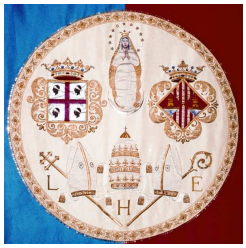


Università degli Studi di Cagliari
Facoltà di Ingegneria
Dipartimento di Ingegneria Elettrica ed Elettronica
Corso di Laurea Specialistica in Ingegneria Elettronica

Tecniche di ottimizzazione per l'analisi della diffusione delle innovazioni nei social networks

Relatori:
Alessandro Giua
Carla Seatzu

Tesi di Laurea di:
Matteo Secci

Anno Accademico 2011/2012

Se un uomo parte da certezze, finirà nel dubbio, ma se accetta di iniziare con i dubbi, troverà certezze.

Francesco Bacone (1561-1626), *Il progresso del sapere* .

Alla mia famiglia.

Riassunto della tesi

Il tema centrale di questa tesi è il problema della massimizzazione dell'influenza nei social networks. In particolare sono stati implementati in linguaggio OPL i seguenti problemi:

1. Identificare il set di innovatori di partenza di numero pari ad r che massimizzano la diffusione dell'innovazione in un orizzonte temporale limitato.

2. Identificare il set di innovatori di partenza più piccolo che consente di diffondere l'innovazione attraverso un target di utenti in un orizzonte temporale limitato.

3. Identificare il set coesivo massimale a partire da un set di innovatori.

I problemi sopracitati presentano un'elevata complessità computazionale, in particolare per il primo problema sono state sviluppate le seguenti strategie:

- Un algoritmo greedy implementato in Matlab basato sulla costruzione di grafi aciclici locali per stimare l'evoluzione della rete analizzando dei processi che in essa si sviluppano localmente.

- Modifica del comportamento dell'algoritmo branch and cut del risolutore CPLEX in modo da ottenere una soluzione subottima con un carico computazionale minore.

Per quanto riguarda il secondo problema è stato implementato un algoritmo in Matlab che riduce la dimensione della rete basato sul fatto che non tutti i nodi sono in grado di raggiungere gli utenti obbiettivo nell'orizzonte temporale prefissato. Il terzo problema non è stato oggetto di alcuna ottimizzazione ma è stato comunque implementato in linguaggio OPL per completare il set di strumenti software per l'analisi della diffusione dell'innovazione nei social networks.

Summary

The central question of this thesis is influence maximization in social networks. In particular will be implemented in OPL the following problems:

1. Identify the set of r innovators which maximizes the diffusion of innovation in finite time.
2. Identify the smallest set of innovators wich maximize the spread of innovation through a target of users in finite time.
3. Identify the maximum choesive set.

These problems mentioned above are of type NP hard ie characterized by a computational time that increases non linearly with the size of the data. Related to problem 1 were developed following strategies:

- A greedy algorithm implemented in Matlab based on the construction of acyclic graphs of network by analyzing the processes developed locally.
- Change the behavior of the algorithm solver CPLEX branch & cut in order to improve the efficiency calculation.

For the second problem was implemented in Matlab an algorithm by reducing the size of the network to improve the performance of the simulator.

Indice

Elenco delle tabelle

Elenco delle figure

Capitolo 1

Introduzione

1.1 Obbiettivi della tesi

L'analisi dell'influenza nella società ha una serie di applicazioni reali. Una delle più importanti è certamente rappresentata dalla massimizzazione dell'influenza nelle applicazioni di *marketing virale*. La principale motivazione risiede nella necessità di individuare dei potenziali consumatori con lo scopo di minimizzare i costi derivanti dalle operazioni promozionali e massimizzare il profitto. Per esempio una compagnia vuole immettere sul mercato un nuovo prodotto facendo leva sulle relazioni interpersonali tra i consumatori. L'obbiettivo della compagnia è quello di convincere all'acquisto un piccolo insieme di utenti influenti della rete in modo da massimizzare la diffusione del proprio prodotto. Per raggiungere questi obbiettivi è necessario conoscere dei parametri che riassumano il comportamento di ciascun utente rispetto all'innovazione immessa sul mercato (per esempio il profitto percepito da ciascun utente) e il valore di ciascun utente (per esempio la sua capacità di persuadere all'acquisto altri utenti). Questi aspetti sono stati ampiamente studiati nell'ambito del *marketing decision*. Domingos e Richardson [9],[10] hanno formulato questi aspetti assegnando ad ogni utente un indice della loro influenza, il modello di diffusione adottato è quello cosidetto a rete di Markov. La formulazione di modelli di influenza e di diffusione risulta ancora un problema non completamente risolto. Importante risulta il lavoro di Kempe *et al.* [8] nel quale vengono trattati due importanti modelli di diffusione dell'innovazione e nel quale viene formalmente dimostrato che il problema della massimizzazione dell'influenza è di tipo NP hard. Questo aspetto risulta essere molto importante e in questa ottica sono stati proposti in letteratura numerosi approcci volti ad ottenere una soluzione approssimata con oneri computazionali ragionevoli. Oltre questo aspetto dal problema della massimizzazione dell'influenza e dalla sua formalizzazione nascono numerose interessanti varianti. In particolare in questa tesi focalizzeremo la nostra attenzione sulle estensioni proposte nel recente lavoro di Giua e Rosa

[23]. Le estensioni proposte sono le seguenti:

1. identificare il set di innovatori di partenza di numero pari ad r che massimizzano la diffusione dellinnovazione in un orizzonte temporale limitato.

2. identificare il set di innovatori di partenza più piccolo che consente di diffondere l'innovazione su un target di utenti in un orizzonte temporale limitato.

Oltre a questi problemi viene proposto un metodo risolutivo per la determinazione del set coesivo massimale della rete basato sulla risoluzione di una serie di problemi di programmazione lineare. La determinazione del set coesivo massimale dato un set di innovatori di partenza consente di definire lo stato finale della rete, ossia quali utenti adottano l'innovazione come mostrato in [7]. I problemi sono stati formalizzati nell'ambito del modello di *diffusione a soglia lineare*.

Nel presente lavoro di tesi sono stati in primo luogo implementati i tre problemi sopracitati in linguaggio OPL per il risolutore CPLEX con lo scopo di verificarne l'efficienza. Data la loro natura combinatoria i problemi sopracitati risultano poco trattabili per istanze reali ossia per reti di grandi dimensioni. Quindi un ulteriore ma non meno importante obbiettivo è stato quello di studiare dei metodi risolutivi che potessero rendere trattabili i problemi sopracitati. In particolare per il primo problema è stato sviluppato in Matlab un'euristica basata sulle proprietà dei grafi aciclici, e un approccio risolutivo basato sulla taratura del risolutore CPLEX, mentre dall'analisi del secondo problema è stato sviluppato un approccio basato sulla riduzione della dimensione della rete di partenza anch'essa implementata in Matlab.

1.2 Rassegna della letteratura

Lo studio della diffusione dell'innovazione nei social networks è un campo vastissimo e multidisciplinare. Tra i maggiori esperti ritroviamo non solo gli economisti industriali, ma anche e soprattutto gli studiosi di marketing gli storici di economia e altri ancora. Quindi le possibili combinazioni realizzabili riguardanti gli approcci metodologici e le definizioni dell'oggetto di studio sono molto numerose. Il problema fonda le sue radici sulla ricerca e analisi dei meccanismi del passaparola [20, 21, 9, 22, 10]. I primi modelli matematici riguardo la diffusione delle innovazioni risalgono agli anni 70' [1],[2] durante i quali viene proposto il modello a soglia lineare. Un altro tema interessante trattato in letteratura è quello delle interazioni locali tra individui di una rete: a tal proposito citiamo il lavori di Ellison e Morris [3],[5] dedicati al problema della coordinazione tra individui in giochi o competizioni. Oltre al modello di *diffusione a soglia lineare* citato precedentemente esistono altri modelli, un esempio è il *modello a cascata* introdotto per la prima volta nel contesto del marketing da Goldenberg *et al.* [24]. Un riassunto esaustivo sui principali modelli impiegati in letteratura viene mostrato nel lavoro di Wortman [6]. Il lavoro di tesi in particolare è incentrato su problemi e metodi basati sul modello a soglia lineare. Al riguardo risulta molto importante il recente lavoro proposto da Yildiz *et al.* [7], dove viene mostrata un estensione di un idea proposta in [5]. L'idea di fondo si basa sul concetto di gruppo coesivo. I membri appartenenti ad un set coesivo hanno la seguente caratteristica: Essi non adottano l'innovazione se nessuno degli innovatori appartiene al set coesivo stesso. Viene quindi mostrato come sia possibile determinare facilmente lo stato finale della rete dalla conoscenza del set coesivo massimale partendo da un set di innovatori di partenza. Nel recente lavoro di Giua *et al.* [23] vengono mostrati due metodi per determinare il set coesivo massimale basato sulla programmazione matematica. Il primo metodo si basa su un problema di ottimizzazione discreto e data la sua natura combinatoria viene proposto un metodo alternativo basato sulla risoluzione di una serie di problemi rilassati che portano alla soluzione ottima. Un altro problema fondamentale trattato in letteratura è quello della massimizzazione dell'influenza. Esso consiste nella ricerca di un set di innovatori di dimensione fissata, in base ad esempio al budget della compagnia che vuole attuare una strategia di marketing, che massimizza la diffusione del prodotto o innovazione. Questo problema viene ampiamente trattato nel lavoro di Kempe *et al.*[8] dove in particolare vengono mostrati i seguenti risultati:

- Il problema della massimizzazione dell'influenza è di tipo NP hard sia per il modello a soglia lineare che per il modello a cascata indipendente.
- Viene proposto un approccio approssimato basato sulla costruzione di un algoritmo greedy.

In letteratura sono stati proposti numerosi approcci approssimati [11, 12, 10, 9]. Nel lavoro di tesi in particolare faremo riferimento all'approccio euristico proposto in [11] che sfutta anch'esso le proprietà di supermodularità e monotonicità citate nel lavoro di Kempe *et al.* [8]. In particolare l'approccio proposto si prefigge di stimare il numero atteso di utenti che adottano l'innovazione dalla costruzione di grafi aciclici centrati sui nodi della rete. Viene dimostrato in particolare che la determinazione del set di utenti che adottano l'innovazione in un grafo aciclico può essere determinato con tempo lineare rispetto alla dimensione della rete. Quindi da questo concetto viene sviluppato un approccio che permette di stimare la diffusione dell'innovazione su tutta la rete dalla conoscenza delle interazioni locali che si sviluppano nei grafi aciclici. Dal problema classico della massimizzazione dell'influenza derivano delle estensioni, un esempio importante è dato dal recente lavoro proposto in [23]. Il fattore fondamentale che distingue il problema classico da queste estensioni è il concetto dell'orizzonte temporale. Lo scopo che si prefigge il problema classico è quello di determinare quali utenti adottano l'innovazione al termine del processo di diffusione. In realtà dal punto di vista pratico è più rilevante determinare il set di innovatori che massimizza la diffusione dell'innovazione in un orizzonte temporale limitato. Consideriamo un'azienda che vuole promuovere il proprio prodotto, lo scopo di questa non è soltanto quello di massimizzare la diffusione del prodotto a lungo termine ma potrebbe voler battere sul tempo la concorrenza fidelizzando quanti più consumatori possibili nel periodo di lancio. Questo è lo scopo principale di questa estensione. Un altro problema pratico è quello di diffondere l'innovazione su un target prestabilito di utenti, in particolar modo l'estensione si prefigge di determinare il set di innovatori minimo che garantisce la diffusione sul set obbiettivo in un orizzonte temporale limitato. Entrambe le estensioni vengono modellate come dei problemi di ottimizzazione discreta e per loro natura sono computazionalmente onerose. Il lavoro di tesi ruoterà sull'analisi di queste estensioni e i possibili approcci per renderle più trattabili, in particolare si cercherà di adattare l'approccio proposto in [11] al problema 1. Infine per quanto concerne l'implementazione degli algoritmi in linguaggio OPL sono stati consultati i manuali [16, 17, 18, 19] disponibili nel sito della IBM.

1.3 Struttura della tesi

La tesi è strutturata nel modo seguente:

- **Capitolo 2:** In questo capitolo verranno trattati i concetti sul quale si fonda tutto il lavoro di tesi. Particolare enfasi verrà posta nella definizione dei principali modelli di diffusione dell'innovazione proposti in letteratura e dei modelli di rete. Verrà quindi analizzato in maniera puntuale il problema classico della massimizzazione dell'influenza.

- **Capitolo 3:** Questo capitolo è dedicato ai richiami di programmazione lineare. Nella prima parte vengono trattati i concetti principali della programmazione lineare continua quali: approccio geometrico, metodo del simplesso classico e proprietà di dualità. Nella seconda parte viene trattato il tema dell'ottimizzazione discreta con particolare enfasi sui metodi risolutivi.

- **Capitolo 4:** Nel presente capitolo viene trattato il tema della diffusione dell'innovazione nei social network. Nella prima parte sia analizza il concetto di set coesivo massimale e verrà analizzato l'approccio risolutivo proposto in [23]. Nella seconda parte vengono mostrati due problemi di ottimizzazione discreti inerenti le estensioni del problema di massimizzazione dell'influenza proposte in [23].

- **Capitolo 5:** In questo capitolo vengono trattati in modo formale gli approcci euristici proposti in questa tesi. Nella prima parte verrà analizzata l'euristica basata su un algoritmo greedy proposto in [11]. Da questa euristica proporremo un piccola estensione per rendere trattabile il problema di massimizzazione dell'influenza in tempo finito. Nella seconda parte viene mostrato un metodo per ridurre la dimensione della rete dedicato alla risoluzione del problema della diffussione dell'innovazione su un set di utenti target.

- **Capitolo 6:** In questo capitolo vengono mostrati gli approcci implementativi per la risoluzione dei problemi citati nei precedenti capitoli. Nella prima parte viene descritta l'euristica per il problema di massimizzazione dell'influenza in tempo finito. Nella seconda parte vedremo l'implementazione dei tre problemi in linguaggio OPL, e viene mostrato con un semplice esempio il metodo di riduzione della rete sviluppato per il secondo problema.

- **Capitolo 7:** In questo capitolo verranno mostrati i risultati delle simulazioni con particolare enfasi sui tempi di calcolo e sulla precisione degli approcci approssimati.

- **Capitolo 8:** In questo capitolo vengono discussi brevemente i risultati ottenuti e vengono proposti degli sviluppi futuri.

- **Appendice A:** Questa appendice è dedicata al linguaggio OPL e al risolutore CPLEX.
- **Appendice B:** In questa appendice sono riportati tutti i codici scritti in ambiente Matlab.
- **Appendice C:** In questa appendice sono riportati tutti i codici scritti in linguaggio OPL.

1.4 Contributi personali

Nella parte iniziale del lavoro di tesi sono stati implementati in linguaggio OPL i problemi di ottimizzazione proposti in [23] che sono stati indicati nella sezione degli obbiettivi. Succesivamente sono stati sviluppati degli approcci di ottimizzazione per renderli più trattabili. Per il problema 1 sono stati sviluppati:

- Un'euristica implementata in linguaggio Matlab basata sull'approccio euristico proposto in [11] per risolvere il problema classico della massimizzazione dell'influenza.
- Un'approccio approssimato basato sulla taratura del risolutore CPLEX. In particolare vengono modificati i parametri che regolano l'algoritmo branch and cut del risolutore in modo da ottenere una soluzione subottima accettabile diminuendo il carico computazionale.

Per il problema 2 è stato sviluppato: Un algoritmo implementato in Matlab per la riduzione della dimensione della rete basata sullo studio del problema di ottimizzazione. L'algoritmo scarta i nodi della rete che non sono in grado di raggiungere i nodi target nell'orizzonte temporale prefissato. I dati vengono passati all'ottimizzatore che risolve il problema 2 su un set di dati più piccolo ottenendo la soluzione che si otterrebbe con la rete di partenza. L'algoritmo fornisce anche una rappresentazione grafica della rete ottenuta in seguito alla riduzione della rete di partenza.

Capitolo 2

Modelli di diffusione dell'innovazione

2.1 Introduzione

Per decenni gli esperti di sociologia si sono interessati allo studio dei fenomeni di diffusione delle mode, dei comportamenti e delle innovazioni nei social networks. In letteratura sono stati proposti numerosi modelli atti a caratterizzare fenomeni di diffusione che ricoprono svariati campi applicativi. Si parte dalla propagazione dell'informazione nei blogs passando alla diffusione di nuove pratiche in campo di agricoltura e medicina fino ad arrivare all'analisi dei fenomeni di propagazione dell'obesità all'interno della popolazione. La diffusione su vasta scala di Internet ha reso possibile lo studio di reti di elevate dimensioni che in molti casi possono raggiungere centinaia di milioni di utenti. Quindi ben presto internet è divenuto un terreno fertile per studiare delle efficaci strategie di marketing. In questo capitolo vedremo in particolare i principali modelli di diffusione dell'innovazione proposti in letteratura, vedremo poi il problema classico di massimizzazione dell'influenza nei social networks.

2.2 Modelli di diffusione

I primi studi empirici riguardo alla diffusione nei social network risalgono alla metà del 20 secolo. Qualche decennio più tardi sono stati proposti i primi modelli matematici da Granovetter [1] e Schelling [2]. Come detto in precedenza esistono numerosi modelli proposti che però hanno in comune la definizione di rete. Formalmente un *social network* viene definito tramite un grafo $\mathcal{G}(V, E)$, ogni vertice $v \in V$ del grafo rappresenta un utente della rete. Considerando dei grafi diretti un arco $(u, v) \in E$ rappresenta l'influenza del nodo u sul nodo v. Possiamo infine indicare con $\mathcal{N}(v)$ l'insieme di individui che hanno influenza diretta sul nodo v. Formalmente possiamo definire

l'insieme di nodi vicini come $\mathcal{N}(v) = \{u \in V : (u, v) \in E\}$. Nelle successive sezioni vedremo i principali modelli di diffusione nei social networks.

2.2.1 Interazioni locali

Consideriamo due piloti che gareggiano in una competizione automobilistica. Entrambi dovranno deviare la loro traiettoria per evitare di collidere col proprio avversario. Se entrambi girano contemporaneamente a destra o a sinistra eviteranno la collisione mentre se uno dei due contendenti opta per la manovra opposta rispetto all'altro la collisione risulterà inevitabile. Questo scenario è un esempio estremo di coordinazione in un gioco o competizione. In questo tipo di problemi di coordinazione sono presenti due contendenti il cui interesse comune è prendere la medesima decisione. Se un contendente prende la decisione A, questa è la migliore anche per l'avversario.

	A	**B**
A	1-q,1-q	0,0
B	0,0	q,q

Tabella 2.1: Semplice esempio di problema di coordinazione

Tramite la semplice tabella 2.1 è possibile riassumere il problema di coordinazione, ogni riga rappresenta un azione del giocatore 1 mentre ogni colonna rappresenta l'azione dell'altro giocatore. I due numeri che compaiono in ciascuna casella della matrice corrispondono rispettivamente al guadagno rispetto all'azione scelta dal giocatore 1 e dal giocatore 2, si vede infatti dai valori in antidiagonale come sia svantaggioso scegliere l'azione opposta rispetto all'avversario. Il parametro q infine rappresenta la bontà dell'azione A rispetto alla B.

Prendiamo ora in considerazione un esempio più realistico supponiamo che due amici vogliano comunicare attraverso un software di instant messaging e che ve ne siano due di pari caratteristiche ma tra loro incompatibili. Se entrambi scelgono di utilizzare AOL messenger possono comunicare con successo, analogamente se entrambi sceglieressero di utilizzare Yahoo! messenger. Se invece uno dei due scegliesse un software diverso rispetto all'altro i due amici non potrebbero comunicare a causa dell'incompatibilità tra le due tecnologie software. Lo scenario diventa ancora più interessante se consideriamo che i due amici possono far parte di una rete più grande nel quale ogni utente vuole comunicare con i suoi vicini. È chiaro che ciascun individuo se vuole comunicare con i suoi vicini deve optare per il software utilizzato da essi. L'esempio appena fatto è un esempio di interazione locale, ossia un

estensione del problema di coordinazione tra due giocatori. Un esempio di formalizzazione del problema di interazione locale si deve a Morris [5].

2.2.2 Modello a cascata

Il modello a cascata è stato studiato per la prima volta nel contesto del marketing da Goldenberg *et al.*[6]. Ogni individuo è caratterizzato da una certa probabilità di attivare o convincere i propri vicini una volta che diviene attivo. Un semplice esempio è dato dal modello *Indipendent cascade model* nel quale la probabilità di attivazione di un individuo in seguito all'attivazione dei nodi vicini è indipendente rispetto al set di utenti attivati in passato che hanno cercato di attivarlo. Quindi partendo da un set iniziale di nodi attivi il processo di attivazione di altri nodi può essere studiato ad intervalli regolari. Al tempo t ogni nodo v attivo ha una possibilità di attivare i nodi vicini u per ogni $v \in \mathcal{N}(u)$ ossia ogni nodo v ha una possibilità di attivare i nodi u sul quale esercita una certa influenza. Di conseguenza un certo nodo u può divenire attivo al tempo $t+1$ con probabilità $p_{u,v}$. È comunque possibile generalizzare questo modello considerando, anziché la probabilità $p_{u,v}$, la probabilità che il nodo v attivi il nodo u in maniera dipendente dal set di nodi che in passato hanno cercato di attivare u. Possiamo formalmente definire la probabilità $p_{u,v}(S)$ per ogni insieme $S \in \mathcal{N}(u)\backslash\{v\}$. Quindi in primo luogo un nodo v diviene attivo e può attivare a sua volta un nodo u con probabilità $p_{u,v}(S)$ dove S rappresenta il set di nodi vicini a u che in passato hanno cercato di attivarlo senza successo. Per garantire questo meccanismo, definiamo il modello nel seguente modo: Consideriamo solo il set di probabilità $p_{u,v}(S)$ tali per cui la probabilità che un nodo u diventi attivo in seguito all'influenza di un set U risulti indipendente dall'ordine col quale i membri del set tentano di attivarlo. Senza questa proprietà non è ben chiaro cosa succede se più di un vicino si attiva contemporaneamente.

2.3 Modello a soglia

Il problema di interazione locale può essere visto come una semplice sottoclasse del modello più generale a soglia. Ricordiamo che nell'interazione locale, un nodo v sceglie la sua azione al tempo t in base al comportamento adottato dal suo avversario all'istante di tempo precedente. Se v sceglie l'azione A, il suo guadagno sarà $1 - q$ volte il numero di vicini che hanno optato per la soluzione A. Mentre se scegliesse la soluzione B il suo guadagno sarebbe pari a q volte il numero di vicini che hanno optato per questa soluzione. Di conseguenza A è la migliore scelta per il nodo v se e solo se una frazione q o maggiore dei suoi vicini scelgono questa azione. Formalmente il

nodo v sceglie la soluzione A se vale la seguente condizione:

$$\frac{1}{|\mathcal{N}(v)|}\sum_{u\in\mathcal{N}(v)} X_{u,t-1} \geq q$$

dove $X_{u,t-1}$ vale 1 se u sceglie l'azione A al tempo $t-1$ mentre viceversa vale 0. Il parametro q può essere visto come una soglia di decisione. Il modello a soglia risale a Granovetter [1] e Schelling [2]. In particolare il modello più semplice è il cosidetto *modello a soglia lineare* . In questo modello ogni nodo $v \in V$ è caratterizzato da un peso non negativo $\omega_{v,u}$ per ogni nodo $u \in \mathcal{N}(v)$, per il quale risulta $\sum_{u\in\mathcal{N}(v)} \omega_{v,u} \leq 1$ e da una soglia θ_v. Date le soglie e un set iniziale $\mathcal{A}_1$ di nodi attivi il processo di attivazione dei nodi si dipana in maniera deterministica in una sequenza di passi. Al tempo t ogni nodo che risultava attivo al passo precedente rimane attivo. Ogni nodo v che al tempo $t-1$ era inattivo si attiverà al tempo t se vale la seguente relazione:

$$\sum_{u\in\mathcal{N}(v)} \omega_{v,u} X_{u,t-1} \geq \theta_v$$

Dove $X_{u,t-1}$ vale 1 se u era attivo al tempo $(t-1)$, 0 altrimenti. Intuitivamente il peso $\omega(v,u)$ rappresenta l'influenza esercitata dal nodo u sul nodo v mentre θ_v rappresenta la personale tendenza del nodo v ad adottare la nuova tecnologia o innovazione. Il modello a soglia lineare viene definito *progressivo* perché quando un nodo v opta per un certo comportamento o azione continuerà a mantenerla anche nei passi successivi.

