i casi i grafici non sono sovrapposti e quindi non sono uguali a meno di una costante moltiplicativa.

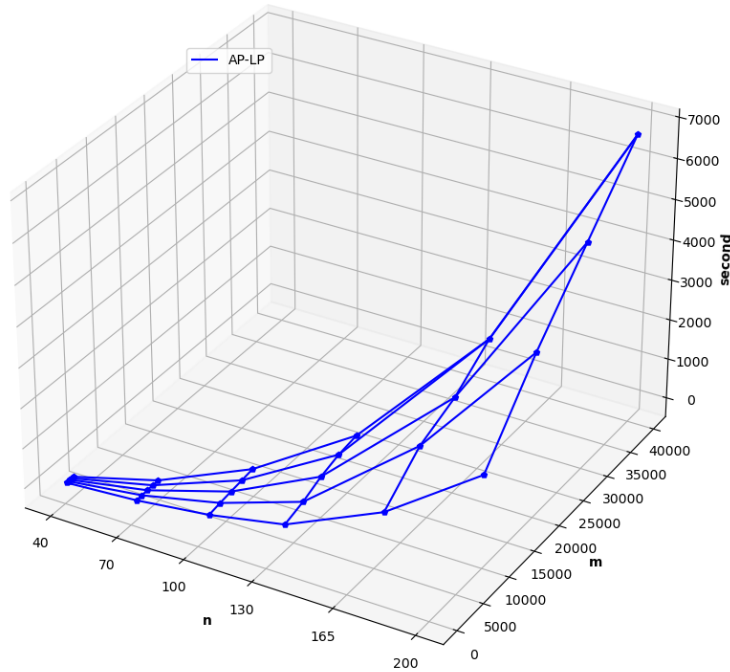


Figura 3.17: Tempi computazione di AP-LP.

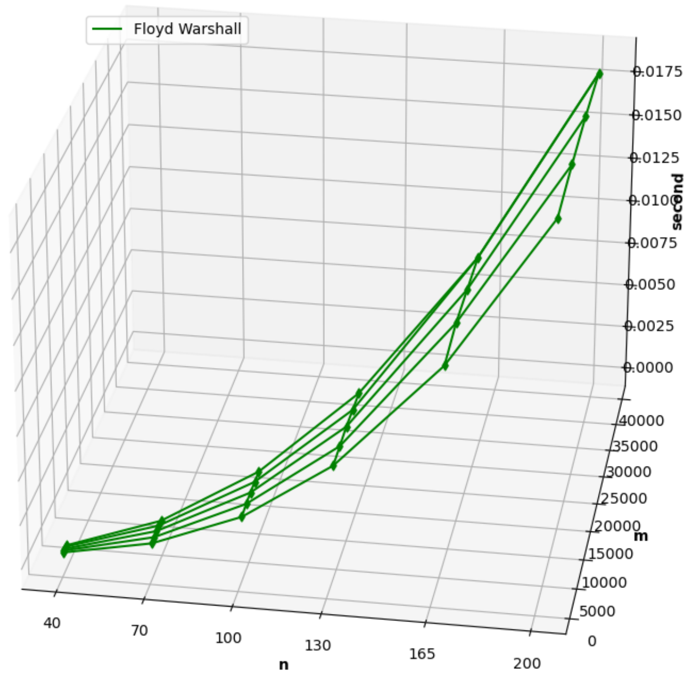


Figura 3.18: Tempi computazione di FW.

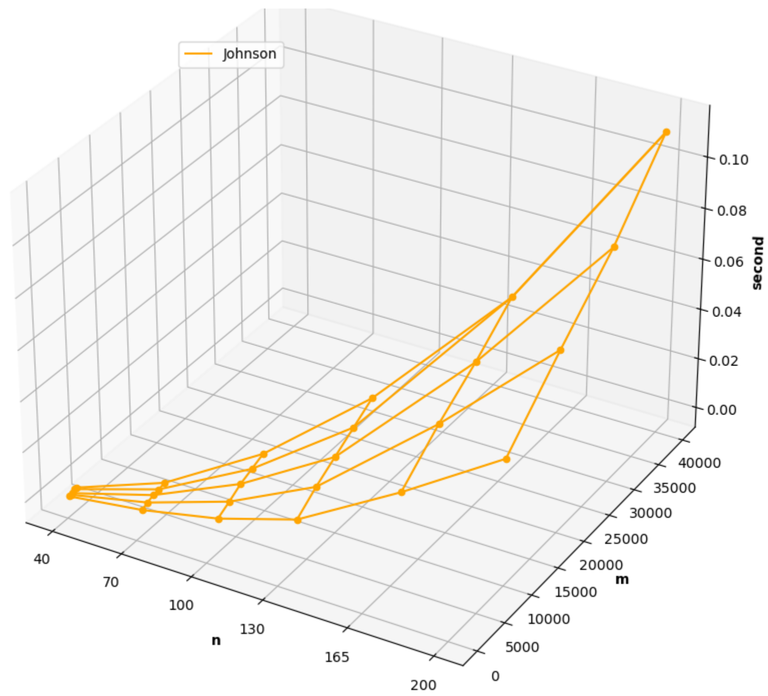


Figura 3.19: Tempi computazione di JS.

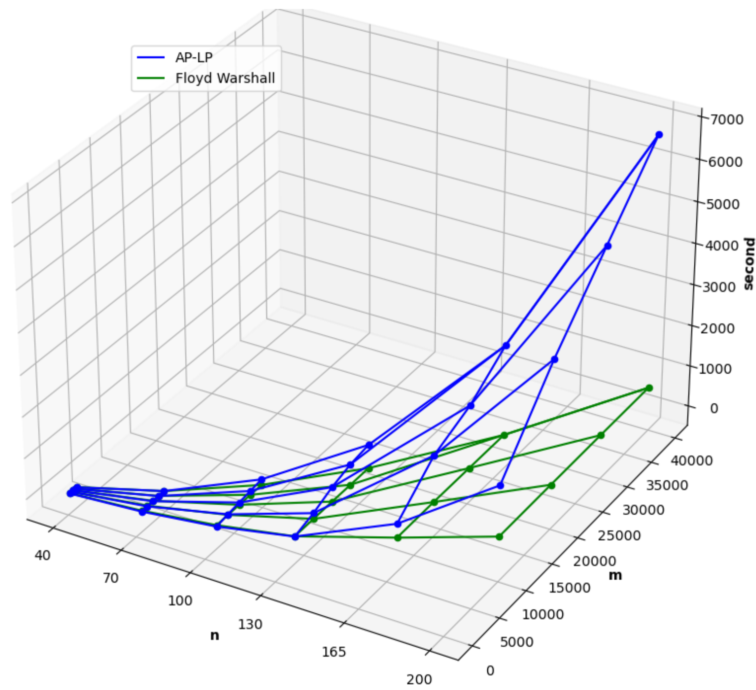


Figura 3.20: Tempi di computazione di FW (moltiplicati per $c_1 = 33434.158$) e AP-LP (non modificati).

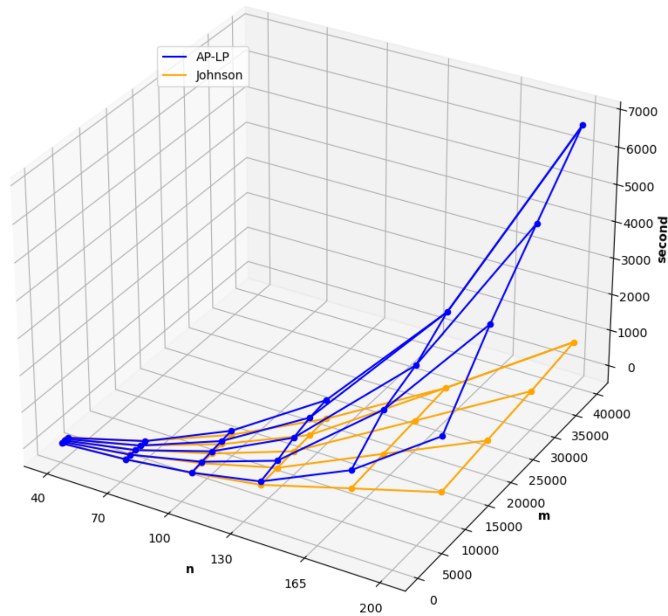


Figura 3.21: Tempi di computazione di JS (moltiplicati per $c_2 = 7270.9$) e AP-LP (non modificati).

Successivamente è stata fatta una analisi di fitting soltanto sul grafico di AP-LP per trovare una funzione $f(n,m)$ che ne approssima i tempi di calcolo. Nelle Figure 3.4 sono riportate delle sezioni del grafico in Figura 3.17 con n fissato: $n = 40$, $n = 70$, $n = 100$, $n = 130$, $n = 165$ e $n = 200$. Inoltre nelle Figure 3.4 è riportata la miglior retta della forma $c_1 m + c_2$ con $c_1, c_2 \in \mathbb{R}$, ottenuta tramite i minimi quadrati, che approssima i tempi di calcolo. In Tabella 3.4 sono riportati i coefficienti c_1 e c_2 trovati per ognuna della 6 rette.

n	c_1	c_2
40	0.0010168859620845376	0.03565789795400474
70	0.007250140053857379	-0.5821098039252675
100	0.021658520406029558	8.636781292533659
130	0.04561281049471322	29.49656796881156
165	0.08928332348936457	133.32853378560964
200	0.16374681793140236	240.43231046575553

Tabella 3.4: Coefficiente (c_1) e quota (c_2) della miglior retta che approssima le sezioni del Grafico 3.17 con n fissato.

Si osserva che per ognuna delle 6 figure il tempo cresce in modo lineare al crescere del numero degli archi, Quindi il numero degli archi m deve comparire nella funzione $f(n,m)$ soltanto con esponente 1.

La funzione $f(n,m)$ che approssima i tempi di calcolo, per quanto appena osservato, deve essere della forma: $f(n,m) = g(n)m + h(n)$ per qualche funzione $g(n)$ e $h(n)$.