2.3.1 Modello a soglia lineare stocastico

Il modello a soglia lineare è in grado di caratterizzare in maniera efficace l'influenza interpersonale nell'adozione di un comportamento all'interno della rete. Questo modello però non tiene conto di un importante aspetto della diffusione dell'innovazione la dipendenza dal percorso. L'idea di base consiste nel fatto che degli eventi all'interno di un percorso della rete siano in grado di alterare il corso della storia. Per tener conto di questo fattore Yildiz *et al.*[7] hanno proposto il modello a soglia lineare stocastico. Il modello a soglia lineare viene modificato nel seguente modo: indichiamo con $x_i(k)$ lo stato di un certo nodo i al passo k, questo può assumere i seguenti valori $\{0, 1, -1\}$ che indicano rispettivamente che il nodo i non adotta ancora l'innovazione, la adotta oppure rifiuta l'adozione. Assumiamo che all'iterazione $k = 0$ un set $\hat{\phi}(0) \subseteq \mathcal{V}$ venga selezionato come insieme di innovatori di partenza. Al passo successivo un individuo $i \in \mathcal{V}\backslash\hat{\phi}(0)$ considera attivamente di adottare l'innovazione se al più una frazione $\phi_i \in (0, 1]$ dei suoi vicini appartengono al set di innovatori di partenza, per esempio:

$$\frac{|\hat{\phi}(0)\bigcap\mathcal{N}_i(\mathcal{G})|}{|\mathcal{N}_i(\mathcal{G})|} \geq \phi_i \rightarrow i \in \hat{\phi}(1)$$

In altre parole il set $\hat{\phi}(1)$ è composto dagli individui che subiscono un esposizione all'innovazione e che sono persuasi dai propri vicini sulla bontà dell'adozione dell'innovazione. L'esito del processo di decisione viene modellato attraverso una distribuzione di Bernoulli con parametro $p \in [0, 1]$. Il parametro p è specifico per ogni individuo, al contrario della distribuzione. In altre parole per ogni $i \in \hat{\phi}(1)$, $x_i(1) = 1$ con probabilità p mentre $x_i(1) = -1$ con probabilità $1 - p$. Quindi il parametro p determina la probabilità di attivazione quando l'utente considera attivamente di adottare l'innovazione. Il set di individui che adottano o rifiutano l'innovazione al passo $k = 1$ è formato da $\mathcal{A}(1)(\mathcal{R}(1))$, ossia $\mathcal{A}(1) = \{i \in \mathcal{V} : i \in \hat{\phi}(1), x_i(1) = 1\}$, $\mathcal{R}(1) = \{i \in \mathcal{V} : i \in \hat{\phi}(1), x_i(1) = -1\}$. Quindi per $k \geq 0$ possiamo generalizzare nel modo seguente: Un nodo $i \in \mathcal{V} \setminus \bigcup_{l=0}^{k-l} \hat{\Phi}(l)$ considera attivamente di adottare l'innovazione al passo k se:

$$\frac{|\{\hat{\Phi}(0) \cup \bigcup_{l=1}^{k-1} \mathcal{A}(l)\} \bigcap \mathcal{N}_i(\mathcal{G})|}{|\mathcal{N}_i(\mathcal{G})|} \geq \phi_i \rightarrow i \in \hat{\Phi}(k)$$

e il nodo adotterà o rifiuterà l'innovazione al passo k in accordo con la distribuzione di Bernoulli con parametro p. La differenza tra il modello stocastico e quello classico deriva dal fatto che un individuo non adotta necessariamente l'innovazione se una frazione di individui suoi vicini che l'hanno adottata supera la sua soglia.

2.4 Massimizzazione dell'influenza

Il problema classico della massimizzazione dell'influenza consiste nell'identificare il set di utenti maggiormente influenti che permettano la maggiore diffusione dell'innovazione o prodotto. Questo problema è stato introdotto per la prima volta da Domingos e Richardson [9] che notarono come le tecniche di *data mining*[1] applicate con successo in varie applicazioni di *direct marketing*[2] non prendessero in considerazione molti aspetti importanti delle dinamiche di rete. Infatti nel *direct marketing* tipicamente si decide di fare pubblicità ad un certo target di utenti in base alle proprie caratteristiche quindi tramite la conoscenza dell'utenza. Questo aspetto tralascia completamente la possibilità di sfruttare le relazioni interpersonali come potente

[1] Il data mining è l'insieme di tecniche e metodologie che hanno per oggetto l'estrazione di un sapere o di una conoscenza a partire da grandi quantità di dati (attraverso metodi automatici o semi-automatici) e l'utilizzo scientifico, industriale o operativo di questo sapere.

[2] Il direct marketing è una tecnica di marketing attraverso la quale aziende e enti (esempio organizzazioni pubbliche e no profit) comunicano direttamente con clienti e utenti finali. Il direct marketing consente di raggiungere un target definito, con azioni mirate che utilizzano una serie di strumenti interattivi, ottenendo in tal modo delle risposte misurabili.

strumento di diffusione. Questi aspetti vengono sfruttati efficacemente dalle tecniche del *viral marketing*[3], un esempio lampante è dato dal servizio Hotmail che ha ottenuto 12 milioni di registrazioni in soli 18 mesi con un budget limitato. Questo risultato è stato ottenuto includendo un messaggio promozionale su ogni email inviata dagli utenti già iscritti. Il problema della massimizzazione dell'influenza può essere formalizzato in vario modo. Domingos e Richardson [9],[10] modellarono il problema attraverso una *rete di Markov*, proposero poi un'euristica mirata a massimizzare la differenza tra il profitto ottenuto attraverso la strategia di marketing e quello ottenuto senza attuare alcuna politica di pubblicizzazione. Questo approccio viene spesso utilizzato nel contesto del marketing diretto. Nel recente lavoro proposto in [8] viene definito un diverso approccio, si assume che la compagnia o azienda che vuole promuovere il proprio prodotto disponga di un budget sufficiente a convincere un set di k utenti. Il problema è quindi quello di trovare quale insieme di utenti massimizza la diffusione dell'innovazione. Formalmente Kempe *et al.*[8] definiscono *l'influenza* di un certo set $\mathcal{A}$ denotandola $\sigma(\mathcal{A})$ come il valore atteso di utenti che alla fine del processo di diffusione adottano l'innovazione. Viene quindi mostrato come impiegando vari modelli di diffusione quali quello a soglia lineare e a cascata indipendente il problema della massimizzazione dell'influenza sia di tipo NP hard ossia il tempo computazionale non cresce linearmente con la dimensione dei dati in ingresso. Per rendere il problema trattabile vengono spesso utilizzate delle euristiche basate su *algoritmi greedy* che in sostanza sfruttano delle proprietà del modello o del problema. Vedremo questi concetti con maggiore dettaglio nei capitoli successivi.

2.5 Set coesivo massimale

Finora abbiamo visto i principali modelli di diffusione dell'innovazione e abbiamo definito il problema classico della massimizzazione dell'influenza. Nasce ora la necessità di stabilire quanti utenti adottano l'innovazione alla fine del processo di diffusione. Nel recente lavoro di Yildiz *et al.* [7] viene proposto un metodo di determinazione dello stato finale della rete basato sul concetto di *set coesivo*. Dal punto di vista informale un set viene definito coesivo se nessun membro adotta l'innovazione a partire da un set di innovatori $\mathcal{A}$ non appartenenti al set coesivo stesso. in questo particolare approccio viene utilizzato il modello a soglia lineare. L'algoritmo proposto si basa sulla determinazione del set coesivo massimale complemento del set

[3]Il marketing virale è un tipo di marketing non convenzionale che sfrutta la capacità comunicativa di pochi soggetti interessati per trasmettere il messaggio ad un numero elevato di utenti finali. La modalità di diffusione del messaggio segue un profilo tipico che presenta un andamento esponenziale. È un'evoluzione del passaparola, ma se ne distingue per il fatto di avere un'intenzione volontaria da parte dei promotori della campagna.

di innovatori di partenza, in base alla conoscenza di questo insieme è possibile determinare l'insieme di utenti che adottano l'innovazione attraverso la seguente relazione:

$$\Phi^* = \mathcal{V} - \mathcal{M}$$

Dove $\mathcal{V}$ è l'insieme dei nodi della rete, mentre $\mathcal{M}$ rappresenta l'insieme coesivo massimale complemento degli innovatori di partenza e Φ^* indica l'insieme di utenti che alla fine del processo adottano l'innovazione.

2.5.1 Diffusione dell'innovazione e struttura della rete

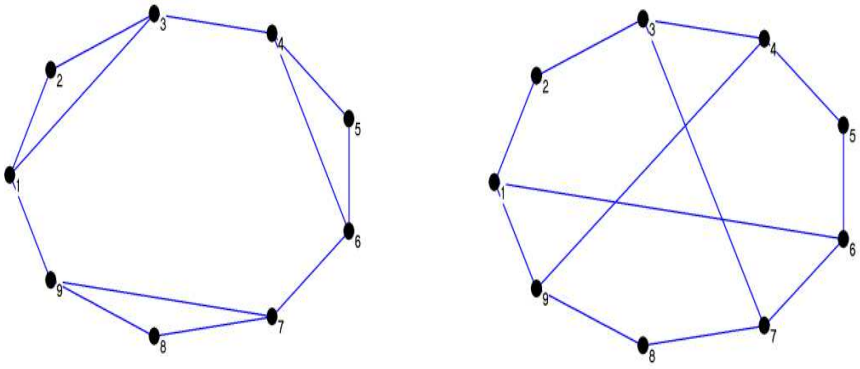
Finora abbiamo visto che noto il set coesivo massimale è possibile prevedere la configurazione finale di utenti che adottano l'innovazione. Nel lavoro di D. Acemoglu *et al.*[7] viene inoltre mostrato intuitivamente come il processo di diffusione possa dipendere in qualche modo dalla struttura della rete in termini topologici e rispetto alle soglie di decisione di ciascun utente. I set coesivi non sono necessariamente unici dato un grafo e date le soglie.

I fattori che influenzano la diffusione dell'innovazione sono sostanzialmente due:

- Valore delle soglie
- Struttura di rete

Valore delle soglie Per capire il contributo sulla diffusione dell'innovazione da parte delle soglie si può pensare di fissare la topologia di rete e far variare le soglie. All'aumentare delle soglie i nodi della rete costituiscono set coesivi con un numero via via inferiore di utenti. Questo processo porta a configurare la rete con set coesivi di piccole dimensioni accrescendo la difficoltà di diffondere l'innovazione. La diffusione dell'innovazione oltre che dalle soglie, viene regolata dalla topologia della rete.

Struttura di rete Per capire come la struttura della rete intervenga nel processo di diffusione prenderemo in considerazione il concetto di *clustering*. Consideriamo l'esempio in figura 2.1. La rete a sinistra ha un coefficiente di clustering maggiore rispetto a quella a destra. Il coefficiente di clustering si ottiene dal rapporto tra il numero di triangoli presenti nella rete e quelli possibili, di conseguenza la rete a sinistra ha un coefficiente di clustering pari a 0.33 dato dal rapporto dei tre triangoli presenti sui nove possibili. La rete a destra invece ha un coefficiente di clustering nullo dato che non contiene alcun triangolo. Entrambe le reti sono caratterizzate dallo stesso numero di nodi e archi e il valore delle soglie per tutti gli utenti in entrambe le reti è pari a 0.5. La rete con coefficiente di clustering più elevato è composta dai seguenti set coesivi:$\{\{1,2,3\},\{4,5,6\},\{7,8,9\}\}$, mentre l'altra è composta



(a) Coefficiente di clustering 0.33 (b) Coefficiente di clustering 0

Figura 2.1: Reti con diverso coefficiente di clustering

dai seguenti set: $\{\{1, 2, 7, 8, 9\}, \{3, 4, 5, 6\}\}$. Da questa informazione è possibile definire un legame tra coefficiente di clustering e set coesivi, possiamo infatti affermare che la rete con coefficiente pari a 0.33 è più strutturata ed è costituita da piccoli set coesivi al contrario dell'altra rete che è composta da soli due set coesivi di maggiori dimensioni. Nel lavoro proposto in [7] si afferma che una rete con coefficiente di clustering più alto è caratterizzata da set coesivi di piccole dimensioni che non consentono di diffondere agevolmente l'innovazione come una caratterizzata da un basso coefficiente di clustering.

2.6 Altri modelli e approcci euristici

Finora sono stati trattati i modelli di diffusione a cascata e a soglia lineare, esistono però altri modelli e approcci euristici votati alla risoluzione del problema classico della massimizzazione dell'influenza come mostrato in [25]:

1. *High-degree heuristic*: L'euristica seleziona gli innovatori di partenza in accordo con un indice d_v che identifica il grado di influenza di ciascun utente. La strategia è molto semplice ma al tempo stesso rappresenta un meccanismo naturale secondo il quale gli utenti con un maggior numero di relazioni impone più facilmente la sua influenza. In sociologia questo approccio viene definito "centralità" del grado.

2. *Low-distance heuristic:* Un altro modello comune è quello che in sociologia viene definito "centralità della distanza". La strategia si basa in sostanza sull'intuizione secondo cui gli individui sono maggiormente influenzati da quelli col quale hanno una relazione più stretta che in questo caso viene modellata dalla distanza.

3. *Degree discount Heuristic:* L'idea generale di questo approccio è che se un nodo u è stato selezionato come nodo seme, poi quando si considera la selezione del nodo v come un nuovo innovatore in base il suo grado, allora non dovremmo considerare l'arco dal nodo v al nodo u col relativo grado. In maniera più specifica per un nodo v col suo grado d_v e l'insieme di suoi vicini t_v appartenenti al set di innovatori di partenza, allora consideriamo il suo grado in relazione ai tali nodi vicini secondo la relazione: $2t_v(d_v - t_v)t_v p$.

Nella sezione 5.1.1 vedremo un altro approccio euristico per la risoluzione del problema della massimizzazione in tempo finito che abbiamo illustrato brevemente nel capitolo introduttivo. Questo si basa sulle specifiche caratteristiche del modello a soglia lineare che sarà il modello di diffusione che verrà trattato in questa tesi.

Capitolo 3

Richiami di programmazione lineare

3.1 Introduzione

Il primo metodo risolutivo per problemi di programmazione lineare vennè ideato nel 1947 da George Dantzig che sviluppò l'algoritmo del simplesso. Il metodo del simplesso deve il suo successo alla sua effettiva efficienza congiuntamente alla contingente nascita dei calcolatori elettronici che ne hanno permesso l'effettivo impiego. I moderni solutori di problemi di programmazione impiegano l'algoritmo del simplesso nella sua forma standard per la risoluzione di problemi continui e in svariate estensioni nell'ambito della risoluzione di problemi più complessi come i problemi interi. In questo capitolo vedremo i concetti basilari di programmazione lineare continua tratto principalmente dalle dispense [14] e faremo alcuni richiami alla programmazione intera tratti da [15].

3.2 Forma canonica e forma standard

Indichiamo con $c = (c_1, c_2, ..., c_n)^T$ il *vettore dei costi* e con $x = (x_1, x_2, ..., x_n)^T$ il *vettore di decisione*. La funzione obbiettivo viene indicata nella seguente forma vettoriale:

$$z = c^T x$$

Introduciamo inoltre la matrice $m \times n$:

$$\mathcal{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

e il *vettore dei termini noti* $b = (b_1, b_2, ..., b_m)^T$. Un problema di programmazione lineare si dice in forma canonica quando i vincoli sono di diseguaglianza e tutte la variabili sono non negative:

$$\begin{cases} \textit{Minimizza} & c \cdot x \\ \textit{tale che} & \mathcal{A} \cdot x = (\leq \textit{or} \geq) 0 \\ & x \geq 0 \end{cases}$$

Un problema di programmazione lineare è in forma standard quando tutti i vincoli sono in forma di uguaglianza e tutte la variabili risultano essere non negative:

$$\begin{cases} \textit{Massimizza} & c \cdot x \\ \textit{tale che} & \mathcal{A} \cdot x = b \\ & x \geq 0 \end{cases}$$

Nel caso in cui si debba massimizzare la funzione obbiettivo è sufficiente cambiare il segno al vettore dei costi per riportarsi alla forma standard. Qualora invece si debba risolvere un problema nella seguente forma:

$$\sum_{j=1}^{n} a_{ij} x_{ij} \leq b_i$$

è necessario introdurre una variabile aggiuntiva detta *slack variable* che permette di ricondurci in forma standard:

$$\sum_{j=1}^{n} a_{ij} x_{ij} + s_i = b_i \qquad s_i \geq 0$$

Se il problema è nella forma forma seguente:

$$\sum_{j=1}^{n} a_{ij} x_{ij} \geq b_i$$

analogamente a quanto fatto nel caso precedente è possibile ricondursi alla forma standard aggiungendo un variabile detta di *surplus*:

$$\sum_{j=1}^{n} a_{ij} x_{ij} - s_i = b_i \qquad s_i \geq 0$$

3.3 Interpretazione geometrica

Nel caso in cui il vettore di decisione x sia composto da due sole componenti è possibile risolvere il problema di programmazione attraverso un procedimento di tipo grafico. La retta che contraddistingue la funzione obbiettivo di equazione $z = c_1 x_1 + c_2 x_2$ si muoverà all'interno della regione delle soluzioni ammissibili nella direzione del *gradiente* $(c_1, c_2)^T$.

$$\nabla_z = \begin{bmatrix} \frac{\partial z}{\partial x_1} \\ \\ \frac{\partial z}{\partial x_1} \end{bmatrix} = c$$

Se il problema che si sta risolvendo è una minimizzazione allora z si muoverà in direzione opposta al gradiente viceversa se si risolve una massimizzazione la retta seguirà la direzione del gradiente. In figura 3.1 notiamo come la soluzione ottima sia posizionata in un dei vertici del poliedro che descrive la regione della soluzioni ammissibili.

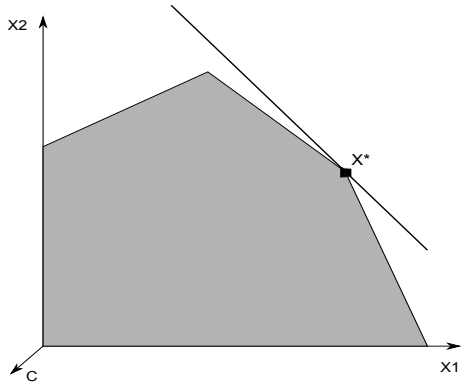
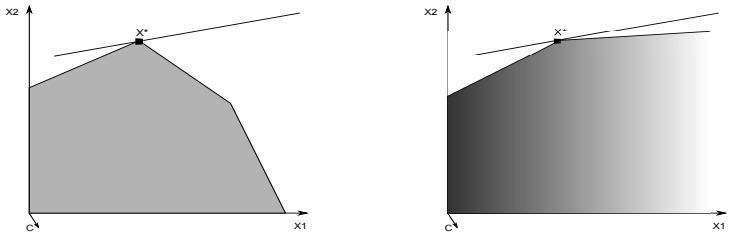


Figura 3.1: Interpretazione grafica della regione ammissibile

Nell'esempio la soluzione trovata era l'unica possibile ma si possono verificare diversi casi:

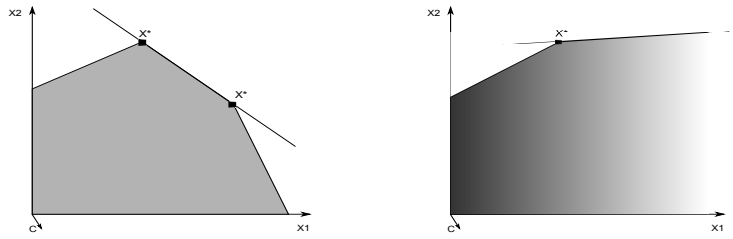
- **Soluzione ottima unica e finita:** In figura 3.2(a) si vede chiaramente come la retta che contraddistingue la funzione obbiettivo sia posizionata in uno dei vertici del poliedro e che quest'ultimo descriva un'area limitata. La figura a destra pone invece in evidenza come la soluzione sia unica ma con una regione della soluzioni ammissibili non limitata.



(a) limitato (b) Spazio illimitato

Figura 3.2: Soluzione ottima e spazio delle soluzioni illimitato

- **Soluzioni ottime multiple:** In figura 3.3(a) viene mostrato un esempio nel quale la retta che rappresenta la funzione obbiettivo collima con un lato del poliedro che descrive la regione ammissibile. Questo fa si che esistano due soluzioni ottime.



(a) limitato (b) Spazio illimitato

Figura 3.3: Soluzione ottima multipla

- **Regione ammissibile vuota:** Nessuno punto dello spazio fa parte della regione ammissibile, in tal caso i vincoli vengono detti *inconsistenti.*
- **Soluzione ottima e regione ammissibile illimitate:** In tal caso come mostrato in figura 3.4 sia la soluzione che lo spazio delle soluzioni ammissibili risulta essere illimitata.

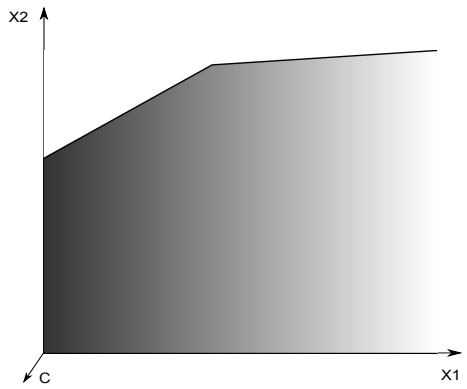


Figura 3.4: Soluzione ottima illimitata

3.4 Metodo del simplesso classico

Consideriamo un problema di programmazione lineare in forma standard:

$$\begin{cases} \textit{Massimizza} & c \cdot x \\ \textit{tale che} & \mathcal{A} \cdot x = b \\ & x \geq 0 \end{cases}$$

Senza alcuna perdita di generalità possiamo considerare che la matrice $\mathcal{A}$ abbia dimensioni (m, n) con $m < n$ dato che comunque un sistema di equazioni lineari ammette in tal caso ∞^{n-m} soluzioni inoltre rappresenta certamente il caso più significativo nella programmazione lineare. Ipotizziamo che la matrice $\mathcal{A}$ sia a rango pieno, attraverso opportune permutazioni è possibile portare la matrice nella forma seguente:

$$\mathcal{A} = [B \mid N]$$

Dove la matrice B contiene le m colonne linearmente indipendenti dalla matrice $\mathcal{A}$ mentre la matrice N contiene le restanti colonne della matrice $\mathcal{A}$. Analogamente possiamo scomporre il vettore delle incognite x e quello dei costi c ottenendo:

$$x^T = [x_B^T \mid x_N^T]$$

$$c^T = [c_B^T \mid c_N^T]$$

Possiamo quindi porre il problema di programmazione lineare standard nella seguente forma:

$$\begin{cases} min\{z = c^T x = c_B^T x_B + c_N^T x_N\} \\ \mathcal{A}x = Bx_B + Nx_N = b \\ x \geq 0 \end{cases} \tag{3.1}$$

La soluzione $x^T = x_B$ viene detta *soluzione di base ammissibile*. Questa si ottiene ponendo:

$$\begin{cases} x_B = B^{-1}b \\ x_N = 0 \end{cases}$$

La soluzione trovata deriva dal fatto che il problema di programmazione ha $(n-m)$ gradi di libertà e pertanto risulta possibile fissare ad arbitrio $(n-m)$ variabili x_N. Una soluzione x^* risulta ammissibile se rispetta il vincolo di non negatività, in particolare la soluzione di base è ammissibile se risulta $x_B \geq 0$ inoltre le x_N soddisfano sempre la condizione. A partire da una soluzione di base ammissibile è possibile determinare una nuova soluzione di base che porta ad una migliore funzione obbiettivo. Questo meccanismo è alla base del funzionamento dell'algoritmo del simplesso.