Se $f(n,m) = g(n)m + h(n)$ allora fissando n otteniamo che $g(n)$ diventa il coefficiente angolare di una retta. Per individuare la funzione $g(n)$ avendo il Grafico 3.17 possiamo fissare la variabile n ad intervalli regolari, trovare la miglior retta che approssima le sezioni del Grafico 3.17 per ogni n fissato, fare un grafico con sulle ascisse il numero dei nodi n e sulle ordinate il coefficiente angolare della miglior retta che approssima la sezione del Grafico 3.17 con n fissato e infine trovare la funzione che approssima i punti del grafico dei coefficienti angolari appena costruito.

In Figura 3.23 sono mostrati i coefficienti angolari delle rette nelle Figure 3.4 al variare del valore fissato di n . I coefficienti angolari in Figura 3.23 sono approssimati bene dalla funzione cn^3 con $c = 2.03651852 \times 10^{-8}$.

Inoltre se $f(n,m) = g(n)m+h(n)$ allora fissando n otteniamo che $h(n)$ diventa la quota di una retta. Per individuare la funzione $h(n)$ avendo il Grafico 3.17 possiamo fissare la variabile n ad intervalli regolari, trovare la miglior retta che approssima le

sezioni del Grafico 3.17 per ogni n fissato, fare un grafico con sulle ascisse il numero dei nodi n e sulle ordinate la quota della miglior retta che approssima la sezione del Grafico 3.17 con n fissato e infine trovare la funzione che approssima i punti del grafico delle quote appena costruito.

In Figura 3.24 sono mostrati le quote delle rette nelle Figure 3.4 al variare del valore fissato di n .

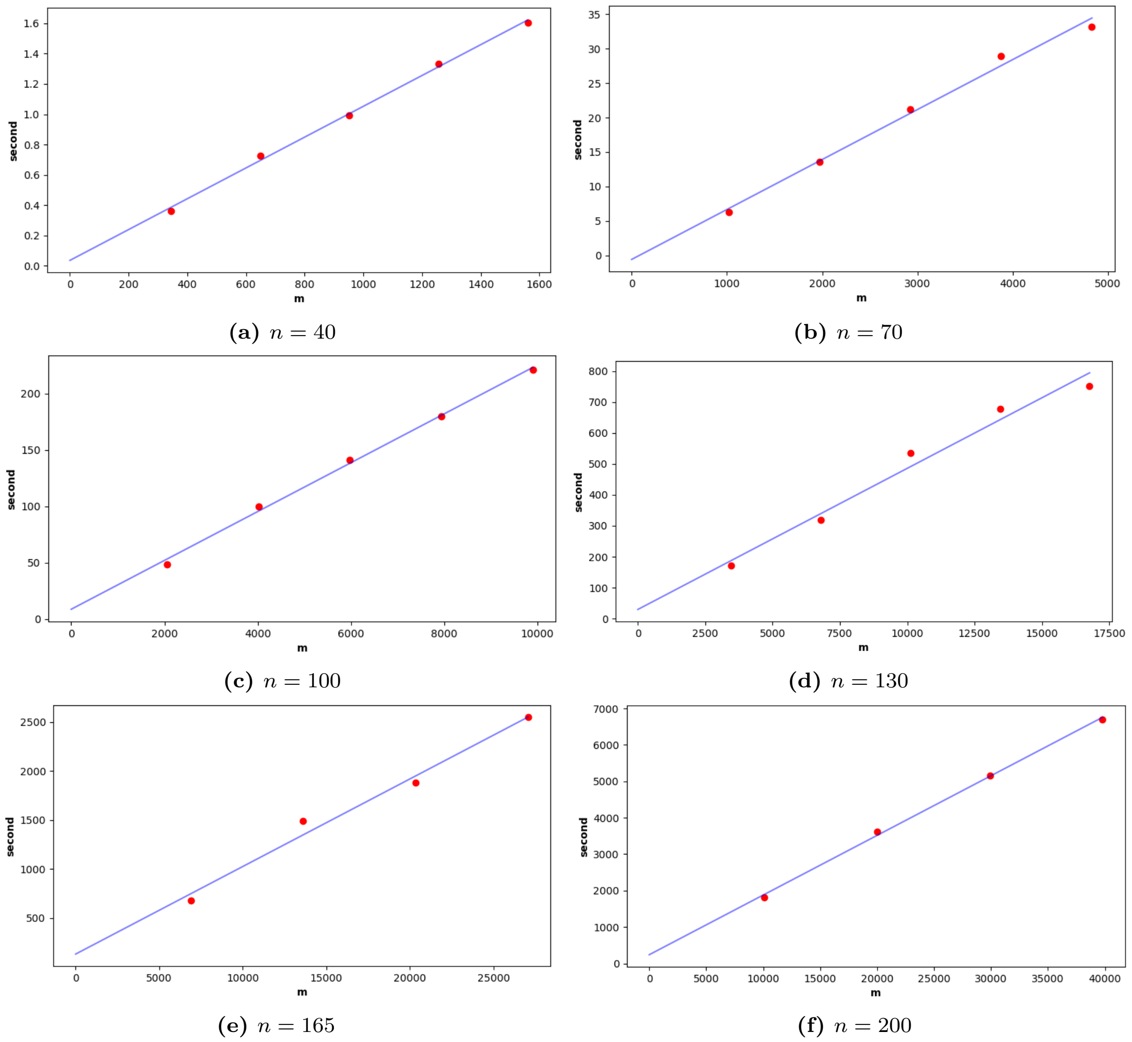
Le quote in Figura 3.24 sono approssimate dalla funzione $c_1n^3 + c_2n^2 + c_3$ con $c_1 = 5.26871007 \times 10^{-5}$, $c_2 = -4.43532057 \times 10^{-3}$ e $c_3 = 1.59895423$.

In conclusione, sui tempi di calcolo di AP-LP è stato fatto fitting di funzione del tipo $f(n, m) = c_1n^3m + c_2n^3 + c_3n^2 + c_4$ con $c_1, c_2, c_3, c_4 \in \mathbb{R}$. La funzione che approssima i dati è:

$$\begin{aligned} f(n, m) &= c_1n^3m + c_2n^3 + c_3n^2 + c_4 \\ c_1 &= +2.0425056115593598 \times 10^{-8} \\ c_2 &= +6.417216291894058 \times 10^{-5} \\ c_3 &= -0.007088847750291314 \\ c_4 &= 12.62289457269237 \end{aligned} \tag{3.3}$$

In Figura 3.25 è riportato come la Funzione 3.3 approssima i tempi calcolo di AP-LP. Graficamente la Funzione 3.3 approssima bene i tempi di calcolo di AP-LP.

Dai grafici evince che la Funzione 3.3 approssima bene i tempi di calcolo di AP-LP e quindi si può concludere che il costo computazionale asintotico di AP-LP sia $\Theta(n^3m)$.



(a) $n = 40$ **(b)** $n = 70$

(c) $n = 100$ **(d)** $n = 130$

(e) $n = 165$ **(f)** $n = 200$

Figura 3.22: Sezioni del grafico della funzione che approssima i tempi di computazione di AP-LP fissando la variabile n.

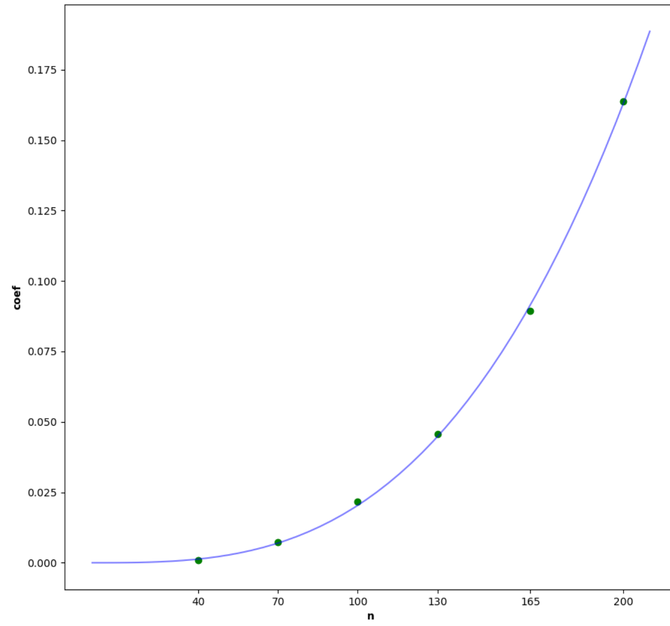


Figura 3.23: Coefficienti angolari al variare di n, delle rette nelle Figure 3.4.

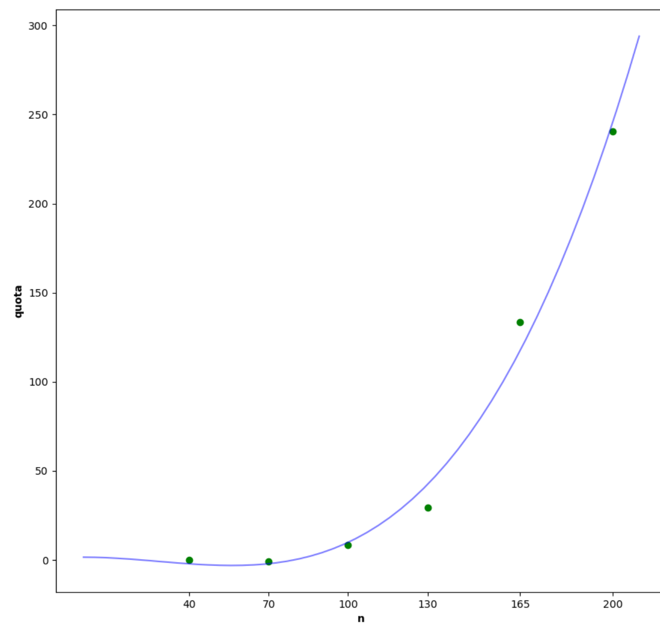


Figura 3.24: Quota al variare di n, delle rette nelle Figure 3.4.