Criterio di ingresso e uscita delle variabili Il nostro proposito è quello di determinare una soluzione di base ammissibile che permetta di ottenere un valore della funzione obbiettivo inferiore al valore corrente. Indichiamo nel modo seguente il valore della funzione obbiettivo in corrispondenza della soluzione di base ammissibile di partenza:

$$z_0 = c_B^T B^{-1} b$$

La nuova soluzione di base ammissibile si ottiene scambiando opportunamente una colonna della matrice B con una della matrice N ciò comporterà un cambiamento del valore della funzione obbiettivo dato che sono state scambiate due variabili una appartenente all'insieme delle soluzioni di base ammissibili e una fuori da tale insieme. Vediamo in dettaglio quali variabili vengono coinvolte nello scambio. È possibile ricavare dal sistema 3.1 il valore delle variabili di base:

$$x_B = B^{-1}b - B^{-1}Nx_n = \hat{b} - \sum_{j\in\mathcal{R}} B^{-1}a_j x_j = \hat{b} - \sum_{j\in\mathcal{R}} y_j x_j$$

Dove $\hat{b}$ rappresenta il valore corrente del vettore di base mentre $\mathcal{R}$ rappresenta l'insieme di indici associati alle variabili non appartenenti alla base ammissibile, inoltre a_j e x_j rappresentano rispettivamente le colonne della matrice N e le componenti del vettore x_n. In corrispondenza di tale soluzione di base la funzione obbiettivo assume il seguente valore:

$$c^T x = c_B^T x_B + c_N^T x_N = z_0 - \sum_{j\in\mathcal{R}} (z_j - c_j)x_j$$

Dove:

$$c_B^T y_j = c_B^T B^{-1} a_j = \omega^T a_j \omega^T = c_B^T B^{-1}$$

Soltanto una delle variabili x_j passa da un valore nullo ad un valore positivo, ciò comporta che per minimizzare la funzione obbiettivo z è necessario scegliere un indice $j = k$ tale per cui:

$$z_k - c_k = \max_{j\in\mathcal{R}}(z_j - c_j)$$

Questo è il criterio con il quale si seleziona una variabile da includere nella base ammissibile. Analogamente dobbiamo selezionare una variabile che esce dalla base, il criterio di scelta è rappresentato dalla seguente relazione:

$$\frac{\hat{b_r}}{y_{rk}} = \min_{1\leq i\leq m}\left[\frac{\hat{b_i}}{y_{ik}}; y_{ik} \geq 0\right] = x_k$$

Quindi la variabile di base x_{br} di indice r entra a far parte delle variabili fuori base. Il coefficiente y_{kr} viene detto *pivot* e serve ad eseguire l'aggiornamento delle variabili di base in seguito all'accesso in base della variabile x_k. Tale procedimento iterativo viene eseguito finché esistono variabili fuori dalla base in grado di far decrescere il valore della funzione obbiettivo. Possiamo quindi riassumere l'algoritmo del simplesso nei seguenti passi:

1. Si inizializza il problema selezionando la soluzione di base ammissibile.
2. Si seleziona tramite il criterio di ingresso la variabile da portare in base. Nel caso in cui risultasse $z_k - c_k \leq 0$ allora la soluzione corrente è ottima e l'algoritmo si arresta, altrimenti si va al passo 3.

3. Tramite il criterio di uscita si seleziona la variabile da includere nell'insieme di variabili non di base. Se i coefficienti $y_{ik} \leq 0$ per $i = 1, ..., m$ allora non esiste ottimo finito. In caso contrario si va al passo 4.

4. Si aggiornano le variabili di base e si va al passo 2 dell'algoritmo.

I moderni risolutori impiegano una versione rivisitata del metodo del simplesso che permette una minore occupazione di memoria.

3.5 Programmazione lineare duale

Ad ogni problema di programmazione lineare è possibile associare un problema duale. L'insieme di questi due problemi definisce una teoria della dualità che è una parte molto importante in quanto permette la comprensione dei problemi di programmazione lineare, la definizione di nuovi algoritmi e l'interpretazione dei risultati forniti dall'algoritmo del simplesso. Per non confondere i due problemi d'ora in poi chiameremo *problema primale* il problema di partenza mentre chiameremo *problema duale* il problema ricavato da quello di partenza.

3.5.1 Forma canonica e standard del problema duale

Consideriamo un *problema primale* nella forma canonica:

$$\begin{cases} \textit{Minimizza} & c \cdot x \\ \textit{tale che} & \mathcal{A} \cdot x \geq b \\ & x \geq 0 \end{cases}$$

La forma duale ad esso associata ha la seguente forma:

$$\begin{cases} \textit{Massimizza} & b^T \cdot \omega \\ \textit{tale che} & \mathcal{A}_d \cdot \omega \leq c \\ & \omega \geq 0 \end{cases}$$

Tra i due problemi di programmazione sussistono le seguenti proprietà :

- Il numero di variabili del problema duale è pari al numero di vincoli del problema di partenza.
- Il vettore dei costi del problema primale costituiscono il vettore dei termini noti del problema duale.
- Tra la matrice dei vincoli del problema primale e quella del problema duale sussiste la seguente relazione:

$$\mathcal{A} = \mathcal{A}_d^T$$

Tra i due problemi sussiste una relazione biunivoca ossia uno è il primale dell'altro e viceversa. Data la forma standard del problema lineare:

$$\begin{cases} \textit{Massimizza} & c \cdot x \\ \textit{tale che} & \mathcal{A} \cdot x = b \\ & x \geq 0 \end{cases}$$

La sua forma duale ha la seguente forma:

$$\begin{cases} \textit{Massimizza} & b^T \cdot \omega \\ \textit{tale che} & \mathcal{A}_d \cdot \omega \leq c \\ & \forall \omega \end{cases}$$

Le stesse proprietà viste per la forma canonica valgono per la forma standard.

3.5.2 Relazioni tra la forma primale e duale

Tra la forma primale e duale sussitono delle importanti relazioni il primo è quello della dualità debole:

Teorema della dualità debole. *Per qualsiasi soluzione x_0 ammissibile di un problema di minimizzazione Il valore della funzione obiettivo è sempre maggiore o uguale al valore della funzione obiettivo per qualsiasi soluzione ω_0 ammissibile del duale di massimizzazione. Quindi risulta che:*

$$c \cdot x_0 \geq b^T \cdot \omega_0$$

•

Dal teorema della dualità debole discendono le seguenti proprietà:

- Per qualsiasi soluzione ammissibile di un problema di minimizzazione il valore della funzione obiettivo fornisce un limite superiore al valore ottimo della funzione obiettivo del problema duale.
- Per qualsiasi soluzione ammissibile di un problema di massimizzazione il valore della funzione obbiettivo fornisce un limite inferiore al valore ottimo della funzione obbiettivo del corrispettivo duale di minimizzazione.
- Il problema duale risulta inammissibile se il suo primale ha una soluzione illimitata. Tale relazione è ovviamente biunivoca ossia risulterà che il problema primale è inammissibile se il suo duale fornisce una soluzione illimitata.

- Se il problema primario ammette una soluzione x_0 e il suo duale ammette soluzione ω_0 e inoltre i due problemi hanno in corrispondenza delle rispettive soluzioni lo stesso valore per la funzione obbiettivo allora la soluzione x_0 e la soluzione ω_0 sono soluzioni ottime rispettivamente per il problema primale e duale.

La seconda relazione tra la forma primale e duale è data dal *teorema della dualità forte.*

Teorema della dualità forte. *Date le soluzioni ammissibili x_0 e ω_0 rispettivamente del problema primale e duale se vale che $c^T x_0 = b^T \omega_0$ allora le soluzioni sono ottime per i rispettivi problemi.*

3.6 Programmazione lineare intera

Finora abbiamo discusso le problematiche relative alla programmazione lineare su variabili continue. In molte applicazioni però le variabili del problema risultano essere soggette ad un ulteriore vincolo che è quello di interezza. Questo concetto risulta piuttosto chiaro tramite un esempio grafico. Consideriamo il seguente problema di ottimizzazione lineare discreto:

$$\begin{cases} \textit{Minimizza} & c \cdot x \\ \textit{tale che} & \mathcal{A} \cdot x = (\leq \textit{or} \geq) 0 \\ & x \in \mathbb{Z}_+^n \end{cases}$$

Dove $\mathbb{Z}_+^n$ denota l'insieme dei vettori n dimensionali a componenti intere non negative. In figura 3.5 è riportata la rappresentazione geometrica di un problema di ottimizzazione discreto, dove la regione ammissibile è data dai soli punti che cadono sugli incroci della quadrettatura.

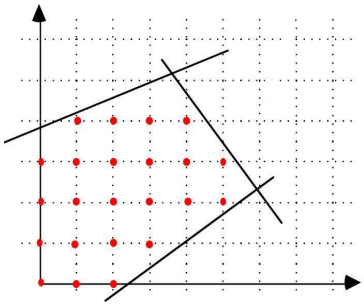


Figura 3.5: Interpretazione grafica della programmazione intera

Le tecniche di programmazione lineare non possono essere applicate direttamente a questa classe di problemi dato che si otterrebbero soluzioni nel quale le variabili possono assumere valore frazionario.

3.7 Scelta della formulazione e metodi di soluzione poliedrali

A differenza della programmazione lineare continua, nella programmazione lineare intera la scelta della formulazione ha una notevole influenza sull'efficienza dei metodi di soluzione e spesso, risulta essere decisiva. Prendiamo in considerazione il seguente esempio:

$$\begin{cases} \textit{Massimizza} & x_1 + 0.64x_2 \\ & 50x_1 + 31x_2 \geq 0 \\ & 3x_1 - 2x_2 \leq -4 \\ & x_1, x2 \in \mathbb{Z}_+^n \end{cases}$$

In figura 3.6 riportiamo la regione delle soluzioni ammissibili e il valore ottimo del problema ottenuto rilassando i vincoli di interezza del problema di partenza.

La soluzione del problema continuo chiaramente non ricade in nessuno

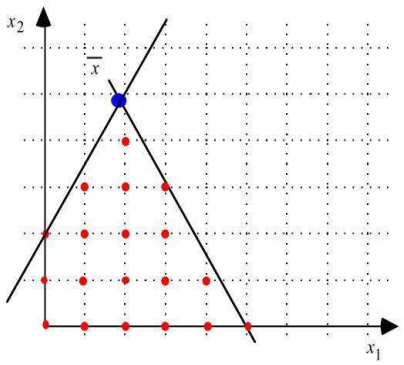


Figura 3.6: regione ammissibile e soluzione ottima del problema continuo

dei valori interi definiti dalla regione ammissibile. Si potrebbe ipotizzare di trovare una soluzione al problema approssimando i valori frazionari al loro intero più vicino. Essendo la soluzione del problema continuo $[x_1 = \frac{376}{193}, x_2 = \frac{950}{193}]$ la soluzione approssimata dovrebbe valere$[x_1 = 2, x_2 = 4]$. La soluzione ottima intera vale $[x_1 = 5, x_2 = 0]$ quindi l'approssimazione trovata risulta essere molto distante da quella ottima. Cosa sarebbe successo se, invece di utilizzare la formulazione con due vincoli data sopra, avessimo utilizzato la seguente:

$$\begin{cases} \textit{Massimizza} & x_1 + 0.64x_2 \\ & -x_1 + x_2 \leq 2 \\ & x_1 + x_2 \leq 6 \\ & 63x_1 + 2x_2 \leq 15 \\ & x_1, x_2 \in \mathbb{Z}_+^n \end{cases}$$

Possiamo notare immediatamente dalla figura 3.7 che dal punto di vista della regione ammissibile le due formulazioni sono del tutto equivalenti perché racchiudono lo stesso insieme di punti interi. La seconda formulazione però ha tutti i vertici in corrispondenza a soluzioni ammissibili per il problema di PLI. Ricordando che in un problema di programmazione lineare se l'ottimo esiste finito almeno un vertice del poliedro è ottimo, questo implica che se ignoriamo i vincoli di interezza e risolviamo il corrispondente problema di PL, la soluzione che otteniamo è intera e quindi òttima anche per il problema originale. Le considerazioni sviluppate nell'esempio precedente mettono in luce come sia importante individuare una buona formulazione del problema.

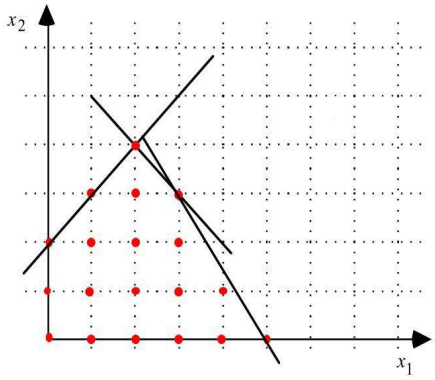


Figura 3.7: Riformulazione del problema di ottimizzazione discreto

3.8 Metodi poliedrali e dei piani di taglio

L'idea di risolvere un problema di ottimizzazione discreto attraverso metodi risolutivi della programmazione lineare mista è alla base dei cosidetti *metodi poliedrali*. Tali metodi talvolta definiti *metodi dei piani di taglio* non fanno altro che raffinare iterativamente il poliedro dove pensiamo che si trovi la soluzione ottima. Consideriamo il seguente problema di ottimizzazione discreta:

$$\begin{cases} \textit{Minimizza} & c \cdot x \\ \textit{tale che} & \mathcal{A} \cdot x \geq b \\ & x \in \mathbb{Z}_+^n \end{cases}$$

e il relativo problema in cui rilassiamo il vincolo di interezza:

$$\begin{cases} \textit{Minimizza} & c \cdot x \\ \textit{tale che} & \mathcal{A} \cdot x = \geq 0 \\ & x \geq 0 \end{cases}$$

Quest'ultimo è un problema di programmazione lineare che supponiamo ammetta soluzione ottima finita $\bar{x}$. Se $\bar{x}$ ha tutte componenti intere allora coincide con la soluzione ottima del problema intero, altrimenti
è possibile riformulare la versione rilassata aggiungendo opportuni vincoli.

Definizione 1: Una disuguaglianza valida è un vincolo $gx \geq \gamma$ tale che: $\forall x \in \{x \in \mathbb{Z}_+^n : Ax \geq b\}$

Definizione 2: un piano di taglio è una disuguaglianza $gx \geq \gamma$ valida se: $g\bar{x} \leq \gamma$

Pertanto, se individuiamo un piano di taglio per il problema rilassato e la sua soluzione ottima frazionaria $\bar{x}$, possiamo aggiungere tale disuguaglianza alla formulazione e iterare la procedura avvicinandoci progressivamente alla soluzione ottima intera. Il procedimento è riassunto dalla seguente procedura:

Procedura dei piani di taglio.

1. *ottimo = falso*
2. *Risolvi il problema rilassato*
3. *Se $\bar{x}$ ha tutte componenti intere*
4. *ottimo = vero vai al passo 8*
5. *Altrimenti*
6. *Aggiungi taglio*
7. *Vai al passo 2*
8. *Restituisci la soluzione ottima*

Il punto chiave risulta quindi trovare un metodo che ci permetta di generare dei piani di taglio il più efficaci possibile, cioè vicini al poliedro dell'inviluppo convesso. Un metodo utile per ricavare delle disuguaglianze valide è quello di Chvátal, che però non genera necessariamente piani di taglio per la soluzione frazionaria.

3.9 Algoritmi di enumerazione implicita

I problemi di ottimizzazione discreta possono venire affrontati, in alternativa ai metodi poliedrali, per mezzo di algoritmi di tipo enumerativo. Il concetto che sta alla base di questa classe di algoritmi è abbastanza semplice: bisogna enumerare sistematicamente tutte le soluzioni ammissibili del problema, valutarne la funzione obiettivo e scegliere la migliore. Ovviamente per problemi di grandi dimensioni è assolutamente impossibile enumerare tutte le possibili combinazioni di soluzioni ammesse dal problema discreto. La soluzione migliore è determinare delle strategie che consentano, analizzando il problema specifico, di scartare delle soluzioni del problema che risultano distanti dalla soluzione ottima. In tal caso si parla allora di *algoritmi di enumerazione implicita*.

3.9.1 Algoritmo branch and bound

Il Branch and Bound è una tecnica generale per la risoluzione di problemi di ottimizzazione combinatoria che si basa sulla scomposizione del problema originale in sottoproblemi più semplici da risolvere. Questo metodo è stato inizialmente proposto da A. H. Land e A. G. Doig nel 1960 per risolvere problemi di programmazione lineare. Supponiamo di avere un problema $P^0 = (z, F(P^0))$ dove z è la funzione obbiettivo del problema mentre $F(P^0)$ è la regione delle soluzioni ammissibili. La soluzione ottima è data da:

$$z^* = z(P^0) = z(x) : x \in F(P^0)$$

Suddividiamo il problema P^0 i K sottoproblemi $P^1, P^2, ..., P^k$. Anche lo spazio delle soluzioni verrà suddiviso in modo tale che valga la seguente relazione:

$$\bigcup_{i=0}^{K} = F(P^0)$$

Preferibilmente lo spazio delle regioni ammissibili viene suddiviso in modo tale che:

$$F(P^i) \cup F(P^j) = \emptyset \qquad \forall P^i, P^j : i \neq j$$

Questo processo di ramificazione (branching) si può rappresentare mediante un albero decisionale come mostrato in figura 3.8,

ogni nodo rappresenta un sottoproblema mentre ogni arco la relazione di discendenza. Risolvere il problema P^0 è quindi equivalente alla risoluzione

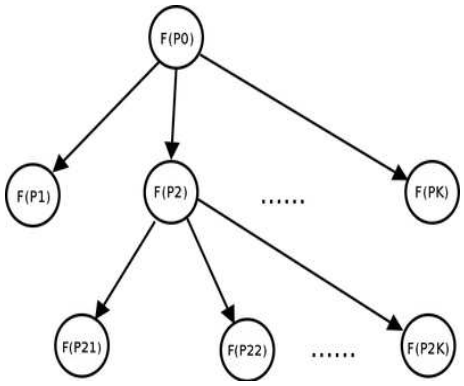


Figura 3.8: Albero dei sottoproblemi

dei K sottoproblemi descritti. Un sottoproblema P^i si può considerare risolto se si verifica uno dei casi seguenti:

1. Si determina la soluzione ottima di P^i
2. Si dimostra che $F(p^i)$ è impossibile
3. Si dimostra che $Z(P^i) \geq Z^{best}$ la soluzione del sottoproblema è peggiore della migliore conosciuta.

Nel caso non si riesca a risolvere un sottoproblema questo viene ramificato. Per ognuno di essi è possibile definire un limite inferiore della soluzione in modo da seguire una strategia di esplorazione dell'albero più efficiente. Dato il limite $L(P^i)$ inferiore se si verifica che $L(P^i) \geq z^{best}$ è possibile escludere il sottoproblema i dato che la migliore soluzione che si può sperare di trovare è peggiore della soluzione ammissibile del problema originale. Per ricavare un limite inferiore $P^i = (z, F(p^0))$ è necessario ottenere un rilassamento $R(P^i) = (z_r, F_r(P^i))$ tale che:

1. $F_r(P^i) \subseteq F(P^i)$
2. $z_r(y) \leq z(y) \qquad \forall y \in F(p^i)$

Il problema rilassato è risolvibile in modo più semplice rispetto al problema originale, quindi è possibile trovarne la soluzione ottima che rappresenta un limite inferiore del problema originale. Il rilassamento inoltre deve essere scelto in modo che sia più vicino possibile (tight) al problema originale, in alcuni casi basta un rilassamento continuo (facilmente risolvibile attraverso l'algoritmo del simplesso), in altri casi può essere conveniente utilizzare altri rilassamenti come il *rilassamento surrogato* o il *rilassamento Lagrangiano*. Molti risolutori combinano metodi di enumerazione implicita con metodi poliedrali, un esempio è CPLEX che impiega l'algoritmo di *branch and cut*.

Algoritmo branch and cut Sviluppato negli anni 90, nasce come unione tra due tecniche:

- Branch and bound
- Cutting planes

Elimina i principali difetti di tali tecniche, in particolare:

- garantisce un rafforzamento dinamico del problema rispetto al branch and bound
- garantisce l'eliminazione del fenomeno del tailing off [1], tipico del cutting planes

Una possibile realizzazione si ottiene unendo algoritmo branch and bound col metodo dei Tagli di Gomory. Ad un generico nodo al quale corrisponde il problema originale più i vincoli di branching , i tagli generati sono ricavati a partire dal rilassamento continuo del problema corrente e hanno validità locale ossia relativa al nodo corrente ed ai suoi eventuali discendenti. Attraverso tecniche specifiche (Lifting o Sequential Lifting) è possibile ricavare tagli che abbiano validità globale. Questi vengono memorizzati in una struttura globale detta *pool dei vincoli*. Questo comporta che venga persa la dipendenza tra il nodo corrente ed i tagli per esso ricavati ossia qualsiasi nodo vede tutti i tagli appartenenti al pool globale. A livello di ciascun nodo vengono effettuate le seguenti operazioni:

1. Si definisce la formulazione corrente dato dal problema iniziale più i vincoli di branching
2. Si risolve il rilassamento continuo del problema corrente
3. Nel caso in cui la soluzione sia frazionaria si scandisce il pool di vincoli alla ricerca di vincoli violati da aggiungere alla formulazione corrente e si risolve il nuovo rilassamento
4. Se la nuova nuova soluzione è ancora frazionaria e tutti i vincoli del pool sono soddisfatti si ricercano nuovi tagli globali da inserire nel pool (separazione), oppure si esegue il branching

La definizione di efficaci procedure di separazione è il punto cruciale dell'algoritmo, per esse esistono due diverse metodologie di sviluppo:

- procedure di separazione general-purpose, applicabili a ogni generico problema di programmazione lineare intero.
- procedure di separazione specifiche per una data classe di problemi, si sviluppa cioè una procedura ad hoc per il problema in esame.

[1] il principale problema dell'algoritmo del piano di taglio è la bassa velocità di convergenza; in molti casi pratici, l'algoritmo esegue una lunga sequenza di passi in cui il valore della soluzione ottima del problema di partenza cambia pochissimo.

3.10 Algoritmi approssimati

Trovandoci di fronte ad istanze di un problema NP hard, possiamo prevedere che i tempi di soluzione siano elevati. Molto spesso però, il tempo a nostra disposizione per risolverli è limitato e dobbiamo per forza di cose abbassare le nostre pretese accontentandoci di una buona soluzione, non necessariamente ottima, purché ottenuta rapidamente. Inoltre osserviamo che l'ottimalità è molto spesso una proprietà importante dal punto di vista teorico, ma di incerto significato in pratica, quest perché possono esserci discrepanze tra realtà e modello. Tale distanza è dovuta anche alla difficoltà (o spesso impossibilità) di conoscere esattamente tutte le grandezze in gioco, pertanto i coefficienti che compaiono nel modello sono affetti da errori spesso considerevoli, alcuni vincoli possono essere stati tralasciati, etc. Ciò contribuisce a rendere in molti casi poco rilevante dal punto di vista pratico la ricerca dell'ottimo teorico. In tali casi si fa ricorso ad algoritmi approssimati, detti anche euristiche perché fanno ricorso a conoscenze della struttura del particolare problema e delle caratteristiche delle soluzioni ottime. Nel caso di uso di algoritmi euristici è importante (ma non sempre facile, né possibile) effettuare un'analisi dell'errore che si commette accettando la soluzione ottenuta invece di quella ottima. Dato un problema di ottimizzazione l'errore relativo compiuto da un algoritmo approssimato vale:

$\mathcal{E}_r = \frac{Z_a - Z_{opt}}{Z_{opt}}$ Dove Z_a è la soluzione approssimata mentre Z_{opt} è la soluzione ottima. Un algoritmo è ϵ*approssimato* con $\epsilon \geq 0$ se:

$$\mathcal{E}_r \leq \epsilon$$

Uno schema di approssimazione è un algoritmo approssimato tale che, per ogni $\epsilon \geq 0$ produce una soluzione con $\mathcal{E}_r \leq \epsilon$ Uno schema di approssimazione viene definito:

- *polinomiale* se la sua complessità è polinomiale rispetto a n, dove n è la dimensione dell'input
- *pienamente polinomiale* se la sua complessità è polinomiale in n e in $\frac{1}{\epsilon}$

Esistono numerosi schemi approssimati, in questo ambito ci limitiamo ad enunciare i più significativi:

- algoritmi greedy(voraci)
- algoritmi di ricerca locale

I primi determinano la soluzione attraverso una sequenza di decisioni parziali (localmente ottime), senza mai tornare, modificandole, sulle decisioni prese. Questi algoritmi sono di facile implementazione e notevole efficienza

computazionale, ma, sia pure con alcune eccezioni di notevole rilievo, non garantiscono l'ottimalità, e a volte neppure l' ammissibilità. Esiste comunque una classe di problemi che vengono risolti all'ottimo dagli algoritmi greedy che viene denominata classe dei *matroidi*. La ricerca locale invece a partire da una soluzione ammissibile, cerca possibili soluzioni migliori ottenute apportando piccole modifiche alla soluzione data. I due tipi di algoritmo possono venire utilizzati in sequenza, ricorrendo ad un greedy per generare una soluzione ammissibile, che successivamente si tenta di migliorare con un algoritmo di ricerca locale.