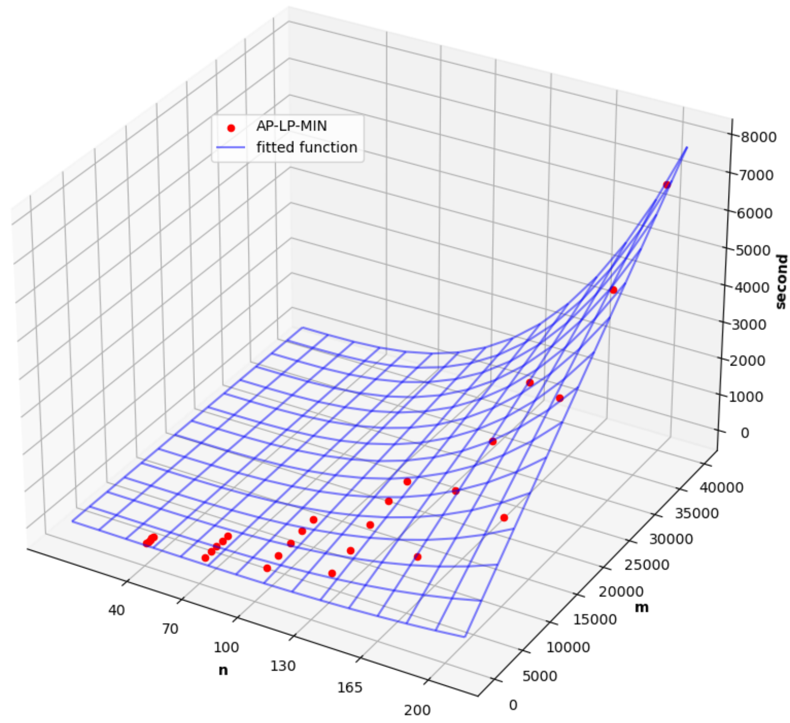


Figura 3.25: Funzione che approssima i tempi di computazione di AP-LP.

3.5 Albero di Copertura Minimo

In questa sezione sono stati effettuati alcuni benchmark su modelli in programmazione lineare, elencati in Tabella 3.5, per risolvere il problema dell'albero di copertura di costo minimo in un grafo non diretto. Sono stati misurati i tempi di calcolo e analizzati i risultati ottenuti.

Nome	**Breve descrizione**
NL	Formulazione in programmazione lineare con livello assegnato a nodi. Vedi Modello 1.7.
MF	Formulazione in programmazione lineare con flussi multipli. Vedi Modello 1.6.

Tabella 3.5: Riassunto dei modelli di programmazione lineare per il problema dell'albero di copertura minimo sui quali sono stati effettuati i benchmark.

Inizialmente è stato effettuato un esperimento preliminare dove ogni modello in programmazione lineare in Tabella 3.5 è stato eseguito sullo stesso dataset di 60 grafi. Il dataset è stato costruito generando 3 grafi per ogni coppia (n, m) con $n \in \{9, 16, 23, 30\}$ e m scelto nel seguente modo: per ogni n, m è stato scelto prendendo 5 numeri naturali equispaziati nell'intervallo tra n e $\frac{n(n-1)}{2}$. Per ogni modello, i tempi di calcolo misurati su grafi con lo stesso numero di nodi e archi sono stati accorpati con una media aritmetica.

In Figura 3.26 e 3.27 sono riportati rispettivamente lo scatter plot e i grafici dei tempi in secondi di NL e MF ottenuti al variare di n e m. MF raggiunge molto più velocemente tempi di calcolo più elevati, il grafico di MF raggiunge un valore massimo di 1396.546 secondi in corrispondenza di un grafo con 30 nodi e 435 archi, mentre NL raggiunge soltanto un tempo massimo di 0.048 secondi un corrispondenza di un grafo con 23 nodi e 161 archi. Il grafico di MF cresce molto più rapidamente di quello di NL e per questo motivo è stato scartato da ulteriori esperimenti. È stato effettuato un secondo esperimento con grafi di dimensione maggiore per analizzare meglio l'andamento del costo computazionale di NL.

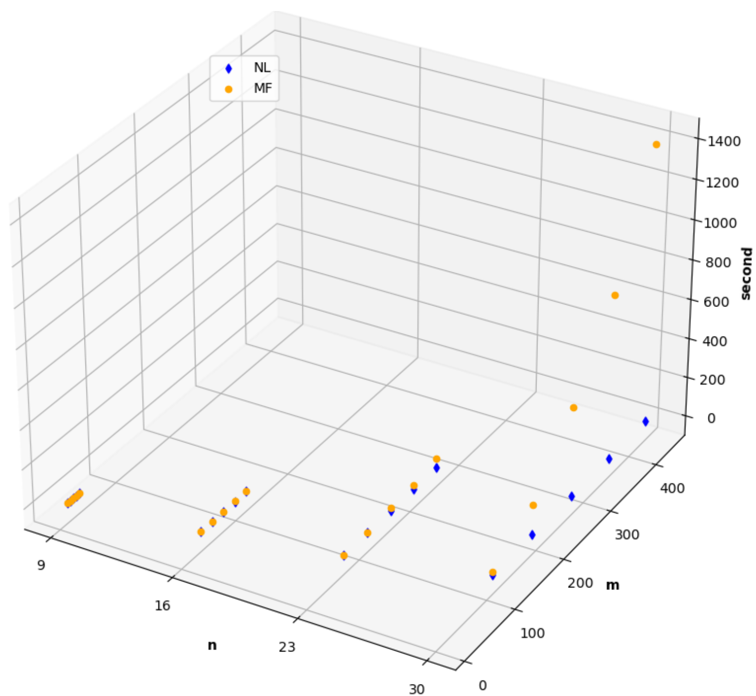


Figura 3.26: Scatter plot dei tempi di computazione di NL e MF nell'esperimento preliminare.

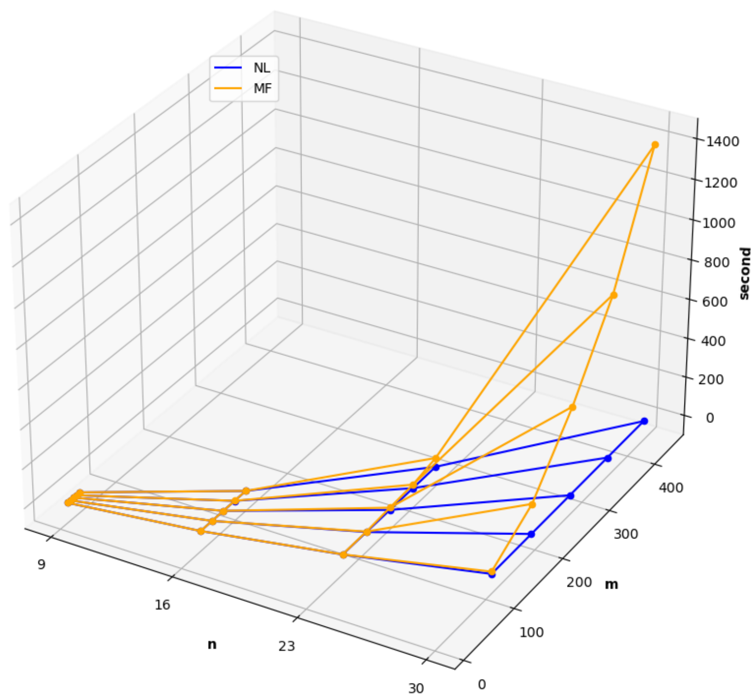


Figura 3.27: Grafico dei tempi di computazione di NL e MF nell'esperimento preliminare.

Il secondo esperimento è stato effettuato eseguendo NL su un dataset di 450 grafi. Il dataset è stato costruito generando 10 grafi per ogni coppia (n, m) con $n \in \{20, 30, 40, 50, 60, 70, 80, 90, 100\}$ e m scelto nel seguente modo: per ogni n, m è stato scelto prendendo 5 numeri naturali equispaziati nell'intervallo tra n e $\frac{n(n-1)}{2}$.

In Figura 3.28 è riportato lo scatter plot dei tempi in secondi di NL ottenuti al variare di n e m nel secondo esperimento. I tempi in Figura 3.28 non sono stati accorpati in nessun modo e con nessun tipo di media.

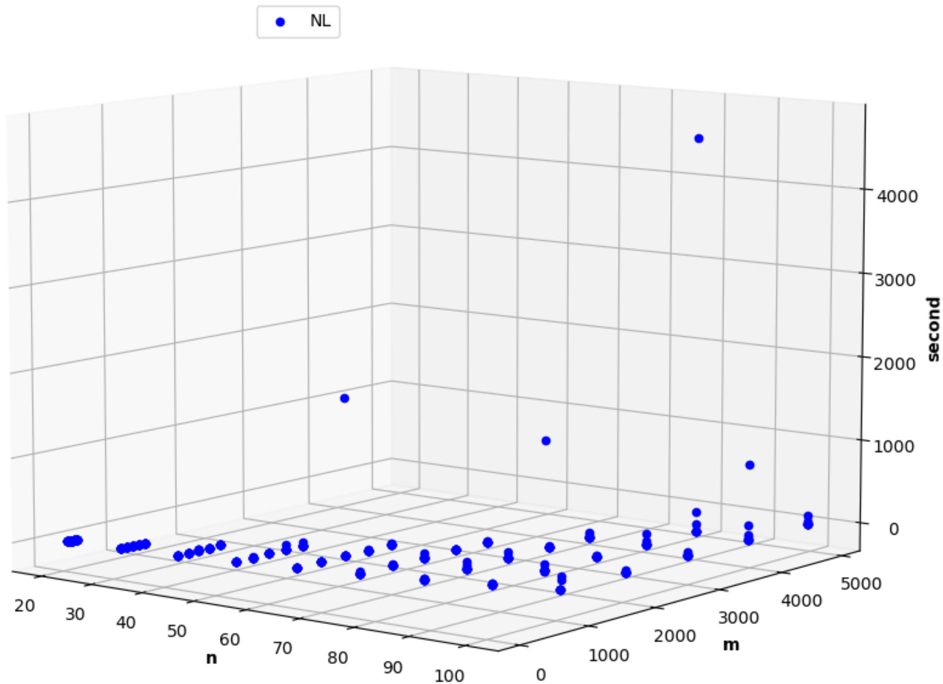


Figura 3.28: Scatter plot dei tempi di computazione di NL nel secondo esperimento.