Capitolo 4

Diffusione dell'innovazione nei social networks

4.1 Introduzione

In questo capitolo focalizzeremo la nostra attenzione sulle principali dinamiche che governano la diffusione dell'innovazione all'interno di un social network. Particolare enfasi verrà posta sul cosidetto *modello a soglia lineare* originariamente proposto in [1] e [2]. Molti modelli proposti in letteratura rappresentano il social network attraverso un grafo i cui nodi rappresentano gli utenti facenti parte della rete, mentre gli archi rappresentano le relazioni tra di essi. Secondo il modello a soglia lineare ad ogni utente viene assegnata una soglia λ_i che rappresenta la personale preferenza rispetto ad un prodotto o ad una innovazione. Il comportamento dell'utente all'interno della rete viene ampiamente influenzato, oltre che dalle specifiche preferenze individuali, dalle decisioni prese dagli utenti vicini [13] come ampiamente provato in [5, 6]. Un altro aspetto che verrà affrontato è quello del *set coesivo massimale* recentemente proposto da Yildiz *et al.*[7] basato sul modello a soglia lineare e che estende un'idea proposta in [5]. Un gruppo viene detto coesivo se non adotta un'innovazione a partire da un set di innovatori non appartenenti al set stesso. In [7] viene mostrato un importante risultato sui set coesivi che afferma: dato il più grande set coesivo della rete è possibile definire lo stato finale, in termini di utenti che adottano l'innovazione, della rete a partire da un set di innovatori di partenza. In relazione alla determinazione del set coesivo massimale vedremo due metodi: uno basato su un problema di ottimizzazione discreto e uno basato sulla programmazione lineare. Un'altro problema ampiamente trattato in letteratura è quello della *massimizzazione dell'influenza* che può essere riassunto nel modo seguente: Trovare il set di r utenti che massimizza la diffusione dell'innovazione all'interno della rete. Questo problema è NP hard e in letteratura sono stati proposti numerosi metodi risolutivi [9, 10, 11, 12, 8] basati tipicamente

su algoritmi greedy. Tratteremo in particolare due estensioni del problema classico della massimizzazione dell'influenza proposte in [23]. La prima rappresenta una generalizzazione del problema classico della massimizzazione dell'influenza e si prefigge di determinare il set di innovatori di partenza che massimizza la diffusione dell'innovazione in un orizzonte temporale limitato. Il secondo ha l'obbiettivo di determinare il set più piccolo di innovatori di partenza che diffonde l'innovazione su un set di utenti obbiettivo in K passi. Il seguente capitolo è tratto dall'articolo [23].

4.2 Concetti preliminari

La rete può essere efficacemente rappresentata attraverso un grafo $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ dove $\mathcal{V} = 1, 2, ..., n$ rappresenta il set di nodi della rete ed $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$ rappresenta il set di archi che li congiungono. Ogni nodo $i \in \mathcal{V}$ rappresenta un utente della rete mentre un arco orientato $(i, j) \in \mathcal{E}$ rappresenta l'influenza del nodo i sul nodo j. In particolare l'influenza che ha un nodo su se stesso non verrà rappresentata, ciò comporterà che gli archi del tipo (i, i) risulteranno assenti. Indichiamo con $\mathcal{N}_i = j \mid (j, i) \in \mathcal{E}$ l'insieme di *nodi vicini* che influenzano il nodo i_ esimo. L'informazione topologica del grafo viene rappresentata attraverso la *Matrice di adiacenza* $A \in \{0, 1\}^{n \times n}$ definita nel modo seguente:

$$A(i, j) = \begin{cases} 1 \text{ se esiste un arco diretto dal nodo } i \text{ al nodo } j \\ 0 \text{ altrimenti} \end{cases}$$

Definiamo quindi la *Matrice di adiacenza scalata* $\hat{A} \in [0, 1]^{n \times n}$:

$$\hat{A}(i, j) = \frac{A(i, j)}{|\mathcal{N}_j|}$$

4.3 Set coesivo massimale

Per formalizzare il concetto di set coesivo è necessario introdurre alcune definizioni. Innanzitutto assegniamo ad ogni nodo della rete una *soglia* $\lambda_i \in [0, 1]$ che modella la preferenza personale di ciascun utente verso il prodotto o l'innovazione. Quest'informazione può essere codificata attraverso una matrice delle soglie $\Lambda = diag([\lambda_1, \lambda_2, ..., \lambda_n])$ nel quale soltanto i termini posti nella diagonale sono diversi da zero. Definiamo quindi ϕ_0 come il *set di innovatori di partenza* ossia il set di utenti che all'istante iniziale $t = 0$ adottano l'innovazione. A partire da essi l'innovazione si diffonde attraverso la rete, quindi ad un certo istante t la popolazione degli innovatori può essere indicata dal set ϕ_t. Il set di utenti che adottano l'innovazione nell'intervallo temporale $[0, t]$ è dato da $\Phi_t = \bigcup_{j=0}^{t} \phi_j$. Secondo il *modello*

a soglia lineare il generico nodo i che all'istante di tempo t non ha ancora adottato l'innovazione la adotta al tempo $t+1$ se vale la seguente condizione:

$$\frac{|\Phi_t \bigcap \mathcal{N}_i|}{|\mathcal{N}_i|} = \frac{|\bigcup_{j=0}^{t} \phi_j \bigcap \mathcal{N}_i|}{|\mathcal{N}_i|} \geq \lambda_i \tag{4.1}$$

Quindi un nodo i addotta l'innovazione se il rapporto tra il numero di vicini che adottano l'innovazione e il numero totale dei suoi vicini supera la soglia λ_i. Vediamo un semplice esempio:

esempio 4.1. *Consideriamo la semplice rete in figura 4.1, nel quale* $\lambda_1 = \lambda_2 = 0.49$ *e* $\lambda_3 = \lambda_4 = 0.60$. *I nodi cerchiati di rosso rappresentano i nodi che hanno già adottato l'innovazione. Anche il nodo 2 adotterà l'innovazione al passo successivo dato che la sua soglia di decisione viene superata* $\lambda_2 \leq \frac{2}{3}$

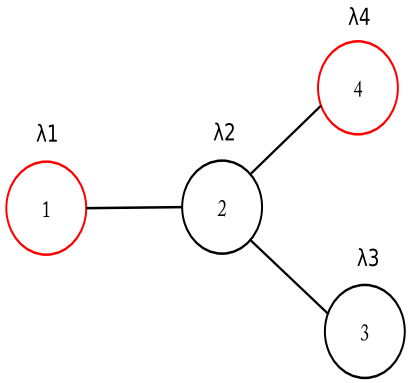


Figura 4.1: Esempio di rete con $n = 4$

Ad un certo punto la diffusione dell'innovazione si arresta configurando la popolazione finale di utenti che adottano l'innovazione che denotiamo con Φ^*. Associamo ad ogni set di nodi un vettore caratteristico così definito:

Definizione 4.3.1. *Dato un set* $X \in \mathcal{V}$ *questo sarà caratterizzato da un vettore caratteristico* $\boldsymbol{x} \in \{0, 1\}$ *dove* $x_i = 1$ *solo se* $i \in X$ *mentre risulta nullo negli altri casi.*

Ora abbiamo tutti gli strumenti per formalizzare il concetto di set coesivo:

Definizione 4.3.2. *Un set* $X \in \mathcal{V}$ *è coesivo se per ogni nodo* $i \in X$ *vale la seguente relazione:*

$$\frac{|X \cap \mathcal{N}_i|}{|\mathcal{N}_i|} \geq 1 - \lambda_i$$

In altre parole un set $X \subset \mathcal{V}$ è coesivo se per ogni nodo $i \in X$ il rapporto tra il numero di nodi vicini appartenenti al set coesivo e il numero totale di vicini è strettamente minore di λ_i. Un'importante risultato mostrato in [7] in relazione ai set coesivi può essere riassunto dal seguente lemma:

Lemma 4.3.1. *Dato il set di innovatori di partenza $\phi_0 \subset \mathcal{V}$ sia $\mathcal{M} \subset \mathcal{V} - \phi_0$ il massimo set coesivo del complemento di ϕ_0. Il set di innovatori finali $\mathbf{\Phi}^*$ è dato dalla seguente relazione:*

$$\mathbf{\Phi}^* = \mathcal{V} - \mathcal{M} \tag{4.2}$$

4.4 Identificazione del massimo set coesivo

La conoscenza del massimo set coesivo del complemento del set di innovatori di partenza ϕ_0 permette di identificare immediatamente il set di utenti $\mathbf{\Phi}^*$ che alla fine del processo di diffusione adottano l'innovazione come mostrato nel lemma 4.2.1. In questa sezione vedremo due possibili approcci per identificare il massimo set coesivo. Partiamo dalla seguente definizione:

Definizione 4.4.1. *Un set $X \subset \mathcal{V}$ è coesivo se per ogni nodo $i \in X$ il suo vettore caratteristico $\boldsymbol{x}$ soddisfa la seguente proprietà :*

$$x^T \hat{A}(\cdot, i) \geq 1 - \lambda_i$$

Dimostrazione. Ciò discende dalla seguente osservazione:

$$\boldsymbol{x}^T \hat{A}(\cdot, i) = \frac{\boldsymbol{x}^T \hat{A}(\cdot, i)}{\mathbf{1}^T \hat{A}(\cdot, i)} = \frac{|X \cap \mathcal{N}_i|}{\mathcal{N}_i}$$

□

Il massimo set coesivo può essere determinato tramite il problema di programmazione lineare binario:

Proposizione 4.4.1. *Dato un grafo $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ e il set di innovatori di partenza $\phi_0 \subset \mathcal{V}$ con vettore caratteristico $\boldsymbol{y}$. Il massimo set coesivo contenuto in $V \backslash \phi_0$ con vettore caratteristico $\boldsymbol{x}$ può essere ottenuto tramite il seguente problema di programmazione binario:*

$$\begin{cases} \max \quad \mathbf{1}^T \cdot \boldsymbol{x} & \\ \quad \boldsymbol{x} & \leq \mathbf{1} - \boldsymbol{y} \\ \left[I - \Lambda - \hat{A}^T \right] \cdot \boldsymbol{x} & \leq \mathbf{0} \\ \quad \boldsymbol{x} \in \{0,1\}^n & \end{cases} \tag{4.3}$$

Dimostrazione. In primo luogo osserviamo dal primo vincolo della 4.3 che:

$$\mathcal{M} \cap \phi_0 \iff \boldsymbol{x} + \boldsymbol{y} \leq 1$$

inoltre dato che $\mathcal{M}$ è un set coesivo dal lemma 4.2.1 risulta che:

$$\begin{array}{c} \forall i \in \mathcal{M}, \qquad \boldsymbol{x}^T \hat{A}(\cdot, i) \geq 1 - \lambda_i \\ \Updownarrow \\ \forall i \in \mathcal{V}, \qquad \boldsymbol{x}^T \hat{A}(\cdot, i) \geq (1 - \lambda_i) x_i \\ \Updownarrow \\ \boldsymbol{x}^T \hat{A} \geq \boldsymbol{x}^T [I - \Lambda] \end{array}$$

Che può essere riscritto come il secondo vincolo del problema di programmazione 4.3. In definitiva il set coesivo calcolato attraverso il problema di programmazione binario 4.3 è massimale dato che viene massimizzata la funzione obbiettivo. □

Come dimostrato in [7] esiste sempre un set coesivo massimale, che potrebbe essere anche un insieme vuoto, e risulta essere unico. Finora abbiamo mostrato un approccio basato su un problema di programmazione binario. Questo comporta che il problema risulterà non trattabile per istanze molto grandi dei dati. Di seguito presentiamo un approccio alternativo basato sulla risoluzione di una serie di problemi di programmazione lineari continui.

4.4.1 Set coesivo massimale attraverso un LPP

Consideriamo in primo luogo una versione rilassata del problema 4.3 e caratterizziamone la soluzione.

Proposizione 4.4.2. *Dato il grafo $\mathcal{G}(\mathcal{V}, \mathcal{E})$ e il set di innovatori di partenza $\phi_0 \subset \mathcal{V}$ con vettore caratteristico $\boldsymbol{y}$. Sia $\mathcal{M}$ il set coesivo massimale contenuto in $V \backslash \phi_0$. Consideriamo il seguente problema di programmazione lineare:*

$$\begin{array}{ll} \max & \mathbf{1}^T \cdot \boldsymbol{x} \\ \left\{ \begin{array}{rcll} \boldsymbol{x} & \leq & \mathbf{1} - \boldsymbol{y} & (a) \\ \left[I - \Lambda - \hat{A}^T \right] \cdot \boldsymbol{x} & \leq & \mathbf{0} & (b) \\ \boldsymbol{x} & \geq & 0 & \end{array} \right. \end{array} \tag{4.4}$$

sia $x^* \in [0,1]^n$ la soluzione ottima del problema di programmazione lineare 4.4.

1. *Per ogni $i \in \mathcal{M}$, $x_i^* = 1$.*
2. *se $\boldsymbol{x}^* \in \{0,1\}^n$ tale che $\mathcal{M} = \{i \in V \mid x_i^* = 1\}$.*

Dimostrazione. 1. Il primo risultato può essere dimostrato per assurdo. Assumiamo che $\boldsymbol{x}$ sia una soluzione ottima della (4.4) tale che $Z = \{i \in \mathcal{M} \mid x_i < 1\}$ sia non vuoto, e consideriamo $\boldsymbol{x'}$ dove $x'_i = x_i$ se $i \notin Z$ altrimenti $x'_i = 1$. Possiamo affermare che $\boldsymbol{x'}$ soddisfa il set di vincoli della (4.4).

Infatti il vincolo (a) viene verificato banalmente da $\boldsymbol{x'}$, dato che per ogni $i \in Z$ vale $y_i = 0$.

Consideriamo ora il vincolo (b). Per ogni $i \in V \setminus Z$ vale

$$\boldsymbol{x'}^T \hat{A}(\cdot, i) \geq \boldsymbol{x}^T \hat{A}(\cdot, i) \geq (1 - \lambda_i) x_i = (1 - \lambda_i) x'_i$$

Mentre per ogni $i \in Z \subseteq X$ si ha

$$\boldsymbol{x'}^T \hat{A}(\cdot, i) \geq \frac{|X \cap \mathcal{N}_i|}{|\mathcal{N}_i|} \geq 1 - \lambda_i = (1 - \lambda_i) x'_i$$

dato che $\mathcal{M}$ è un set coesivo. Come mostrato nella dimostrazione della proposizione 4.3.1 questi due risultati implicano che $\boldsymbol{x'}$ soddisfi il vincolo (b). In definitiva, dato che $\mathbf{1}^T \cdot \boldsymbol{x'} > \mathbf{1}^T \cdot \boldsymbol{x}$, allora $\boldsymbol{x}$ non è una soluzione ottima e pertanto contraddice l'assunzione iniziale.

2. Se $\boldsymbol{x}^* \in \{0, 1\}^n$ allora $\boldsymbol{x}^*$ è anche la soluzione ottima del problema binario (4.3) e di conseguenza il suo vettore caratteristico è l'insieme $\mathcal{M}$.

□

Possiamo quindi presentare l'algoritmo iterativo per il calcolo del massimo set coesivo. Prima di cominciare facciamo alcuni commenti sull'algoritmo:

1. Ad ogni passo viene risolto un problema di programmazione lineare, ogni nodo i con $x_i^{(k)} < 1$ non appartiene al set $\mathcal{M}$ in accordo con la proposizione 4.4.2. Quindi al passo 5 possiamo estendere il set $Y^{(k)}$ con vettore caratteristico $\boldsymbol{y}^{(k)}$ con il set $Y^{(k+1)}$. Chiaramente il set $\mathcal{M}$ che vogliamo determinare è anche quello massimale contenuto in $V \setminus Y^{(k+1)}$ in accordo con la proposizione 4.4.2.

2. Se la soluzione ottima trovata è di tipo binario siamo allora certi che questa rappresenti il massimo set coesivo $\mathcal{M}$.

Di seguito viene mostrato in pseudo codice l'algoritmo di calcolo del set coesivo massimale basato sulla programmazione lineare.

. Determinazione del set coesivo massimale tramite un problema di programmazione lineare:

INPUT: Un grafo $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ con matrice di adiacenza scalata $\hat{A}$ e matrice delle soglie Λ. Il set $\phi_0 \subset V$ con vettore caratteristico $\boldsymbol{y} \in \{0,1\}^n$.
OUTPUT: Il vettore caratteristico del massimo set coesivo X contenuto in $V \setminus Y$.

---

1. *Sia $k = 0$ e $\boldsymbol{y}^{(0)} = \boldsymbol{y}$.*
2. *Sia $\boldsymbol{x}^{(k)} \in [0,1]^n$ la soluzione ottima del problema di programmazione lineare*

$$\begin{array}{ll} \max & \mathbf{1}^T \cdot \boldsymbol{x} \\ \left\{ \begin{array}{rcl} \boldsymbol{x} & \leq & \mathbf{1} - \boldsymbol{y}^{(k)} \\ \left[I - \Lambda - \hat{A}^T\right] \cdot \boldsymbol{x} & \leq & \mathbf{0} \\ \boldsymbol{x} & \geq & 0 \end{array} \right. \end{array}$$

3. ***While*** *$\boldsymbol{x}^{(k)} \notin \{0,1\}^n$*
   (a) *Sia $k = k + 1$.*
   (b) *Sia $\boldsymbol{y}^{(k)} = \left\lceil \mathbf{1} - \boldsymbol{x}^{(k-1)} \right\rceil$.*
   (c) *Sia $\boldsymbol{x}^{(k)} \in [0,1]^n$ sia una soluzione ottima del problema di programmazione lineare*

$$\begin{array}{ll} \max & \mathbf{1}^T \cdot \boldsymbol{x} \\ \left\{ \begin{array}{rcl} \boldsymbol{x} & \leq & \mathbf{1} - \boldsymbol{y}^{(k)} \\ \left[I - \Lambda - \hat{A}^T\right] \cdot \boldsymbol{x} & \leq & \mathbf{0} \\ \boldsymbol{x} & \geq & 0 \end{array} \right. \end{array}$$

4. *End* ***while****.*
5. *$\boldsymbol{x}^{(k)}$ è il vettore caratteristico di $\mathcal{M}$.*

L'algoritmo appena presentato permette di calcolare il massimo set coesivo attraverso una serie di passi computazionali. Il numero di passi eseguiti dall'algoritmo dipendono dalla dimensione della rete, dalla dimensione del set di innovatori di partenza e dalla dimensione del set coesivo massimale. In particolare vale il seguente risultato:

Proposizione 4.4.3. *L'algoritmo necessita di un numero di ripetizioni del ciclo while pari a $\bar{k}$ il cui valore è dato dalla seguente relazione:*

$$\bar{k} \leq n - \mid \phi_0 \mid - \mid \mathcal{M} \mid + 1.$$

Dimostrazione. Nell'algoritmo ad ogni passo del ciclo while la cardinalità del vettore $\boldsymbol{y}$ viene incrementato al massimo di una unità, di conseguenza il massimo numero di incrementi è pari a $n - \mid \phi_0 \mid - \mid \mathcal{M} \mid$. □

4.5 Massimizzazione dell'influenza in tempo finito

La massimizzazione dell'influenza rappresenta uno dei casi di studio più importanti in relazione alla diffusione dell'innovazione nei social networks. Il problema può essere riassunto nel modo seguente: Data una rete descritta dal grafo $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, trovare un set di partenza $\phi_0 \subseteq \mathcal{V}$ di r innovatori che massimizzano la diffusione dell'innovazione all'interno della rete. Questo problema classico consente di determinare quali nodi della rete adottano l'innovazione dopo che il processo di diffusione dell'innovazione si è arrestato, idealmente per $t = \infty$. Nella realtà si è più interessati a capire quale sia il comportamento della rete nelle prime fasi di immissione del prodotto o dell'innovazione. Da qui nasce la necessità di determinare quale set di innovatori massimizzano l'innovazione in un orizzonte temporale limitato. D'ora in poi definiremo il problema come *massimizzazione dell'influenza in tempo finito* (IMFTP(r,k)) con parametri r e k. Questo problema può essere formalizzato nel seguente modo: Data una rete caratterizzata dal grafo $\mathcal{G}(\mathcal{V}, \mathcal{E})$ determinare il set di innovatori iniziali ϕ_0 di cardinalità $|\phi_0| = r$ che massimizza la diffusione dell'innovazione in k passi temporali. Il problema appena enunciato rappresenta una naturale estensione del problema classico di massimizzazione dell'influenza e pertanto, per orizzonti temporali estesi, fornirà gli stessi risultati di quest'ultimo. Il problema di massimizzazione dell'influenza in tempo finito è di natura combinatoria, infatti per determinare quale set di r innovatori massimizza l'influenza in k passi è necessario provare tutte le possibili combinazioni di n utenti presi in gruppi di r elementi. In definitiva il numero di possibili combinazioni è la seguente:

$$\binom{n}{r} = \frac{n!}{r!(n-r)!}$$

Nelle prossime sezioni vedremo una possibile soluzione del problema attraverso un problema di programmazione lineare binario.