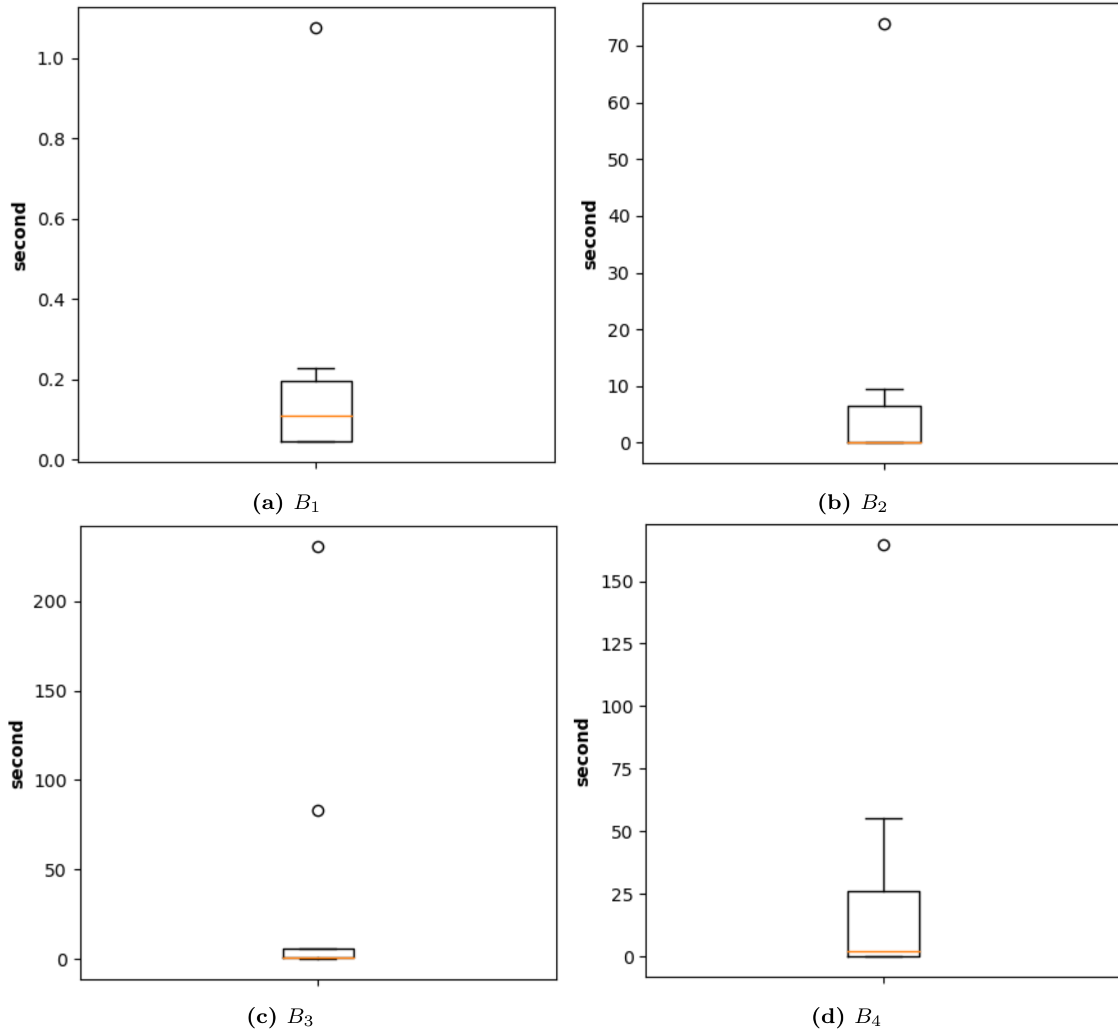
Dalla Figura 3.28 si osserva che per alcuni grafi il tempo di calcolo di NL si comporta in modo anomalo rispetto ai tempi misurati per tutti gli altri grafi, sia per grafi con diverso numero di nodi e archi, sia per grafi uguale numero di nodi e archi. In particolare ci sono 4 grafi in tutto il dataset per i quali il tempo di calcolo di NL è molto maggiore rispetto al tempo di calcolo di tutti gli altri grafi nel dataset. Chiamiamo G_1, G_2, G_3, G_4 questi 4 grafi che si comportano in modo anomalo e in Tabella 3.6 è riportato per ognuno dei 4 grafi il numero di nodi, il numero di archi e il tempo di calcolo misurato.

	Nodi	**Archi**	**Tempo di calcolo (secondi)**
G_1	60	1086	1840.427
G_2	90	1656	1507.295
G_3	90	4005	4662.476
G_4	100	3980	895.986

Tabella 3.6

Nelle Figure 3.29a, 3.29b, 3.29c e 3.29d sono riportati rispettivamente i boxplot B_1, B_2, B_3, B_4 dei tempi di calcolo in secondi di NL ciascuno sui 9 grafi appartenenti al dataset, scelti nel seguente modo: $\forall i \in \{1, 2, 3, 4\}$ B_i contiene tutti e soli i grafi nel dataset con numero di nodi e archi uguale a G_i ma diversi da G_i.

Dalle Figure 3.29a, 3.29b, 3.29c e 3.29d si può osservare che $\forall i \in \{1, 2, 3, 4\}$ il tempo di calcolo di NL sul grafo G_i ha proprio un ordine di grandezza diverso rispetto ad ogni altro tempo di calcolo di NL su un grafo con stesso numero di nodi e archi di G_i.