4.5.1 Definizioni preliminari

Per comprendere meglio la definizione del problema attraverso un problema di programmazione binario è necessario introdurre alcuni concetti preliminari. Partiamo dalla definizione del *vettore evoluzione*:

Definizione 4.5.1. *Un vettore $\boldsymbol{w}$ associato al set di innovatori di partenza ϕ_0 viene definito vettore evoluzione se le sue componenti $\boldsymbol{w}_0, \boldsymbol{w}_1, \ldots, \boldsymbol{w}_k$ sono vettori caratteristici rispettivamente di $\phi_0, \Phi_1, \ldots, \Phi_k$ in accordo col modello a soglia lineare.*

L'informazione sull'evoluzione della rete in relazione al propagarsi dell'innovazione all'interno di essa viene efficacemente espressa dal vettore evoluzione come mostrato nel seguente lemma:

Lemma 4.5.1. *Dato il grafo $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, sia $\phi_0 \subset \mathcal{V}$ un set di partenza, e ad ogni istante temporale t sia $\boldsymbol{x}_t$ e $\boldsymbol{w}_t$ il vettore caratteristico rispettivamente di ϕ_t and Φ_t. Vale allora la seguente proprietà.*

$$\forall t \in \mathbb{R}, \qquad [\hat{A}^T + \Lambda]\boldsymbol{w}_t - \Lambda \boldsymbol{w}_{t+1} \geq \mathbf{0}_n \tag{4.5}$$

Dimostrazione. Un nodo $i \in \mathcal{V}$ tale che $i \notin \Phi_t$ adotta l'innovazione al tempo $t+1$, se e solo se

$$\boldsymbol{w}_t \hat{A}(:, i) \geq \lambda_i \tag{4.6}$$

Ciò deriva dalla seguente osservazione:

$$\boldsymbol{w}_t^T \hat{A}(:, i) = \frac{\boldsymbol{w}_t^T A(\cdot, i)}{\mathbf{1}^T A(\cdot, i)} = \frac{\mid \Phi_t \cap \mathcal{N}_i \mid}{\mid \mathcal{N}_i \mid}$$

Quindi:

$$\forall i \in \phi_{t+1}, \quad \hat{A}^T(\cdot, i)\boldsymbol{w}_t \geq \lambda_i$$

Di conseguenza:

$$\begin{gathered} \forall i \in \mathcal{V}, \quad \hat{A}^T(:, i)\boldsymbol{w}_t \geq \lambda_i \boldsymbol{x}_{t+1}(i) \\ \Updownarrow \\ \hat{A}^T \boldsymbol{w}_t - \boldsymbol{\Lambda} \boldsymbol{x}_{t+1} \geq \mathbf{0}_n \end{gathered}$$

Dato che $\boldsymbol{x}_{t+1} = \boldsymbol{w}_{t+1} - \boldsymbol{w}_t$ allora risulta che:

$$\forall t \in \mathbb{N}, \qquad [\hat{A}^T + \Lambda]\boldsymbol{w}_t - \Lambda \boldsymbol{w}_{t+1} \geq \mathbf{0}_n$$

□

Quindi dato un set ϕ_0, tutte le componenti del vettore evoluzione ad esso associato rispettano la relazione (4.5). Il vettore evoluzione è unico e rispecchia l'evoluzione della rete in seguito all'introduzione del set di innovatori ϕ_0. È importante sottolineare che soltanto il vettore evoluzione rispecchia l'andamento della diffusione dell'innovazione all'interno della rete, infatti ci possono essere altri vettori che soddisfano la (4.5) ma che non rappresentano la reale evoluzione della rete. Definiamo questi ultimi *vettori k*.

Definizione 4.5.2. *Sia ϕ_0 un set di partenza con vettore caratteristico $\hat{\boldsymbol{w}}_0$, e $\hat{\boldsymbol{w}}_0, \hat{\boldsymbol{w}}_1, \ldots, \hat{\boldsymbol{w}}_k$ siano $k+1$ vettori di n elementi. Il vettore $\hat{\boldsymbol{w}}^T = [\hat{\boldsymbol{w}}_0{}^T \ \hat{\boldsymbol{w}}_1^T \ \ldots \ \hat{\boldsymbol{w}}_k^T]$ è un vettore k associato al set ϕ_0 se $\forall i \in \{1, \ldots, k\}$ le componenti $\hat{\boldsymbol{w}}_i \in \{0,1\}^n$, e rispettano l'equazione (4.5).*

Per capire meglio questo concetto consideriamo il seguente esempio .

esempio 4.2. *Consideriamo la semplice rete in figura 4.2, nel quale $\lambda_1 = \lambda_2 = 0.49$ e $\lambda_3 = \lambda_4 = 0.60$. Sia $\phi_0 = \{2\}$, il cui vettore caratteristico è $\boldsymbol{x}_0 = [0100]^T$, allo stesso modo del set $\phi_1 = \{1, 2, 3, 4\}$ con vettore caratteristico $\boldsymbol{w}_1 = [1111]^T$. In accordo con il Lemma 4.5.1 e alla definizione 4.5.1, il*

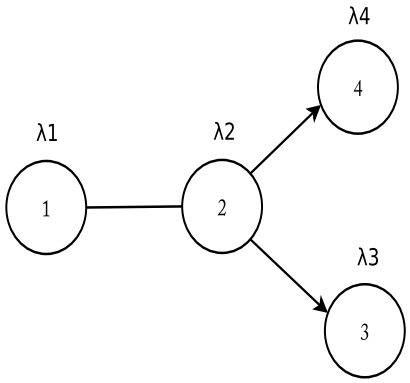


Figura 4.2: Esempio di rete con $n = 4$

vettore $\boldsymbol{w} = [01001111]^T$ *è certamente un vettore k per k=1 associato al set* ϕ_0 *e risulta essere l'unico vettore rappresentativo dell'evoluzione della rete. È facile verificare che* $\boldsymbol{w}$ *non è l'unico vettore k per k=1 associato al set* ϕ_0*, infatti anche i vettori* $\hat{\boldsymbol{w}}' = [01000100]^T$ *e* $\hat{\boldsymbol{w}}'' = [01000101]^T$ *sono vettori k ma non rappresentano la reale evoluzione della rete.*

Tra il vettore evoluzione e i possibili vettori-k intercorre la relazione espressa nel seguente lemma.

Lemma 4.5.2. *Sia* ϕ_0 *un set di partenza il cui vettore evoluzione è* $\boldsymbol{w}$*. Per ogni possibile vettore-k* $\hat{\boldsymbol{w}}'$ *associato a* ϕ_0 *vale la seguente relazione:*

$$\boldsymbol{w}_k \geq \hat{\boldsymbol{w}}_k.$$

Dimostrazione. In accordo al modello a soglia lineare, se un individuo i può adottare l'innovazione al tempo $t \leq k$, allora per ogni $j \geq t$ del vettore evoluzione vale $\boldsymbol{w}_j(i) = 1$, mentre per un vettore-k $\hat{\boldsymbol{w}}$ può verificarsi che $\hat{\boldsymbol{w}}_j(i) = 1$ oppure $\hat{\boldsymbol{w}}_j(i) = 0$, in entrambi i casi la (4.5) è rispettata. Se un individuo i non può adottare l'innovazione al passo k, allora per ogni componente $j \geq k$ deve valere $\boldsymbol{w}_j(i) = \hat{\boldsymbol{w}}_j(i) = 0$. Di conseguenza $\boldsymbol{w}_k \geq \hat{\boldsymbol{w}}_k$. □

4.5.2 Determinazione del set ottimale di innovatori

In base alle definizioni viste precedentemente possiamo definire un algoritmo per la risoluzione del problema IMFTP(r,k) basato su un problema di programmazione binario. Data una rete $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, la scelta dei vincoli garantisce che la soluzione del problema di programmazione binario è un vettore-k associato al set ϕ_0^* di r nodi, che massimizza la diffusione dell'innovazione in $\mathcal{G}$ in k passi. Massimizzando la funzione obbiettivo viene garantito inoltre che la soluzione trovata, oltre a essere un vettore-k che

soddisfa i vincoli, è anche il vettore evoluzione associato al set ϕ_0^*. Consideriamo quindi il problema di programmazione binario associato al problema IMFTP(r,k).

Proposizione 4.5.1 . *Dato un grafo $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ con $\mid \mathcal{V} \mid = n$, consideriamo il seguente problema di programmazione binario:*

$$\begin{cases} \max \quad [\mathbf{1}_{nk}^T \quad (nk)\mathbf{1}_n^T] \cdot \boldsymbol{w} & & & \\ \quad \mathbf{1}_n^T \boldsymbol{w}_0 & = & \boldsymbol{r} & (a) \\ \quad \forall i \in \{1, \ldots, k\}, & & & \\ \quad [\hat{A}^T + \Lambda]\boldsymbol{w}_{i-1} - \Lambda \boldsymbol{w}_i & \geq & \mathbf{0}_n & (b) \\ \quad \boldsymbol{w} & \in & \{0,1\}^{n(k+1)} & (c) \end{cases} \tag{4.7}$$

Dove $\boldsymbol{w}^T = [\boldsymbol{w}_0^T \ \boldsymbol{w}_1^T \ \ldots \ \boldsymbol{w}_k^T]$. Sia $\boldsymbol{w}^*$ una soluzione ottima della (4.7). Allora:

- *$\boldsymbol{w}_0^*$ è il vettore caratteristico del set di partenza ϕ_0^* che risolve il problema IMFTP(r, k);*
- *$\boldsymbol{w}^*$ è il vettore di evoluzione associato a ϕ_0^*.*

Dimostrazione. Per la definizione 4.5.2 segue che i vincoli (b) e (c) garantiscono che ogni soluzione ammissibile di (4.7) sarà un vettore-k associato a ϕ_0^*. Possiamo dimostrare le precedenti proprietà in due passi:

1. Dimostriamo in primo luogo che il vettore $\boldsymbol{w}_k^*$ è il vettore caratteristico associato a Φ_k^* a partire dal set ϕ_0^*;
2. Dimostriamo quindi che $\boldsymbol{w}_0^*$ è il vettore evoluzione associato a ϕ_0^*.

Analizziamo separatamente i due passaggi.

1. Dimostriamo questa affermazione per assurdo. Sia la componente $\boldsymbol{w}_k^*$ della soluzione ottima il vettore caratteristico del set Φ. Supponiamo che $\Phi \neq \Phi_k$ partendo da ϕ_0^*. Dato che $\boldsymbol{w}^*$ è un vettore-k, per il lemma 4.5.2 risulta $\mid \Phi_k \mid > \mid \Phi \mid$. Sia $\mid \Phi_k \mid = m \leq n$, di conseguenza questo può valere al massimo $\mid \Phi_k^* \mid = m - 1$. Per il vettore caratteristico $\boldsymbol{w}_k'$ di Φ_k' risulta di conseguenza:

   $$(nk) \cdot \mathbf{1}_{nk}^T \boldsymbol{w}_k' = nkm$$

   La soluzione ottima Φ^* può valere al massimo:

   $$\mid \phi_0^* \mid = \mid \Phi \mid = m - 1,$$

   perciò

   $$\begin{aligned} [\mathbf{1}_{nk}^T \ \ (nk) \cdot \mathbf{1}_n^T]\boldsymbol{w}^* &= k(m-1) + nk(m-1) \\ &= nkm - nk + mk - k \end{aligned}$$

Dato che $mk - nk$ assume un valore certamente non positivo, segue che:

$$[\mathbf{1}_{nk}^T\ (nk) \cdot \mathbf{1}_n^T]\boldsymbol{w}^* < (nk) \cdot \mathbf{1}_n^T \boldsymbol{w}'_k$$

Perciò risulta che Φ non può essere la $k-$esima componente della soluzione ottima.

2. Dato che stiamo massimizzando la funzione obbiettivo la componente ω_0^* della soluzione rappresenta il vettore evoluzione del set ϕ_0^* associato alla soluzione ottima.

□

4.6 Diffusione dell'innovazione su un target di utenti

Un interessante estensione del problema IMFTP(r,k) presentato precedentemente è il seguente: Minimizzare il set di innovatori di partenza ϕ_0 in modo da diffondere l'innovazione attraverso un set di nodi obbiettivo $\Phi_d \subseteq \mathcal{V}$ in k passi. Anche questo problema può essere caratterizzato tramite un problema di programmazione binario che mostriamo di seguito.

Proposizione 4.6.1 . *Dato un grafo $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ con $\mid \mathcal{V} \mid = n$, sia $\boldsymbol{w}^T = [\boldsymbol{w}_0^T\ \boldsymbol{w}_1^T\ \ldots\ \boldsymbol{w}_k^T]$ un vettore di $n(k+1)$ componenti e $\boldsymbol{x}_d$ sia il vettore caratteristico del set obiettivo $\Phi_d \subseteq \mathcal{V}$. Consideriamo il seguente problema di programmazione binario:*

$$\begin{array}{ll} \min & [\mathbf{1}_n^T\ \ \mathbf{0}_{nk}^T] \cdot \boldsymbol{w} \\ \left\{ \begin{array}{ccccc} \boldsymbol{w}_k & \geq & \boldsymbol{x}_d & & (a) \\ \forall i \in \{0, \ldots, k-1\}, & & & & \\ [\hat{A}^T + \Lambda]\boldsymbol{w}_{i-1} - \Lambda \boldsymbol{w}_i & \geq & \mathbf{0}_n & & (b) \\ & & & & \\ \boldsymbol{w} & \in & \{0,1\}^{n(k+1)} & & (c) \end{array} \right. \end{array} \tag{4.8}$$

Sia $\boldsymbol{w}^*$ la soluzione ottima della (4.8). Allora $\boldsymbol{w}_0^*$ è il vettore caratteristico del set di partenza più piccolo che diffonde l'innovazione nel set di nodi obbiettivo Φ_d in k passi.

Dimostrazione. I vincoli (b) e (c) garantiscono che la soluzione ottima $\boldsymbol{w}^*$ è un vettore-k. In particolare il vincolo(a) garantisce che, partendo dal set ϕ_0^* con vettore caratteristico $\boldsymbol{w}_0^*$ un insieme $\Phi_k^* \supseteq \Phi_d$ può essere raggiunto in k passi, questo implica che la soluzione ottima trovata può diffondere collateralmente l'innovazione, oltre che al set obbiettivo, anche ad altri nodi della rete. Infine la minimizzazione della funzione obbiettivo garantisce che il set di innovatori di partenza sarà il più piccolo possibile. □

Anche in tal caso la complessità computazionale cresce in maniera non lineare rispetto all'orizzonte temporale. Si può considerare la versione rilassata del problema 4.8 per determinare un limite inferiore della cardinalità del set di innovatori di partenza.

Proposizione 4.6.2. *Dato un grafo $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ con $\mid \mathcal{V} \mid = n$, sia $\boldsymbol{w}^T = [\boldsymbol{w}_0^T \ \boldsymbol{w}_1^T \ \dots \ \boldsymbol{w}_k^T]$ un vettore di $n(k+1)$ componenti e $\boldsymbol{x}_d$ sia il vettore caratteristico del set obiettivo $\Phi_d \subseteq \mathcal{V}$. Consideriamo il seguente problema di programmazione lineare:*

$$\begin{array}{l} \min \quad [\mathbf{1}_n^T \ \ \mathbf{0}_{nk}^T] \cdot \boldsymbol{w} \\ \left\{ \begin{array}{rcll} \boldsymbol{w}_k & \geq & \boldsymbol{x}_d & (a) \\ \forall i \in \{0, \dots, k-1\}, & & & \\ [\hat{A}^T + \Lambda]\boldsymbol{w}_{i-1} - \Lambda \boldsymbol{w}_i & \geq & \mathbf{0}_n & (b) \\ \\ \boldsymbol{w} & \geq & \mathbf{0}_{n(k+1)} & (c) \end{array} \right. \end{array} \tag{4.9}$$

Sia w^* la soluzione ottima della (4.9). Valgono allora le seguenti proprietà:

1. *Se $\boldsymbol{w}^* \in \{0,1\}^{n(k+1)}$ allora $\boldsymbol{w}^*$ è anche la soluzione ottima della (4.8);*
2. *$\lceil \mathbf{1}_n^T \boldsymbol{w}_0^* \rceil$ è un limite inferiore della cardinalità del set di innovatori di partenza più piccolo che diffonde l'innovazione attraverso i nodi obbiettivo Φ_d in k passi.*

Per brevità nei capitoli successivi faremo riferimento al problema di massimizzazione in tempo finito indicandolo come problema 1. Indicheremo invece il problema di diffusione dell'innovazione su un target di utenti come problema 2. Infine definiremo come problema 3 quello inerente la determinazione del set coesivo massimale.

Capitolo 5

Euristiche e strategie di riduzione dei dati

5.1 Introduzione

Nel capitolo precedente sono state presentate le principale tematiche inerenti la diffusione dell'innovazione nei social networks. Nella maggior parte dei casi questi problemi presentano un'elevata complessità computazionale e pertanto sono difficilmente trattabili quando si vogliano analizzare reti reali. In quest'ottica quindi ci si accontenta di una soluzione subottima che approssimi quella teorica. In questo capitolo vedremo un'estensione dell'algoritmo greedy proposto in [11] con lo scopo di ottenere un simulatore che dia una soluzione approssimata per il problema 1. Verrà quindi presentato per il problema 2 un algoritmo che riduce preliminarmente la dimensione della rete scartando quei nodi che non possono far parte della soluzione. Mostreremo infine per il problema 1 come sia possibile tarare l'ottimizzatore CPLEX per ottenere un soluzione subottima andando a modificare il comportamento dell'algoritmo branch and cut.

5.2 Euristica per il problema 1

In letteratura esistono numerose euristiche volte a rendere trattabile il problema classico di massimizzazione dell'influenza. I primi studi sistematici basati su problemi di ottimizzazione discreti sono stati condotti da Kempe, Kleinberg e Tardos [8]. In particolare ci soffermeremo su un'euristica studiata appositamente per il modello di diffusione a soglia lineare proposta in [11] anch'essa costruita per risolvere in maniera approssimata il problema della massimizzazione dell'influenza. In questo lavoro di tesi si è cercato di

adattare tale euristica per risolvere il problema 1 proposto in [23]. L'euristica si basa sulla costruzione di grafi diretti aciclici (DAGs) a partire dalla rete che si vuole analizzare. La proprietà principale di tali grafi è che per essi è possibile determinare l'evoluzione della rete e quindi risolvere il problema classico della massimizzazione dell'influenza in tempo lineare. Per stimare quindi l'evoluzione della diffusione nella rete di partenza, che necessariamente non sarà aciclica, si costruiscono dei grafi diretti aciclici centrati su ciascun nodo della rete in esame. In definitiva l'idea di base è quella di stimare il comportamento della rete di partenza come somma delle evoluzioni locali date da ciascun grafo aciclico.

5.2.1 Algoritmo greedy

Rappresentiamo la rete attraverso un grafo $\mathcal{G} = (V, E, \omega)$ dove $V = 1, 2, ..., n$ rappresenta il set di nodi della rete mentre $E \subset V \times V$ rappresenta il set di archi che li congiungono. In particolare $\omega : V \times V \to [0, 1]$ viene detta *funzione dei pesi* ed è così definita : vale 0 se $\omega(u, v) \notin E$, mentre vale $\sum_{u \in V} \omega(u, v) \leq 1$ negli altri casi. Questa è un'estensione rispetto al modello presentato in [23] in quanto gli archi vengono pesati attribuendogli i valori continui che vanno da 0 a 1. Questo modello risulta più aderente alla realtà perché caratterizza il fatto che un utente può risultare influenzato in maniera differente da ciascun utente della rete. Nell'ambito del modello a soglia lineare proposto definiamo $\sigma_L(S)$ il numero atteso di nodi che adottano l'innovazione a partire dal set S di innovatori di partenza per un orizzonte temporale illimitato. In particolare il valore di $\sigma_L(S)$ fissati i valori delle soglie e la topologia di rete è assolutamente univoco, tuttavia nel lavoro proposto in [11] ci si riferisce al valor medio ottenuto al variare delle soglie per poter valutare ogni particolare configurazione di innovatori di partenza. L'euristica proposta da W.Chen *et al.*[11] nasce per risolvere il problema classico della massimizzazione dell'influenza, nel presente lavoro di tesi si è cercato di adattare tale euristica al problema 1, in particolare viene sfruttato il meccanismo di selezione degli innovatori di partenza per poi determinare il numero di utenti che adottano l'innovazione in maniera deterministica cioè fissando il valore delle soglie, oltre che la topologia di rete. Consideriamo ora una funzione $f : 2^V \to N$, dove N rappresenta l'insieme dei numeri naturali, costruita su un sottoinsieme di nodi V, questa funzione è *submodulare* se per ogni $S \subseteq T \subseteq V$ e per ogni $u \in V \backslash T$ vale: $f(S \bigcup \{u\}) - f(S) \geq f(T \bigcup \{u\}) - f(T)$. La funzione f è *monotona* se $f(S) \leq f(T)$ per ogni $S \subseteq T \subseteq V$.

Queste proprietà valgono anche per σ_L come dimostrato in [8] e possono essere sfruttate per costruire un algoritmo greedy che seleziona ad ogni passo computazionale un nodo da aggiungere al set S di innovatori di partenza che garantisce il massimo margine di incremento $f(S \bigcup \{\omega\}) - f(S)$ per il numero

atteso di utenti che adottano l'innovazione. Di seguito mostriamo la schema generale dell'algoritmo greedy proposto in [11]:

Algoritmo greedy.

INPUT: r, f
OUTPUT: S

---

1. *Inizializza* $S=\emptyset$
2. ***for*** i*=1 to* r ***do***
3. *Seleziona* $u = argmax_{\omega \in V \setminus S}(f(S \bigcup \{\omega\}) - f(S))$
4. $S = S \bigcup \{u\}$
5. ***endfor***
6. *restituisci* S

Al passo 1 viene inizializzato S che conterrà gli innovatori di partenza. Al secondo passo viene inizializzato il ciclo di selezione degli r innovatori. Al passo 3 vediamo che il criterio di selezione dei nodi da parte dell'algoritmo prevede la scelta di quello che garantisce il margine massimo di incremento della funzione f tra due passi successivi dell'algoritmo. Questo schema può essere impiegato efficacemente per costruire un algoritmo che selezioni i nodi che determinano il massimo margine di incremento del numero atteso σ_L di utenti che adottano un innovazione tra due passi successivi dell'algoritmo. Alla fine del processo viene restituito il set S contenente gli innovatori. Dobbiamo rimarcare il fatto che trattandosi di un algoritmo greedy, questo non prevede che si possa modificare la soluzione fornita ad un passo precedente di selezione. In definitiva il problema di identificare r innovatori che massimizzino la diffusione dell'innovazione rimane un problema NP Hard di cui l'algoritmo greedy fornisce una soluzione approssimata.

5.2.2 Diffusione dell'innovazione nei grafi aciclici

Il problema classico della massimizzazione dell'influenza è un problema NP-hard per reti arbitrarie. Inoltre per tali reti risulta NP-hard determinare il margine di utenti attesi tra due iterazioni successive dell'algoritmo greedy. Tuttavia in [11] si dimostra che questa operazione può essere effettuata in tempo lineare se il grafo è aciclico. In definitiva l'idea di base è che il processo di diffusione si sviluppa localmente e pertanto può essere studiato scomponendo la rete di partenza in tante sottoreti centrate sui nodi di tale rete. Un grafo aciclico diretto (o DAG, Directed acyclic graph) è un particolare tipo di grafo diretto che non ha cicli diretti, ovvero comunque

scegliamo un vertice del grafo non possiamo tornare ad esso percorrendo gli archi del grafo. Consideriamo il grafo aciclico $D(V, E, \omega)$ e un sottoinsieme $S \subseteq V$. Per ogni nodo $u \in V$ sia $ap(u)$ la probabilità di attivazione del nodo u a partire dal set di innovatori di partenza S. Per definizione $ap(u) = 1$ se $u \in S$. La probabilità di attivazione nei grafi aciclici può essere determinata attraverso la seguente relazione:

$$ap(v) = \sum_{u \in V/\{v\}} ap(u)\omega(u, v) \tag{5.1}$$

Questa relazione è alla base della stima del numero atteso di utenti finali, infatti ci consente di determinare la probabilità di attivazione di ciascun nodo radice di ogni grafo aciclico. Vedremo nella prossima sezione come venga determinato il valore di $\sigma_L(S)$. In [11] viene proposto il seguente algoritmo per determinare la probabilità di attivazione dei nodi in un grafo aciclico.

Algoritmo di calcolo delle probabilità di attivazione.

Determina ap(u) per ogni nodo u del grafo aciclico D dato il set di partenza S:

---

1. *$\forall u$ in $D, ap(u) = 0; \forall u \in S, ap(u) = 1$;*
2. *Ordina topologicamente ogni nodo raggiungibile da S in D in una sequenza ρ*
3. ***for** each node $u \in \rho/S$ in accordo con la sequenza ρ **do***
4. *$ap(v) = \sum_{u \in V \setminus \{v\}} ap(u)\omega(u, v)$*
5. ***endfor***

In [11] viene mostrato come l'algoritmo appena presentato sia poco efficiente e ne viene fornita una forma alternativa che mostreremo nella sezione 5.4.