(a) B_1 **(b)** B_2

(c) B_3 **(d)** B_4

Il motivo di perché NL impieghi un tempo anomalo per trovare l'albero di costo minimo su G_1, G_2, G_3, G_4 come input, può essere ricondotto a una qualche struttura del grafo in cui nodi e archi si collegano in un qualche modo particolarmente sfavorevole per il tempo di esecuzione. Oppure può essere provocato anche da un qualche assegnamento dei pesi archi che influisce negativamente sul tempo di esecuzione.

A causa di questi tempi anomali non è stata individuata una funzione che approssimi con precisione i tempi di calcolo di NL.

In conclusione la formulazione in programmazione lineare intera NL è sicuramente più computazionalmente efficiente della formulazione MF, inoltre è stato osservato che su 4 grafi sui 450 totali del dataset del secondo esperimento la formulazione NL ha impiegato un tempo esecuzione troppo elevato mentre sui restanti grafi il tempo esecuzione impiegato è ragionevole.

Appendice A

Patch Glpsol, Calcolo Wall Time con clock_gettime e clock_monotonic_raw

```
--- glpsol.c 2023-09-22 17:23:28.804114395 +0200
+++ glpsol-patched.c 2023-09-22 17:25:48.104120126 +0200
@@ -37,6 +37,26 @@
 #define xerror  glp_error
 #define xprintf glp_printf

+#include <time.h>
+#include <stdint.h>
+
+#define get_timestamp(timespec_ptr) \
+    if (clock_gettime(CLOCK_MONOTONIC_RAW, timespec_ptr) == -1) { \
+        xerror("Error: clock_gettime failed."); \
+    }
+
+#define diff_timestamp(timespec_ptr_start, \
+        timespec_ptr_end, timespec_ptr_diff) { \
+    (timespec_ptr_diff)->tv_sec = \
+        (timespec_ptr_end)->tv_sec - (timespec_ptr_start)->tv_sec; \
+    (timespec_ptr_diff)->tv_nsec = \
+        (timespec_ptr_end)->tv_nsec - (timespec_ptr_start)->tv_nsec; \
+    if ((timespec_ptr_diff)->tv_nsec < 0) { \
+        (timespec_ptr_diff)->tv_nsec += 1000000000; \
+        (timespec_ptr_diff)->tv_sec--; \
+    } \
```

```
+}
+
 struct csa
 {     /* common storage area */
       glp_prob *prob;
@@ -959,7 +979,7 @@
 #if 0 /* 10/VI-2013 */
       glp_long start;
 #else
-      double start;
+      struct timespec start, end, diff;
 #endif
       /* perform initialization */
       csa->prob = glp_create_prob();
@@ -1368,7 +1388,7 @@
       }
       /*--------------------------------------------------------------*/
       /* solve the problem */
-      start = glp_time();
+      get_timestamp(&start);
       if (csa->solution == SOL_BASIC)
       {  if (!csa->exact)
          {  glp_set_bfcp(csa->prob, &csa->bfcp);
@@ -1421,8 +1441,10 @@
          xassert(csa != csa);
      /*--------------------------------------------------------------*/
      /* display statistics */
-      xprintf("Time used:   %.1f secs\n", glp_difftime(glp_time(),
-         start));
+      get_timestamp(&end);
+      diff_timestamp(&start, &end, &diff);
+      xprintf("Time used:   %jd.%03ld secs\n",
+              (intmax_t)diff.tv_sec, diff.tv_nsec / 1000000);
 #if 0 /* 16/II-2012 */
       {  glp_long tpeak;
          char buf[50];
```

Bibliografia

[Abdelmaguid 2018] Abdelmaguid, Tamer F.: An efficient mixed integer linear programming model for the Minimum spanning tree problem. In: *Mathematics (Basel)* 6 (2018), Nr. 10, S. 183. – ISSN 2227-7390

[András Éles 2019] András Éles, György D.: *Tutorial for Operations Research: Modeling in GNU MathProg*. 2019. – URL `https://tananyagfejlesztes.mik.uni-pannon.hu/index.php/component/phocadownload/category/1-algoritmusok-operaciokutatas?download=111:tutorial-for-operations-research-modeling-in-gnu-mathprog`

[Beyer u. a. 2019] Beyer, Dirk ; Löwe, Stefan ; Wendler, Philipp: Reliable benchmarking: requirements and solutions. In: *International journal on software tools for technology transfer* 21 (2019), Nr. 1, S. 1–29. – ISSN 1433-2779

[Cormen u. a. 2009] Cormen, Thomas H. ; Leiserson, Charles E. ; Rivest, Ronald L. ; Stein, Clifford: *Introduction to Algorithms (3rd Edition)*. Cambridge : MIT Press, 2009 (The MIT Press). – xix–xix S. – ISBN 9780262033848

[Floyd 1962] Floyd, Robert: *Algorithm 97: Shortest path*. 1962

[Johnson 1977] Johnson, Donald: Efficient Algorithms for Shortest Paths in Sparse Networks. In: *Journal of the ACM* 24 (1977), Nr. 1, S. 1–13. – ISSN 0004-5411

[Makhorin 2020] Makhorin, Andrew: *GLPK (GNU Linear Programming Kit)*. 2020. – URL `https://www.gnu.org/software/glpk/`