5.2.3 Stima dell'influenza nelle reti reali

Tipicamente le reti reali non appartengono alla classe dei grafi aciclici quindi non possiamo applicare direttamente i risultati ottenuti precedentemente. Il concetto di base proposto in [11] consiste nella costruzione di un grafo aciclico per ognuno dei nodi della rete di partenza. La probabilità di attivazione

del nodo radice v di un grafo aciclico a partire da S si determina considerando soltanto l'effetto locale dato dai nodi appartenenti al grafo. In base a questa considerazione si può stimare il valore atteso $\sigma_L(S)$ attraverso la seguente relazione:

$$\sigma_L(S) = \sum_{u \in V} ap(v, S) \tag{5.2}$$

Dove $ap(v, S)$ è la probabilità di attivazione del nodo v, nella rete aciclica nel quale v è il nodo radice, a partire dal set S di innovatori di partenza. Ovviamente massimizzare il valore di σ_L costituisce un problema NP hard ma grazie all'algoritmo greedy è possibile fornirne una stima date le proprietà di σ_L. Il valore dato dalla relazione 5.2 si riferisce all'evoluzione della rete per un orizzonte temporale illimitato dato che l'euristica proposta in [11] è rivolta a risolvere il problema classico della massimizzazione dell'influenza. Il nostro proposito è adattare l'euristica al problema 1 proposto in [23] e per far questo modificheremo la costruzione dei grafi aciclici, inoltre per determinare il valore atteso del numero di utenti che adottano l'innovazione non possiamo utilizzare la relazione 5.2 bensì risolveremo iterativamente la seguente relazione in base all'orizzonte temporale scelto:

$$[\hat{A}^T + \Lambda]\boldsymbol{w}_{i-1} - \Lambda \boldsymbol{w}_i \geq \mathbf{0}_n$$

Questa rappresenta la regola di adozione dell'innovazione da parte degli utenti per ogni passo temporale K, applicandola iterativamente è possibile ricavare l'evoluzione della rete sino al passo K.

5.3 Costruzione dei grafi aciclici

I nodi che costituiscono un grafo aciclico vengono selezionati in base all'influenza che essi hanno rispetto al nodo sul quale è centrata la rete. Il criterio di selezione va scelto accuratamente, infatti scegliere un criterio che scarta pochi nodi determina una maggiore precisione della soluzione finale a discapito della velocità di esecuzione dell'algoritmo greedy. In particolare l'estensione di base che proponiamo consiste nel costruire dei grafi aciclici nel quale inserire i nodi che distano non più di K passi dal nodo radice in modo da simulare un orizzonte temporale limitato. Questo criterio viene utilizzato congiuntamente a quello originariamente proposto in [11]. Di seguito mostriamo l'algoritmo di costruzione dei grafi aciclici:

Costruzione dei grafi aciclici.

INPUT: $G(V, E, \omega)$,v,θ,K
OUTPUT:$D(X, Y, \omega)$

---

1. $X = \emptyset; Y = \emptyset, Inf(u, v) = 0; Inf(v, v) = 1$
2. ***While*** $max_{u \in V \setminus X} Inf(u, v) \geq \theta$ *and* $maxdistance(u) \leq K$ ***do***
3. $x = argmax_{u \in V \setminus X} Inf(u, v)$
4. $Y = Y \bigcup \{(x, u) : u \in X\}$
5. $X = X \bigcup \{x\}$
6. ***for*** *each node* $u \in N^{in}(x)$***do***
7. $inf(u, v) += \omega(u, x) Inf(x, v)$
8. ***endfor***
9. ***end while***
10. *Restituisci il grafo aciclico* $D(X, Y, \omega)$

Vediamo di commentare l'algoritmo, al primo passo viene inizializzato l'algoritmo, X rappresenta il set che conterrà i nodi del grafo aciclico, Y è il set di archi del grafo. Il termine $Inf(u, v)$ rappresenta la probabilità di influenza del nodo u sul nodo v, vediamo infatti al primo passo che $Inf(v, v) = 1$ dato che la probabilità di influenza del nodo radice su se stesso è massima. Al passo due vediamo il criterio di arresto dell'algoritmo basato sulla selezione dei nodi che hanno un'influenza maggiore di una soglia θ, secondo il criterio proposto in [11] e che non distano più di K passi in base all'orizzonte temporale. Al passo tre viene selezionato tra tutti i nodi quello maggiormente influente, quindi al passo successivo vengono aggiunti al grafo aciclico gli archi uscenti dal nodo x che contattano nodi già appartenenti al grafo. Al passo cinque viene aggiunto il nodo x. Dal passo sei al passo otto vengono aggiornati gli indici di influenza dei nodi che raggiungono il nodo x dato che questi attraverso quest'ultimo possono influenzare il nodo radice.

5.4 Algoritmo approssimato problema 1

Una volta costruiti i grafi aciclici centrati su ciascun nodo della rete di partenza siamo in grado di selezionare un set r di utenti maggiormente influenti secondo il criterio dell'algoritmo greedy. L'algoritmo per la determinazione delle probabilità di attivazione necessario a determinare σ_L risulta poco efficiente. Viene proposto in [11] un'alternativa basata sulla relazione lineare che intercorre tra due nodi in termini di probabilità di attivazione. Si dimostra infatti che l'incremento di probabilità di attivazione di un nodo v quando un nodo u viene a far parte del set di innovatori S è dato dalla seguente relazione lineare:

$$\Delta ap(v) = (1 - ap(u))\alpha_v(u) \tag{5.3}$$

Quindi è possibile calcolare le probabilità di attivazione di ciascun nodo calcolando preliminarmente i coefficienti α. I coefficienti possono essere determinati attraverso il seguente algoritmo come proposto in [11]:

Algoritmo di calcolo dei coefficienti α.

---

1. $\forall u \in D, \alpha_v(u) = 0, \alpha_v(v) = 1$
2. *Ordina topologicamente ogni nodo che può raggiungere v in una sequenza ρ nel quale v è il primo nodo della sequenza*
3. ***for** each node $u \in \rho \backslash (S \bigcup \{v\})$ in accordo con l'ordine di ρ **do***
4. $\alpha_v(u) = \sum_{x \in N^{out}(u) \bigcap \rho} \omega(u, x)\alpha_v(x)$
5. ***endfor***

Al primo passo vengono inizializzati i coefficienti di tutti i nodi del grafo aciclico. Notiamo che soltanto il coefficiente $\alpha_v(v)$ è non nullo dato che si tratta del nodo radice del grafo che si sta considerando. Il pedice v sta ad indicare proprio il nodo radice mentre il valore tra parentesi indica uno specifico nodo del grafo di cui si sta calcolando il coefficiente. I nodi che raggiungono la radice del grafo vengono ordinati dal più vicino al più lontano e in base a questa sequenza avviene l'aggiornamento dei coefficienti α_v secondo la relazione al passo 4. L'insieme $N^{out}(u)$ denota i nodi raggiungibili dal generico nodo u. Siamo ora in grado di presentare l'algoritmo di selezione

degli innovatori di partenza. L'algoritmo generale è uguale a quello proposto in letteratura eccetto che per l'algoritmo di costruzione dei grafi aciclici mostrato nella sezione 5.3. Vediamo prima da un punto di vista informale i passi principali compiuti dall'algoritmo:

- Fase preparatoria
  - L'algoritmo costruisce per ogni nodo della rete un grafo aciclico.
  - Per ogni rete determina localmente il nodo più influente.
  - Si assegna ad ogni nodo della rete un indice di influenza globale che determina l'influenza rispetto agli altri nodi.
- Flusso principale
  - L'algoritmo seleziona il nodo con indice di influenza globale più elevato.
  - Per ogni rete aciclica viene aggiornato l'indice di influenza locale dei nodi non appartenenti al set di innovatori di partenza determinando l'indice $\alpha_v(u)$ che definisce il legame lineare relativo alla probabilità di attivazione di un nodo in seguito alla selezione di un nuovo innovatore che lo influenza.
  - Per ogni nodo della rete di partenza viene aggiornato l'indice globale di influenza $IncInf(u)$ che determina quale nodo verrà selezionato come innovatore di partenza.

Di seguito mostriamo in maniera puntuale i passi dell'algoritmo:

Algoritmo di selezione del set di innovatori di partenza.

---

1. */*Fase preparatoria*/*
2. $S = \emptyset$
3. $\forall v \in V, Incinf(v) = 0$
4. ***for*** *each node* $v \in V$ ***do***
5. *Genera i grafi aciclici*
6. $\forall u \in D$ *imposta* $ap_v(u) = 0$
7. $\forall u \in D(v, \theta, K)$ *determina i coefficienti lineari* α
8. ***for*** *each* u *in* $D(v, \theta, K)$ ***do***
9. $IncInf(u) += \alpha_v(u)$
10. ***endfor***

11. ***endfor***
12. */*Ciclo di selezione del set di innovatori di partenza*/*
13. ***for*** $i = 1$ *to* r ***do***
14. $s = argmax_{v \in V \setminus S}\{IncInf(v)\}$
15. ***for*** *each* $v \in InfSet(s) \setminus S$ ***do***
16. */*Aggiornare* $\alpha_v(u)$ *per ogni nodo* u *che può raggiungere* s**/*
17. $\Delta\alpha_v(s) = -\alpha_v(s); \forall u \in S, \Delta\alpha_v(u) = 0$
18. *Ordina topologicamente i nodi che possono raggiungere* s *in* $D(v, \theta, K)$ *in una sequenza* ρ *nel quale* s *è il primo.*
19. *Determina* $\Delta\alpha_v(u)$ *per tutti i nodi* $u \in \rho$ *attraverso i passi* $3-5$ *dell'algoritmo che calcola i coefficienti* α *dove* $\rho \setminus (S \bigcup \{v\})$ *viene rimpiazzato da* $\rho \setminus (S \bigcup \{s\})$ *e* $\alpha_v()$ *viene sostituito con* $\Delta\alpha_v()$
20. $\alpha_v(u) += \Delta\alpha_v(u)$ *per ogni nodo* $u \in \rho$
21. $IncInf(u) += \Delta\alpha_v(u)(1 - ap_v(u))$ *per ogni nodo* $u \in \rho$
22. */*Aggiornare* $ap_v(u)$ *per ogni nodo* u *che può essere raggiunto da* s**/*
23. $\Delta ap_v(s) = 1 - ap_v(s); \forall u \in S, \Delta ap_v(u) = 0$
24. *Ordina topologicamente i nodi raggiungibili da* s *in* $D(v, \theta, K)$ *in una sequenza* ρ *nel quale* s *è il primo.*
25. *Determina* $\Delta ap_v(u)$ *per tutti i nodi* $u \in \rho$ *attraverso i passi* $3 - 5$ *dell'algoritmo che calcola le probabilità di attivazione dove* $\rho \setminus S$ *viene rimpiazzato da* $\rho \setminus (S \bigcup \{s\})$ *e* $ap_v()$ *viene sostituito con* $\Delta ap_v()$
26. $ap_v(u) += \Delta ap_v(u)$ *per ogni nodo* $u \in \rho$
27. $IncInf(u) -= \alpha_v(u)\Delta ap_v(u)$ *per ogni nodo* $u \in \rho$
28. ***endfor***
29. $S = S \bigcup \{s\}$
30. ***endfor***
31. *Restituisci* S

Vediamo di commentare l'algoritmo, in primo luogo il pedice v sta ad indicare che ci si riferisce al grafo aciclico centrato sul nodo v. I passi $1-12$ riguardano la fase preparatoria dell'algoritmo nel quale vengono creati i grafi aciclici e vengono determinati i coefficienti α che serviranno ad aggiornare successivamente le probabilità di attivazione. Al passo 9 ad ogni nodo v della rete viene assegnato un indice $IncInf(v)$ che rappresenta l'influenza che ogni nodo ha rispetto agli altri nodi della rete. I passi $12-31$ costituiscono la fase di selezione dei nodi che faranno parte degli innovatori di partenza. Viene innanzitutto selezionato il nodo che ha maggiore influenza sugli altri ossia quello con l'indice $IncInf(u)$ più alto. Per ogni nodo appartenente al set di influenza del seed appena selezionato è necessario aggiornare i coefficienti α e le probabilità di attivazione. Il set di influenza del seed è così definito:

$$InfSet(s) = \{v \in V : s \in D(v, \theta, K)\} \tag{5.4}$$

Quindi rappresenta l'insieme di nodi che possono essere influenzati dal nodo s. Successivamente l'algoritmo selezionerà tutte quelle reti acicliche in cui compare il nodo s, queste conterranno l'insieme di nodi che subiscono un'influenza dal seed s. Per ognuna delle reti acicliche aggiornerà i coefficienti α dei nodi in grado di raggiungere il nodo seed, successivamente aggiorna la probabilità di attivazione dei nodi raggiungibili da s. Questo procedimento viene ripetuto finché non sono stati selezionati r nodi che andranno a costituire il set di innovatori di partenza. Riassumendo l'algoritmo costruisce per ogni nodo della rete un grafo aciclico centrato in esso e per ognuno di esso determina localmente quale sia il nodo più influente caratterizzato da una maggiore capacità di attivare i suoi nodi vicini in base alla relazione lineare (5.3). Una volta completata questa fase si determina fra tutti i possibili grafi aciclici il nodo che risulta maggiormente influente secondo l'indice globale $IncInf(v)$. Questo procedimento viene ripetuto ad ogni iterazione andando a scandire via via quei nodi della rete che non siano già appartenenti all'insieme S, questo comporta che venga selezionato un solo nodo alla volta e che la soluzione ad un passo algoritmico precedente non possa essere modificata secondo lo schema classico degli algoritmi greedy.

5.5 Taratura dell'ottimizzatore

Finora abbiamo visto delle strategie basate sullo studio delle proprietà del problema di partenza. Ora mostreremo un approccio completamente opposto basato sullo studio dell'algoritmo impiegato dal risolutore CPLEX per risolvere i problemi di ottimizzazione discreti. Le impostazioni di default del risolutore permettono comunque di ottenere la soluzione di un dato problema, però ci siamo posti un semplice domanda: È possibile cambiare le

impostazioni dell'ottimizzatore in modo da ottenere una soluzione subottima di un dato problema? La domanda nasce dalla necessità di risolvere i numerosi problemi riscontrati col problema 1, in particolare dovuti ad un eccessiva richiesta di memoria e di tempo di calcolo anche per istanze di piccole dimensioni. L'algoritmo di risoluzione impiegato per i problemi di ottimizzazione discreti è il *branch & cut*, per maggiori dettagli si veda il capitolo 2. In particolare l'algoritmo costruisce una versione rilassata del problema di partenza, costruisce quindi un albero i cui nodi costituiscono delle versioni rilassate del problema iniziale caratterizzati da un set di vincoli differenti per scandire una diversa regione delle soluzioni ammissibili. Quindi la ricerca viene indirizzata verso la porzione di albero che porta verso la soluzione ottima. Abbiamo constatato che questo meccanismo è alla base della grande richiesta di risorse da parte del risolutore, inoltre la convergenza verso la soluzione ottima cresce molto lentamente e non linearmente rispetto alle dimensioni dell'albero di ricerca. Questi fattori ci hanno portato a verificare se fosse possibile ridurre il numero di nodi complessivamente generati durante una simulazione. Effettivamente è possibile impostare l'algoritmo *branch & cut* in modo da risolvere un numero inferiore di problemi rilassati, come atteso a seconda del numero di sottoproblemi risolti abbiamo ottenuto una soluzione più o meno approssimata con un dispendio di risorse computazionali ragionevoli. Il fatto che la soluzione converga lentamente rispetto al numero di sottoproblemi risolti fa si che sia conveniente impostare il risolutore in modo che risolva un numero relativamente basso di sottoproblemi rilassati, infatti soltanto nelle prime fasi il risolutore migliora notevolmente la soluzione al variare del numero di ramificazioni dell'algoritmo *branch & cut*, poi ad un certo punto la velocità di convergenza verso la soluzione ottima diminuisce drasticamente. In figura 5.1 viene mostrato il risultato di una simulazione condotta su una rete di 50 elementi impostando un orizzonte temporale K=6 e un set di innovatori di partenza pari a 8. Si nota chiaramente come la soluzione migliora non linearmente rispetto al numero di problemi rilassati risolti.

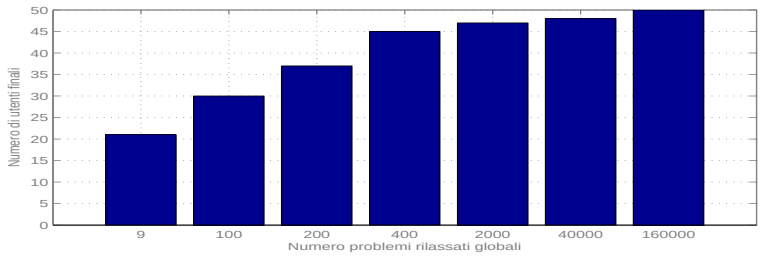


Figura 5.1: Velocità di convergenza della soluzione

5.6 Riduzione preliminare della rete

Anche il problema 2 è basato su un problema di ottimizzazione discreta. In quest'ottica non abbiamo però optato per una soluzione approssimata, piuttosto abbiamo preferito studiare meglio il problema per capire se ci fosse una possibilità di ridurre la dimensione dei dati di partenza andando a scartare dei nodi della rete. Questo approccio consente di ridurre il tempo computazionale senza approssimare la soluzione che continuerà a essere quella ottima. Ricordiamo che il problema 2 si prefigge di determinare il set più piccolo di innovatori di partenza che assicura che venga adottata l'innovazione da parte di un set di utenti obbiettivo in K passi temporali. L'idea di fondo è che se un certo nodo u dista più di K passi da tutti i nodi appartenenti al set obbiettivo allora questo nodo non può essere selezionato come innovatore di partenza. Nel semplice esempio in figura 5.2 viene mostrata una porzione di rete, il nodo colorato in rosso è l'utente obbiettivo, se l'orizzonte temporale fosse K=1 i nodi 7 e 6 verrebbero scartati.

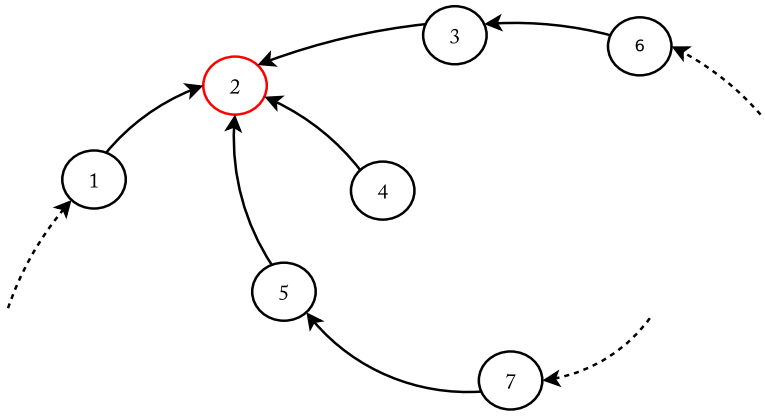


Figura 5.2: Interpretazione grafica

In base a questa idea è stato sviluppato un semplice algoritmo che data la rete di partenza, il set di nodi obbiettivo e l'orizzonte temporale K scarta i nodi che distano più di K passi dal nodo obbiettivo.

Di seguito viene mostrato uno pseudocodice che riassume i passi principali compiuti dall'algoritmo. L'implementazione dell'algoritmo prevede che venga fornita anche la visualizzazione del grafo di partenza e della rete ottenuta in seguito alla riduzione. Per maggiori dettagli si faccia riferimento all'appendice B.

Algoritmo di riduzione della rete.

INPUT: n,K,$|Xd|$
OUTPUT: $\hat{A}^T + \Lambda, \Lambda, Xd, \hat{A}_r^T + \Lambda_r, \Lambda_r, Xd_r$

---

1. *Subset*$= \emptyset$, *Del_ nodes*$= \emptyset$
2. *Genera random* A, Λ, Xd
3. *Costruisce* $\hat{A}^T + \Lambda, \Lambda_r, Xd$
4. ***for i=1 to K do***
5. $\forall u \in Xd$
6. */* Seleziona i nodi che distano i_passi da u e salvali in Subset */*
7. ***end for***
8. *Del_ nodes*$= V \setminus$ *Subset;*
9. */* Riduzione della rete */*
10. *Cancella le righe di A indicate nell'insieme Del_ nodes*
11. *Cancella le colonne di A indicate nell'insieme Del_ nodes*
12. *Cancella le righe di* Λ *indicate nell'insieme Del_ nodes*
13. *Cancella le colonne di Xd indicate nell'insieme Del_ nodes*
14. *Costruisci* $\hat{A}_r^T + \Lambda_r, \Lambda_r, Xd_r$
15. *Restituisci* $\hat{A}^T + \Lambda, \Lambda, Xd, \hat{A}_r^T + \Lambda_r, \Lambda_r, Xd_r$

Vediamo di commentare lo pseudocodice, il pedice r sta ad indicare che ci si riferisce alla rete ridotta. L'algoritmo prende in ingresso la dimensione della rete di partenza, l'orizzonte temporale e il numero di utenti obbiettivo. L'insieme *Subset* rappresenta una struttura dati che contiene i nodi che distano *i_ passi* dal nodo obbiettivo u considerato. Al passo 8 V rappresenta l'insieme dei nodi della rete di partenza. Dal passo 9 al passo 12 vengono cancellate le righe e le colonne della matrice di adiacenza e della matrice delle soglie, mentre al passo 13 vengono cancellate le colonne del vettore Xd. Le righe e le colonne cancellate risultano in corrispondenza agli elementi contenuti nell'insieme *Del_ nodes* in modo tale da poter costruire la rete ridotta, infatti l'insieme *Del_ nodes* contiene quei nodi che vengono scartati dall'algoritmo. All'ultimo passo vengono restituite tutte le strutture dati necessarie a svolgere il problema 2 sia per la rete di partenza sia per quella ottenuta dalla riduzione.

Capitolo 6

Implementazione dei simulatori

6.1 Introduzione

Finora abbiamo analizzato gli aspetti teorici riguardanti i problemi di ottimizzazione. In questo capitolo daremo una visione di insieme degli approcci implementativi spiegando il funzionamento dei principali blocchi che costituiscono i simulatori. Vedremo quindi come siano stati concepiti e i motivi che hanno portato, per ognuno di essi, ad una particolare implementazione.

6.2 Euristica problema 1

Nel precedente capitolo abbiamo mostrato l'euristica sul problema 1 basata sull'implementazione di un semplice algoritmo greedy che ad ogni iterazione sceglie l'utente che massimizza il margine atteso di utenti che adottano l'innovazione. L'euristica in questione è stata implementata in ambiente Matlab per la sua facilità di utilizzo, anche se sarebbe stato preferibile un altro linguaggio di programmazione che sfruttasse meglio le risorse in termini di processore. Questo è un aspetto da non sottovalutare perché può comportare una perdita di prestazione in funzione del tempo di calcolo richiesto per effettuare le simulazioni. Per motivi di implementazione non è stato possibile costruire il codice seguendo fedelmente la struttura presentata nello pseudocodice mostrato nel precedente capitolo. In figura 6.1 viene mostrata la struttura del codice che è stato costruito in maniera modulare, ricalcando quelle che sono le fasi salienti dell'esecuzione del codice piuttosto che i blocchi costituenti visti nella teoria. Questa scelta nasce per rendere più agevole la fase di debug, in particolare il ciclo di selezione è stato incorporato in un unico blocco data la natura sequenziale delle operazioni che risultano più agevoli da analizzare passo per passo. Vediamo dal punto di vista qualitativo ciascun blocco e la funzione da esso implementata:

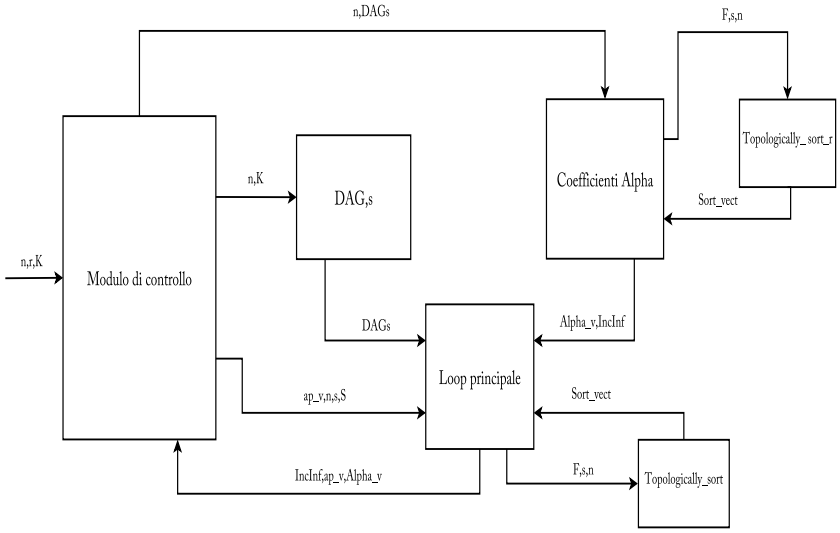


Figura 6.1: Struttura dell'euristica

- **Modulo di controllo**: Il modulo di controllo coordina tutte le fasi di esecuzione del codice, in particolare richiede all'utilizzatore i dati della simulazione, quindi avvia la fase preparatoria, una volta completata esegue il ciclo di selezione degli utenti che faranno parte del set di innovatori di partenza.
- **DAG,s**: Il blocco DAG,s genera la rete diretta di partenza, quindi costruisce su ciascun nodo di essa un grafo aciclico.
- **Coefficienti Alpha**: Questo modulo calcola per ciascun nodo della rete in ognuno dei grafi aciclici un coefficiente che permette di determinare tramite una relazione lineare la probabilità di attivazione.
- **Topologically sort r**: Il blocco dato un grafo aciclico di partenza ordina topologicamente i nodi che possono raggiungere un certo nodo obbiettivo.
- **Topologically sort**: Il blocco ordina topologicamente i nodi di un grafo aciclico che vengono raggiunti da uno specifico nodo della rete.
- **Loop principale**: Questo modulo aggiorna le probabilità di attivazione e i coefficienti α di ciascun nodo della rete, quindi aggiorna l'indice *IncInf* che viene utilizzato per determinare quale nodo sia candidato a far parte del set di innovatori di partenza.

Nelle prossime sezioni vedremo le fasi salienti del flusso di esecuzione della simulazione mostrando quali blocchi sono coinvolti.

6.2.1 Modulo di controllo

Per capire meglio il funzionamento dell'euristica è necessario capire le operazioni principali svolte dal codice. Non entreremo nei dettagli implementativi, piuttosto analizzeremo i blocchi principali e le fasi in cui essi intervengono durante una simulazione. Partiamo dal modulo di controllo che riassume efficacemente i passi salienti della simulazione. Questa si divide in due fasi fondamentali:

1. Fase preparatoria
2. Fase di selezione del set di innovatori di partenza

Si inizia dalla richiesta da parte del modulo di controllo dei dati della simulazione:

```
n=input(Inserisci il numero di utenti della rete:);
K=input(Inserire orizzonte temporale K:);
r=input(Inserire la cardinalit del set di partenza:);
```

In base a questi dati di partenza vengono inizializzate le strutture dati:

```
IncInf=zeros(1,n);
ap_v=zeros(n,n);
S=[];
```

La prima struttura dati contiene un indice dell'influenza che ciascun nodo ha sugli altri nodi della rete, la seconda conterrà le probabilità di attivazione di ciascun nodo nei grafi aciclici in cui esso compare, infine l'ultima struttura dati conterrà il set di innovatori di partenza che in questa fase preparatoria risulta necessariamente vuoto. Una volta inizializzate le strutture dati il modulo di controllo richiama il modulo DAGs :

```
[DAGs,A]= DAGS(n,K);
```

In base ai dati di partenza il blocco invocato costruirà i grafi aciclici centrati sui nodi della rete di partenza. Una volta costruiti i grafi aciclici è necessario generare i coefficienti α, che serviranno in seguito per l'aggiornamento delle probabilità di attivazione, e gli indici iniziali *IncInf* che vengono utilizzati per selezionare i nodi strategici della rete. Questa fase viene effettuata tramite l'invocazione della seguente funzione:

```
[Alpha_v,IncInf]=Coefficienti_alpha(DAGs,n);
```

Queste operazioni concludono la fase preparatoria. La fase di selezione viene ripetuta r volte in base al dato di partenza fornito dall'utente. Per prima cosa viene selezionato l'utente maggiormente influente ossia quello caratterizzato dal valore più alto di *IncInf*:

```
[Y,s] = max(IncInf);
```

Viene quindi invocata la funzione che aggiorna le probabilità di attivazione, gli incrementi di influenza e i coefficienti α:

```
[IncInf,ap_v,Alpha_v]=
Loop_principale(DAGs,Alpha_v,ap_v,IncInf,n,s,S);
```

In particolare la funzione effettua gli aggiornamenti sopracitati perché in seguito alla selezione di un nuovo nodo candidato ad essere innovatore di partenza la possibilità di adozione dell'innovazione da parte degli altri utenti della rete viene a modificarsi. Conclusa questa fase si aggiunge definitivamente all'interno del set di innovatori di partenza l'innovatore appena selezionato, quindi viene posto a zero l'indice *IncInf* dei nodi del set S dato che nel ciclo successivo questi non dovranno essere più selezionati come innovatori di partenza:

```
S=union(S,s);
IncInf(:,S)=0;
```

Nelle sezioni successive vedremo con maggiore dettaglio i blocchi che sono stati invocati dal modulo di controllo durante la simulazione, mostreremo inoltre il funzionamento di alcuni blocchi(topologically sort r,topologically sort) che sono stati omessi nella fase di esecuzione ma che vengono chiamati all'interno dei blocchi che abbiamo appena analizzato.

6.2.2 Grafi aciclici

L'interfaccia della funzione che genera i grafi aciclici è la seguente:

```
function [DAGs,A] =DAGS(n,K)
```

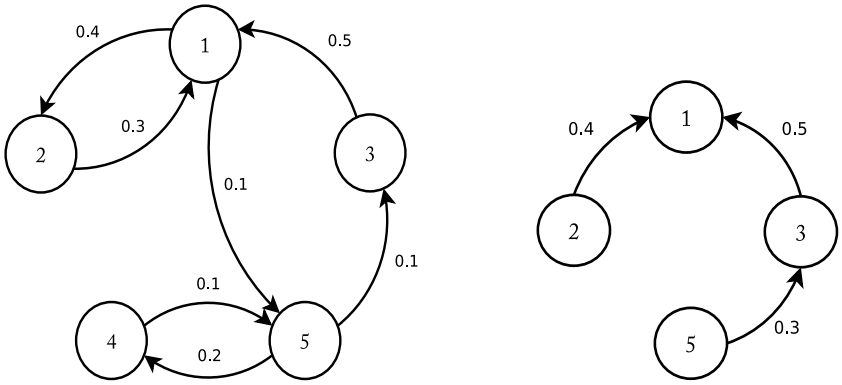
La funzione prende in ingresso la dimensione della rete di partenza e l'orizzonte temporale e in base a questi restituisce i grafi aciclici che vengono salvati in un array di celle. Le operazioni svolte dalla funzione si snodano secondo le seguenti fasi:

1. La prima operazione consiste nel generare in maniera random la rete di partenza in base la dimensione specificata dall'utente. Viene quindi generata la matrice di adiacenza pesata $\hat{A}^T$ e viene effettuato un controllo sul rispetto delle specifiche della rete che ricordiamo non deve contenere *self loop* ossia non deve contenere archi uscenti da un nodo che ritornano sul nodo stesso, inoltre la rete deve essere connessa ossia non possono comparire nodi o gruppi di nodi che costituiscono delle sottoreti disgiunte.

2. Una volta generata la rete di partenza, viene selezionato ogni singolo nodo e viene costruito su di esso un grafo che contiene solo i nodi che da esso distano non più di k passi.

3. Costruite le sottoreti centrate sui nodi della rete di partenza l'algoritmo genera i grafi aciclici in base al parametro θ che definisce un criterio di arresto nella selezione dei nodi che effettivamente andranno a far parte del grafo aciclico, infatti quei nodi la cui influenza sul nodo nel quale è centrato il grafo aciclico non supera la soglia verranno scartati. La soglia θ rappresenta un parametro empirico proposto da [11]. Per le nostre simulazioni abbiamo sostanzialmente utilizzato gli stessi valori proposti in letteratura $\Theta \in \{\frac{1}{320}, \frac{1}{80}\}$.

esempio 6.1. *Consideriamo la rete in figura 6.2(a) nel quale sono già stati selezionati i nodi che distano al più tre passi dal nodo radice* $v = 1$. *Consideriamo quindi di impostare il valore del parametro* $\theta = \frac{1}{7}$.



(a) Rete di partenza (b) Grafo aciclico

Figura 6.2: Grafo di partenza e relativo grafo aciclico

La funzione estrae il grafo aciclico mostrato in figura 6.2(b) attraverso i seguenti passi:

(a) Vengono inizializzati i valori della variabile Inf(u,v) che rappresenta la probabilità di attivazione del nodo v in seguito all'influenza esercitata dal nodo u.
$Inf(1,1) = 1$;
$Inf(2,1) = 0$;
$Inf(3,1) = 0$;
$Inf(4,1) = 0$;
$Inf(5,1) = 0$;
Come si nota soltanto il nodo radice ha un valore di verso da zero e necessariamente pari al valore massimo dato che si tratta dell'influenza su stesso. La funzione quindi include il nodo uno nel grafo aciclico dato che ha il valore massimo di Inf insieme agli archi da esso uscenti verso i nodi già presenti sul grafo aciclico. Al

passo successivo vengono aggiornati i valori di Inf dei nodi direttamente collegati al nodo radice che possono quindi influenzarlo in maniera diretta.

(b) *I nodi direttamente collegati al nodo uno sono il numero due e il tre quindi di conseguenza i valori di Inf vengono aggiornati nel modo seguente:*
$Inf(1,1) = 1;$
$Inf(2,1) = Inf(2,1) + \omega(2,1)Inf(1,1) = 0,3;$
$Inf(3,1) = Inf(3,1) + \omega(3,1)Inf(1,1) = 0,5;$
$Inf(4,1) = 0;$
$Inf(5,1) = 0;$
Viene inserito nel grafo aciclico il nodo numero tre insieme all'arco che lo congiunge al nodo radice. Verrà quindi aggiornato il valore dell'influenza del nodo numero cinque dato che attraverso il nodo tre ora può influenzare il nodo radice.

(c) *I valori di influenza vengono aggiornati:*
$Inf(1,1) = 1;$
$Inf(2,1) = 0,3;$
$Inf(3,1) = 0,5;$
$Inf(4,1) = 0;$
$Inf(5,1) = Inf(5,1) + \omega(3,5)Inf(3,1) = 0,15;$
Il nuovo nodo che andrà a far parte del grafo aciclico è il nodo due insieme al percorso che lo congiunge al nodo radice.

(d) *Il nodo due non ha archi entranti se non quello proveniente dal nodo radice, di conseguenza le influenze non subiranno alcuna variazione:*
$Inf(1,1) = 1;$
$Inf(2,1) = 0,3;$
$Inf(3,1) = 0,5;$
$Inf(4,1) = 0;$
$Inf(5,1) = Inf(5,1) + \omega(5,3)Inf(3,1) = 0,15;$
Rimane quindi da selezionare il nodo numero cinque insieme all'arco che lo congiunge al nodo tre.

(e) *Viene aggiornato il valore dell'influenza del nodo numero quattro:*
$Inf(1,1) = 1;$
$Inf(2,1) = 0,3;$
$Inf(3,1) = 0,5;$
$Inf(4,1) = Inf(4,1) + \omega(4,5)Inf(5,1) = 0,015;$
$Inf(5,1) = 0,15;$
L'algoritmo non seleziona nessun nodo dato che l'ultimo rimasto ha un valore di influenza inferiore alla soglia prefissata.

6.2.3 Coefficienti α

Questa funzione viene utilizza soltanto nella fase preparatoria della simulazione, in realtà i coefficienti α vengono aggiornati anche durante la fase di selezione degli innovatori di partenza, abbiamo deciso comunque di utilizzare funzioni differenti per le due fasi dato che dal punto di vista implementativo abbiamo notato la possibilità di semplificare il codice che calcola i coefficienti lineari per la fase preparatoria. L'interfaccia della funzione è la seguente:

```
function [Alpha_v,IncInf]=Coefficienti_alpha(DAGs,n)
```

La funzione prende in ingresso la dimensione della rete di partenza e i grafi aciclici generati precedentemente. Restituisce i coefficienti α e l'indice *IncInf*. Vediamo i passi principali svolti dalla funzione:

1. per prima cosa vengono inizializzate le strutture dati che conterranno i coefficienti e gli indici di incremento di influenza (*IncInf*).
2. Vengono quindi selezionate le reti acicliche e per ognuna di esse si effettuano le seguenti operazioni:
   - Si invoca la funzione di ordinamento topologico (topologically sort r) che dato un grafo aciclico ordina topologicamente i nodi che possono raggiungere il nodo radice.
   - Per ogni nodo della sequenza rispettando l'ordine topologico vengono calcolati i coefficienti α.
3. Noti i coefficienti dei nodi di ciascun grafo aciclico si determina per ciascun nodo della rete di partenza l'indice *IncInf* dalla somma dei coefficienti α che quel nodo ha in ogni grafo aciclico nel quale compare.

Per capire meglio facciamo un semplice esempio.

esempio 6.2. *Consideriamo il semplice esempio di grafo aciclico riportato in figura 6.3.*

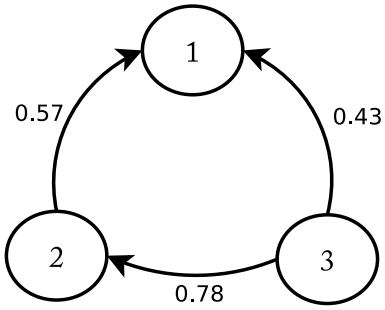


Figura 6.3: Grafo aciclico

La funzione svolge l'aggiornamento dei coefficienti α secondo i seguenti passi:

1. *Inizializza i coefficienti ponendoli tutti a zero tranne quello del nodo radice:*
   $\alpha_1 = [100]$; *il pedice indica che stiamo processando il grafo aciclico centrato sul nodo 1.*

2. *La funzione di ordinamento topologico fornisce la sequenza topologica dei nodi in grado di raggiungere il nodo radice:*
   $Sortvect = [123]$;

3. *Aggiorna il coefficiente rispettando l'ordine topologico definito in precedenza, il nodo radice non subisce questo aggiornamento:*
   $\alpha_1(2) = \omega(2,1)\alpha_1(1) = 0.57$;
   $\alpha_1(3) = \omega(3,1)\alpha_1(1) + \omega(3,2)\alpha_1(4) = 0.87$
   Quindi in definitiva i coefficienti α assumono il seguente valore:
   $\alpha_1 = [1, 0.57, 0.87]$;

6.2.4 Ciclo principale

Una volta che il modulo di controllo seleziona un nodo da inserire tra gli innovatori di partenza il blocco Loop principale si preoccupa di aggiornare: le probabilità di attivazione, gli incrementi di influenza e i coefficienti α. L'interfaccia della funzione è la seguente:

```
function [IncInf,ap_v,Alpha_v]=
Loop_principale(DAGs,Alpha_v,ap_v,IncInf,n,s,S)
```

La funzione prende in ingresso i grafi aciclici, i coefficienti α, le probabilità di attivazione, la dimensione della rete di partenza, il nuovo innovatore e il set di innovatori precedente. Le operazioni principali sono riassunte di seguito:

- La funzione seleziona i grafi aciclici che contengono il nuovo innovatore di partenza, queste reti necessariamente contengono i nodi per il quale il nuovo innovatore può esercitare un'influenza. Ricordiamo che le reti che contengono nodi appartenenti al set di innovatori calcolato precedentemente non vengono selezionate. Per ognuna di queste reti esegue le seguenti operazioni:
  - Vengono ordinati topologicamente i nodi che possono raggiungere il nuovo innovatore
  - Vengono aggiornati i coefficienti α di quei nodi che nel grafo aciclico selezionato sono in grado di raggiungere il nuovo innovatore seguendo la sequenza topologica ottenuta al passo precedente, quindi vengono aggiornati gli indici di influenza (*IncInf*) di ciascun nodo processato.

- Vengono ordinati topologicamente i nodi raggiungibili dall'innovatore di partenza.
- All'interno del grafo aciclico che si sta processando ci possono essere nodi che vengono raggiunti dal nuovo innovatore, di conseguenza questi subiranno un incremento della loro probabilità di attivazione in accordo con l'ordinamento topologico effettuato precedentemente. Anche in tal caso vengono aggiornati gli indici *IncInf* di ciascun nodo processato.

Vediamo un semplice esempio per chiarire.

esempio 6.3. *Prendiamo in considerazione una rete di partenza composta da otto nodi, l'euristica conclusa la fase di processamento produce i seguenti indici di influenza per ciascun nodo della rete:*

$IncInf = [1.14, 1.1, 2.64, 1.85, 2.19, 2.14, 2.09, 3.88]$

In base a questi dati viene selezionato come innovatore di partenza il nodo numero 8 contraddistinto dal valore più grande di influenza. A questo punto l'algoritmo seleziona le reti nel quale compare il nuovo innovatore. In queste reti i nodi raggiungibili dal nuovo innovatore incrementeranno la loro probabilità di attivazione. Per semplicità mostriamo l'aggiornamento del solo grafo aciclico in figura 6.4:

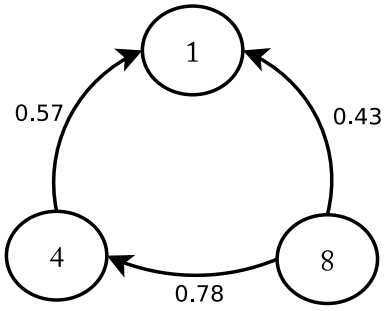


Figura 6.4: Grafo aciclico centrato sul nodo 1

- ***Aggiornamento dei coefficienti*** α***:*** *In primo luogo si impostano le condizioni iniziali del problema:*

  $\Delta\alpha_1 = [0, 0, 0, 0, 0, 0, 0, -\alpha_1(8)] = [0, 0, 0, 0, 0, 0, 0, -0.87]$;

  Dove $\Delta\alpha_1$ *rappresenta l'incremento(o decremento) dei coefficienti* α
  Viene quindi invocato l'algoritmo per l'ordinamento topologico che fornisce l'ordine col quale il nodo innovatore viene raggiunto. In tal caso si ottiene:

$Sortvect = [8]$

Di conseguenza viene aggiornato il coefficiente α del nodo 8 e il suo valore di influenza:

$\alpha_1(8) = \alpha_1(8) + \Delta\alpha_1(8) = 0;$
$IncInf(8) = IncInf(8) + \Delta\alpha_1(8)(1 - ap_1(8)) = 3.88 - 0.87 = 3.01;$

Quindi l'influenza di ciascun nodo della rete assume questa configurazione:

$IncInf = [0.71, 1.1, 2.64, 1.41, 2.19, 2.14, 2.09, 3.01]$

- ***Aggiornamento delle probabilità di attivazione:*** *Vengono impostate le condizioni iniziali:*

$\Delta ap_1 = [0, 0, 0, 0, 0, 0, 0, (1 - ap_1(8))] = [0, 0, 0, 0, 0, 0, 0, 0.428];$

Viene quindi invocata la funzione di ordinamento topologico che fornisce l'ordine con il quale i nodi raggiungibili dall'innovatore vengono contattati.

$Sortvect = [8, 1, 4]$

Quindi viene processato prima il nodo 1 poi il nodo 4, il nodo 8 viene processato successivamente:
$\Delta ap_1(1) = \Delta ap_1(8)\omega(8, 1) + \Delta ap_1(4)\omega(1, 1) = 0.428 + 0 = 0.428;$
$\Delta ap_1(4) = \Delta ap_1(8)\omega(8, 4) = 0.78;$

Quindi le probabilità di attivazione vengono aggiornate:

$ap_1(1) = ap_1(1) + \Delta ap_1(1) = 0.428;$
$ap_1(4) = ap_1(4) + \Delta ap_1(4) = 0.78;$
$ap_1(8) = ap_1(8) + \Delta ap_1(8) = 1;$

Vengono quindi aggiornati i valori di influenza:

$IncInf(1) = IncInf(1) - \alpha_1(1)\Delta ap_1(1) = 0.714;$
$IncInf(4) = IncInf(4) - \alpha_1(4)\Delta ap_1(4) = 1.41;$
$IncInf(8) = IncInf(8) - \alpha_1(8)\Delta ap_1(8) = 3.01;$

I valori di influenza assumono la seguente configurazione:

$IncInf = [0.71, 1.1, 2.64, 1.41, 2.19, 2.14, 2.09, 3.01]$

Notiamo infine che nel grafo aciclico considerato, che risulta centrato sul nodo 1, la probabilità di attivazione del nodo 8 è stato aggiornato col valore 1 dato che il nodo in esame è stato selezionato come innovatore di partenza.

6.2.5 Funzioni di ordinamento topologico

Nelle simulazioni vengono invocate due funzioni di ordinamento topologico:

1. **Topologically sort r**: Fornisce una sequenza topologica dei nodi che possono raggiungere il nodo obbiettivo.
2. **Topologically sort**: Ordina topologicamente i nodi raggiungibili da un nodo prefissato.

Le loro interfacce sono rispettivamente:

```
function [Sort_vect]=topologically_sort_r(F,s,n)
```

```
function [Sort_vect]=topologically_sort(F,s,n)
```

Entrambe prendono in ingresso une rete chiamata F, il nodo target s e la dimensione della rete di partenza n e restituiscono una sequenza denominata Sort vect. Vediamo un semplice esempio.

esempio 6.4. *Consideriamo la rete in figura 6.5 e sia* $s = \{3\}$ *il nodo sul quale costruire la sequenza topologica.*

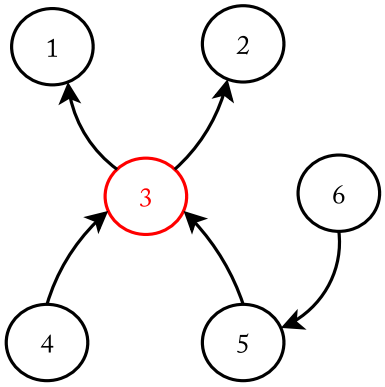


Figura 6.5: Esempio di rete

La funzione ***topologically sort r*** *produce la seguente sequenza:*

$Sortvect = [3, 4, 5, 6]$;

Mentre la funzione ***topologically sort*** *produce la seguente:*

$Sortvect = [3, 1, 2]$;

6.3 Ottimizzatore problema 1

In questa sezione analizzeremo il simulatore per il problema 1. Questo è stato implementato in linguaggio OPL per il risolutore CPLEX. Tale scelta deriva dalla grande efficienza e dalla capacità di questo risolutore di risolvere istanze di problemi anche di notevoli dimensioni. Per maggiori dettagli riguardo ad OPL e CPLEX si veda l'appendice A. In figura 6.6 vengono mostrati i blocchi che compongono il simulatore, in particolare il diagramma è stato costruito prendendo in considerazione i files che intervengono durante la simulazione e le loro relazioni. Ai blocchi colorati di verde invece non è associato nessun file dato che rappresentano un'istanza del problema in una particolare fase di risoluzione.

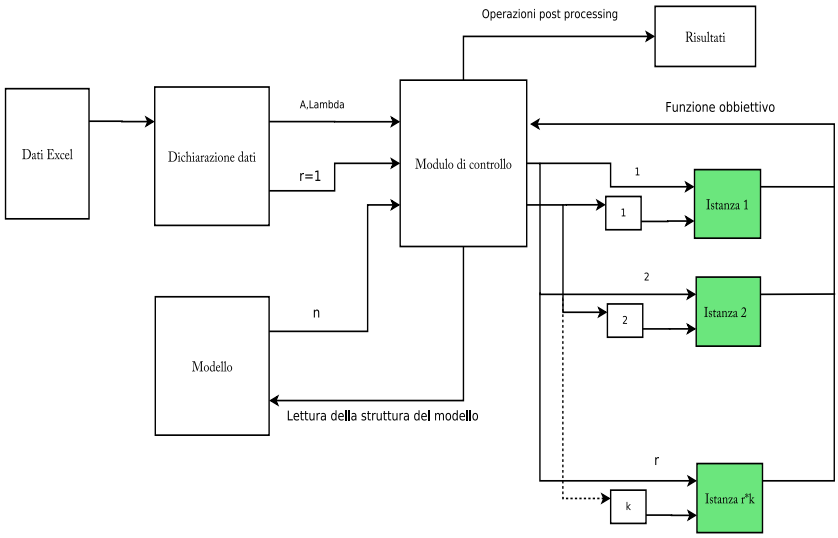


Figura 6.6: Struttura del simulatore 1

Vediamo qualitativamente il ruolo di ciascun blocco:

- **Dati excel:** Questo file contiene i dati per la costruzione dei vincolo del problema di ottimizzazione, in particolare nel foglio 1 è presente la matrice $\hat{A}^T + \Lambda$, mentre nel foglio 2 c'è la matrice delle soglie Λ.

- **Dichiarazione dati:** Questo file contiene la sintassi di dichiarazione dei dati esterni da passare all'ottimizzatore. In sostanza serve a creare un collegamento tra i dati veri e propri e il modello da istanziare. Per la definizione dei dati esterni si faccia riferimento all'appendice A.
- **Modello:** Questo file contiene la struttura del problema di programmazione lineare privo dei dati se non la dimensione della rete da processare che dichiariamo internamente al modello dato che è una costante.
- **Modulo di controllo:** Il modulo di controllo coordina tutte le operazioni da eseguire nelle varie fasi della simulazione. In particolare si occupa di:
  - Accedere al modello e ai dati per generare le istanze dei problemi di ottimizzazione da risolvere.
  - Processare la soluzione di ogni istanza attraverso il *linguaggio di script*.
- I blocchi numerati progressivamente rappresentano dei file prodotti dal modulo di controllo per costruire la giusta istanza in base al valore corrente di orizzonte temporale K.
- **Risultati:** Contiene il prodotto delle operazioni di post processing effettuate dal blocco di controllo sui risultati forniti da ogni istanza.

Questo approccio modulare fornisce numerosi vantaggi sia dal punto di vista del controllo delle operazioni svolte dal simulatore, sia in fase di debug permettendo di individuare in maniera agevole il file che eventualmente necessita di essere corretto.

6.3.1 Modulo di controllo

L'ottimizzatore 1 data la dimensione r del set di innovatori di partenza e l'orizzonte temporale K, determina il set di r innovatori che massimizzano l'influenza nella rete in K passi. Per comodità il simulatore è stato implementato in modo tale che potesse risolvere una serie di problemi di ottimizzazione ognuno contraddistinto da un differente valore della coppia r,K. La simulazione e la creazione delle istanze segue la seguente sequenza:

1. Il modulo di controllo legge il massimo valore di orizzonte temporale K e la dimensione massima del set di innovatori di partenza. In base a questi dati stabilisce quante istanze deve generare. Di seguito vediamo la sintassi:

```
main{
var KTests =10;
var rtest=10;
```

2. Quindi viene letta la struttura del modello per poter creare una nuova istanza:

```
source = new IloOplModelSource("sub_module.mod");
def = new IloOplModelDefinition(source);
theModel = new IloOplModel(def,cplex);
```

3. Per generare l'istanza deve leggere i file che contengono i dati:

```
var data = new IloOplDataSource(i+".dat");/*dichiaro i dati esterni*/
var data1= new IloOplDataSource("Matrix.dat");
theModel.addDataSource(data);/*aggiumgo i dati al modello*/
theModel.addDataSource(data1);
```

   Conclusa la lettura dei dati viene generata l'istanza del problema tramite questa sintassi:

```
theModel.generate();/*creo una nuova istanza del modello*/
```

   A seguito della generazione dell'istanza è necessario modificare il limite superiore del primo vincolo che definisce il numero di innovatori di partenza:

```
theModel.ctone.UB=j;
```

4. A questo punto è possibile attivare il risolutore che risolverà il problema di ottimizzazione:

```
cplex.solve();
```

5. Conclusa la fase risolutiva il modulo di controllo effettua eventualmente delle operazioni di post processing sui risultati. Le operazioni che abbiamo visto vengono eseguite per tutte le possibili coppie di r e k.

6.3.2 Modello

La costruzione del modello è forse la parte più importante dal quale dipende maggiormente la prestazione del simulatore. Particolare importanza riveste la costruzione dei vincoli e la dichiarazione dei dati, per quest'ultimo aspetto si veda l'appendice A. Per la costruzione dei vincoli abbiamo deciso preliminarmente di analizzare precedenti implementazioni del problema 1 ottenute con la libreria di matlab GLPK. In particolare la funzione che risolve i problemi di ottimizzazione ha la seguente interfaccia:

```
[xmin,fmin,status,extra]=glpkmex
(sense,c,A,b,ctype,lb,ub,vartype,param,lpsolver,save);
```

I dati in ingresso A e b rappresentano rispettivamente la matrice dei vincoli e i termini noti del sistema. Questo ci suggerisce che la costruzione del secondo vincolo del problema 1, da parte della libreria GLPK, viene effettuata attraverso il seguente prodotto matrice per vettore:

$$\begin{bmatrix} \hat{A}^T+\Lambda & -\Lambda & \emptyset & \cdots & \cdots & \emptyset \\ \emptyset & \hat{A}^T+\Lambda & -\Lambda & \emptyset & \cdots & \emptyset \\ \emptyset & \emptyset & \hat{A}^T+\Lambda & -\Lambda & \emptyset & \emptyset \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \emptyset & \cdots & \cdots & \emptyset & \hat{A}^T+\Lambda & -\Lambda \end{bmatrix} \begin{bmatrix} \boldsymbol{\omega}_0 \\ \boldsymbol{\omega}_1 \\ \boldsymbol{\omega}_2 \\ \vdots \\ \boldsymbol{\omega}_n \end{bmatrix} \geq \mathbf{0}_{n(k+1)}$$

Si vede chiaramente che questo approccio è decisamente poco efficiente dato che la matrice dei vincoli contiene un numero elevato di zeri. Si poteva seguire lo stesso procedimento anche in linguaggio OPL, però abbiamo deciso di implementare il vincolo cercando di ridurre al minimo la memoria. In sostanza non vengono eseguiti i prodotti tra blocchi contenenti zeri e i vettori ω_i. Questo approccio può essere riassunto nel seguente modo:

$$\begin{bmatrix} (\hat{A}^T+\Lambda)\boldsymbol{\omega}_0 - \Lambda\boldsymbol{\omega}_1 \geq \mathbf{0}_n \\ (\hat{A}^T+\Lambda)\boldsymbol{\omega}_1 - \Lambda\boldsymbol{\omega}_2 \geq \mathbf{0}_n \\ (\hat{A}^T+\Lambda)\boldsymbol{\omega}_2 - \Lambda\boldsymbol{\omega}_3 \geq \mathbf{0}_n \\ \vdots \\ (\hat{A}^T+\Lambda)\boldsymbol{\omega}_{k-1} - \Lambda\boldsymbol{\omega}_k \geq \mathbf{0}_n \end{bmatrix}$$

La sintassi di OPL per la costruzione del vincolo è la seguente:

```
forall(f in Par)
forall(i in Righe)
cttwo:
sum(p in Colonne)  A[i][p]*x[(n*(f-1)+1)+(p-1)] >=
sum(p in Colonne) Lambda[i][p]*x[(n*(f-1)+1)+(p-1+n)] ;
```

Un'altra miglioria che è possibile apportare deriva dal fatto che la matrice Λ è diagonale, per evitare di effettuare i prodotti con i valori nulli è possibile riscrivere il vincolo in modo tale che vengano effettuati i prodotti relativi ai valori in diagonale, di conseguenza la matrice Λ viene sostituita da un vettore contenente le soglie. Il vincolo viene quindi riscritto nel modo seguente:

```
forall(f in Par)
forall(i in Righe)
cttwo:
sum(p in Colonne) A[i][p]*x[(n*(f-1)+1)+(p-1)] >=
sum(p in Colonne)Lambda[i]*x[(n*(f-1)+(i+n))] ;
```

Questo secondo approccio non è stato comunque impiegato dato che non abbiamo riscontrato dei vantaggi significativi.

6.3.3 Post processamento

Non sempre i dati prodotti dalla simulazione sono pronti per una agevole lettura e valutazione, tramite il modulo di controllo è possibile costruire dei file di uscita che permettano di avere un controllo maggiore sui risultati della simulazione. Per l'ottimizzatore 1 sono stati creati dei file di estensione .dat che contengono:

- Una matrice in cui ogni riga rappresenta un vettore evoluzione.
- Una matrice cui ogni riga rappresenta la configurazione del set di innovatori di partenza.
- Una matrice cui ogni riga rappresenta la configurazione del set di utenti che al passo K adottano l'innovazione.

Per maggiori dettagli sulla sintassi delle operazioni di post processing si faccia riferimento all'appendice C. Vediamo invece una semplice simulazione in cui viene mostrato come sono organizzati i file di salvataggio.

esempio 6.5. *Consideriamo la rete in figura 6.7, l'assenza delle frecce indica che la relazione tra i nodi è bidirezionale.*

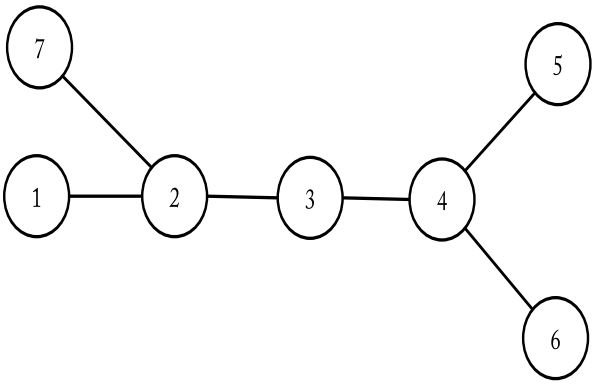


Figura 6.7: Esempio di rete

Effettuiamo la simulazione impostando i seguenti parametri:

- *K=1...3;*
- *r=1...3;*

Cosideriamo il file che contiene il vettore delle evoluzioni per K=2:

```
1 2 3 4 5 6 7  1 2 3 4 5 6 7  1 2 3 4 5 6 7
0 1 0 0 0 0 0  1 1 0 0 0 0 1  1 1 0 0 0 0 1
0 1 0 1 0 0 0  1 1 1 1 1 1 1  1 1 1 1 1 1 1
0 1 0 1 1 0 0  1 1 1 1 1 1 1  1 1 1 1 1 1 1
```

La prima riga rappresenta il vettore evoluzione per r=1, il simulatore sceglie il nodo numero 2 come innovatore di partenza. Questo produce la diffusione dell'innovazione sui nodi 1 e 7 come si vede negli ultimi sette elementi. Come detto precedentemente ad ogni riga è associato un diverso valore di r, infatti analizzando i primi sette elementi della seconda riga si nota che vengono scelti due innovatori dato che la simulazione è stata condotta col valore r=2. Per comodità in fase di post processing sono stati salvati anche i primi e gli ultimi n elementi di ciascun vettore evoluzione in modo da poter visualizzare agevolmente lo stato iniziale e finale della rete.

6.4 Ottimizzatore problema 2

Vedremo ora l'ottimizzatore per il problema 2, anche questo è stato implementato in linguaggio OPL, analogamente a quanto fatto nella sezione precedente mostriamo i blocchi principali e il loro funzionamento.
In figura 6.8 viene mostrata la struttura generale del simulatore e i blocchi che lo compongono.

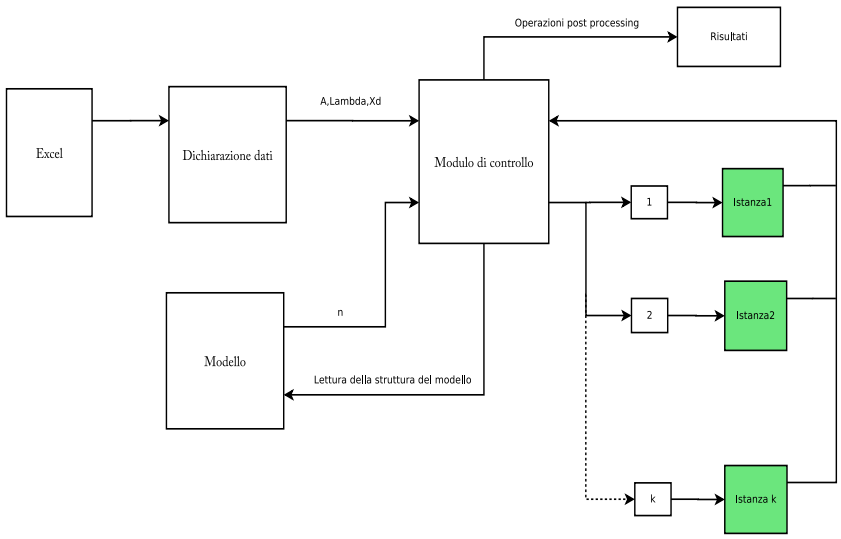


Figura 6.8: Struttura del simulatore 2

Ogni blocco svolge la seguente funzione:

- **Dati excel:** Il file contiene tutti i dati per far partire le simulazioni, in particolare il foglio excel è strutturato nel modo seguente:
  - Foglio 1: Contiene la matrice $\hat{A}^T + \Lambda$.
  - Foglio 2: Contiene la matrice Λ.

  - Foglio 5: Contiene il vettore Xd nel quale vengono specificati gli utenti obbiettivo. Eventualmente i fogli 3,4,7 contengono gli stessi dato ottenuti con l'algoritmo di riduzione della rete.

- **Dichiarazione dati:** Contiene la sintassi per porre in comunicazione i dati contenuti nel file excel col modulo di controllo. In particolare si specificano i fogli del file al quale accedere e il range nel quale sono posizionati i dati in ciascun foglio.

- **Modello:** Contiene la struttura del problema di programmazione e può contenere eventualmente dei dati, ad esempio delle costanti, dichiarati internamente. Per risolvere la versione rilassata è sufficiente accedere a questo file e modificare il tipo che contraddistingue il vettore di decisione.

- **Modulo di controllo:** Il modulo di controllo contiene una sintassi specifica che consente di eseguire istanze multiple dello stesso problema di programmazione. In particolare svolge le seguenti funzioni:

  - Genera le istanze del problema seguendo il flusso di esecuzione specificato.
  - Processa eventualmente le soluzioni fornite dalle istanze e salva i dati in file appositi.

- I blocchi numerati progressivamente contengono ognuno il valore della variabile K per l'istanza corrente.

- **Risultati:** Il blocco sta a rappresentare il file o l'insieme di file prodotti in fase di postprocessing .

6.4.1 Modulo di controllo

Il modulo di controllo analogamente a quello visto per l'ottimizzatore 1 gestisce il flusso di esecuzione della simulazione. Schematicamente il modulo di controllo può essere suddiviso in due parti:

- La prima parte del codice serve sostanzialmente a generare le istanze del problema di ottimizzazione.

- La seconda parte gestisce la soluzione manipolandola eventualmente per rendere più agevole la sua interpretazione.

Sostanzialmente la sintassi della prima fase è assolutamente identica a quella vista per il simulatore 1, le uniche differenze risiedono nel fatto che in tal caso la simulazione si svolge soltanto al variare del parametro K, mentre per l'ottimizzatore 1 il flusso di esecuzione dipendeva sia dal parametro K che dal parametro r. Per maggiori dettagli sulla sintassi si faccia riferimento all'appendice C.

6.4.2 Modello

L'approccio seguito per la costruzione del modello è lo stesso seguito per l'ottimizatore 1. Vediamo comunque alcune parti del codice per capire anche il concetto di dato esterno e interno al modello. Il problema di ottimizzazione si riassume nelle seguenti righe di codice:

```
dvar   boolean  x [ Cols ] ;
minimize
sum ( j  in  Cols )   INK[ j ]*x [ j ] ;
subject  to
{
forall(l in Last_elm)
ctone:
x[l]>=Xd[l-n*K];
forall(f in Par)
forall(i in Righe)
cttwo:
sum(p in Colonne) A[i][p]*x[(n*(f-1)+1)+(p-1)] >=
sum(p in Colonne) Lambda[i][p]*x[(n*(f-1)+1)+(p-1+n)] ;
}
```

Il vettore x rappresenta il vettore di decisione che come vediamo è di tipo booleano. per risolvere la versione rilassata del problema di programmazione è sufficiente dichiarare la variabile di decisione nel modo seguente:

```
dvar   float+  x [ Cols ] ;
```

Nella seconda riga di codice viene dichiarato il problema di ottimizzazione che in questo caso è una minimizzazione. Il vettore INK è stato dichiarato internamente al modello, infatti prima della definizione del problema di programmazione questo viene generato attraverso la seguente sintassi:

```
execute  {
for(var  i  in  Cols)
if(i<=n)
INK[i] =1;
else
INK[i]=0;
}
```

I blocchi di questo tipo inseriti prima della dichiarazione del problema di ottimizzazione vengono detti di preprocessamento, mentre quelli inseriti dopo vengono interpretati da OPL come operazioni di postprocessamento.

6.5 Set coesivo massimale

Vedremo ora il simulatore per la determinazione del set coesivo massimale. Analogamente agli altri simulatori è stato impiegato un approccio modulare. D'ora in poi faremo riferimento al simulatore costruito per il problema di ottimizzazione basato sulla risoluzione di una serie di problemi di programmazione lineare continui. Vedremo che il simulatore basato sulla risoluzione di un problema di ottimizzazione discreto è del tutto simile eccetto che per la dichiarazione delle variabili di decisione. In figura 6.9 vengono mostrati i blocchi che compongono il simulatore, senza soffermarci sul significato di ciascun blocco che risulta essere analogo a quello visto per gli altri simulatori, vediamo come questa struttura differisce dalle precedenti per quanto riguarda la definizione del modello del problema di ottimizzazione.

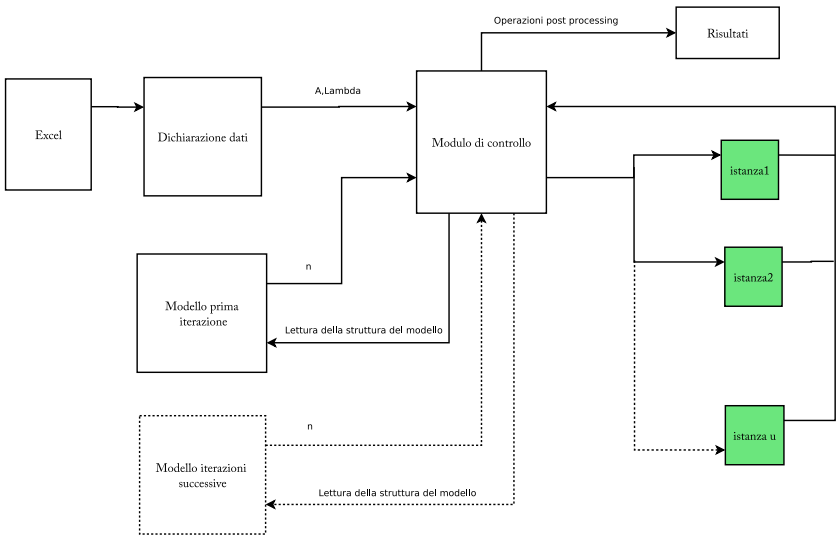


Figura 6.9: Struttura del simulatore 3

Infatti per motivi di implementazione sono stati definiti due modelli: uno che risolve il primo problema di programmazione lineare e l'altro che viene richiamato per le iterazioni successive alla prima. Il secondo blocco è stato indicato con tratto discontinuo per indicare che non necessariamente viene utilizzato in ogni simulazione in quanto se la soluzione della prima istanza non presenta variabili frazionarie allora la simulazione si arresta restituendo questa soluzione. Per quanto riguarda la determinazione del set coesivo massimale basato su un problema di ottimizzazione discreto è sufficiente utilizzare il modello per la prima iterazione modificando il tipo della variabile di decisione con sintassi analoga a quella vista per l'ottimizzatore 2.

6.5.1 Modulo di controllo

Vediamo ora il modulo di controllo, in particolare rispetto a quelli visti in precedenza richiama due modelli differenti. Il primo passo consiste nel generare la prima istanza:

```
source = new IloOplModelSource("Sub_Model_1.mod");
def = new IloOplModelDefinition(source);
theModel = new IloOplModel(def,cplex);
.
.
.
theModel.addDataSource(data);/*aggiumgo i dati al modello*/
theModel.addDataSource(data1);
theModel.generate();
cplex.solve(); /*SOLVE THE FIRST PROBLEM*/
```

Come si vede dalla prima riga di codice viene richiamato il modello per la prima iterazione. Successivamente se il primo problema ha fornito una soluzione con variabili frazionarie si risolvono una serie di problemi continui finché la soluzione tovata ha tutte le variabili intere. Per fare questo si utilizza la seguente sintassi:

```
source = new IloOplModelSource("Sub_Model_K.mod");
def = new IloOplModelDefinition(source);
theModel = new IloOplModel(def,cplex);
.
.
.
theModel.addDataSource(data3);/*aggiungo i dati al modello*/
theModel.addDataSource(data4);
theModel.generate();
cplex.solve(); /*SOLVE THE PROBLEM*/
```

Si vede chiaramente come venga richiamato il modello per le iterazioni successive contenuto nel file Sub Model k.mod.

6.5.2 Modelli

Dal punto di vista della dichiarazione del problema di programmazione il modello per la prima iterazione e quello per le iterazione successive sono assolutamente identici, ciò che fa differire i due moduli sono i dati in ingresso. Questo ci ha portato ha utilizzare due modelli differenti per i due distinti casi senza perdita di prestazione e da un certo punto di vista migliorando la leggibilità del codice. Abbiamo visto nel capitolo sulla diffusione dell'innovazione nei social networks che il limite superiore per il primo vincolo era:

$x \leq 1 - y^{(0)}$

dove x è la variabile di decisione e $y^{(0)}$ è il vettore che contiene il set di innovatori di partenza al primo passo (k=0). Risolto questo problema di programmazione il limite superiore del primo vincolo per il problema di programmazione successivo è dato da:

$x \leq 1 - y^{(k)}$

Dove $y^{(k)} = \lceil 1 - x^{(k-1)} \rceil$ cioè è legato alla soluzione al passo precedente. Per comodità il modello per le iterazioni successive genera il dato per il vincolo internamente con la seguente sintassi:

```
float b[Dim];
/*Genero il vettore identit'a*/
execute  {
for(var  i  in  Dim)
I[i] =1;
}

float C[p in Dim]=ceil(I[p]-y[p]);

execute  {
for(var  f  in  Dim)
b[f]=I[f]-C[f];/*genero il limite superiore del primo vincolo*/
}
```

Il vettore b[f] generato internamente rappresenta il limite superiore peri il primo vincolo, in maniera alternativa si poteva costruire il simulatore mantenendo un solo modello del problema di programmazione. Per modificare il primo vincolo in quel caso si doveva ricorrere all'accesso da parte del modulo di controllo sul limite superiore del vincolo attraverso un'opportuna sintassi. Questo approccio però non sempre è comodo e spesso risulta essere un pò lento per istanze di grandi dimensioni, perciò è stato deciso di costruire due modelli in modo da generare i dati internamente che spesso porta a vantaggi prestazionali, si veda a tal proposito l'appendice A.

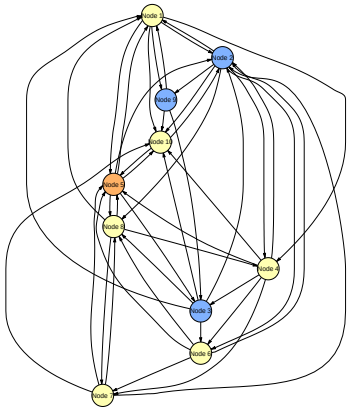
6.6 Algoritmo di riduzione della rete

Nella sezione 6.4 abbiamo visto l'ottimizzatore 2, in particolare per questo simulatore è stato costruito un semplice codice in Matlab che riduce la dimensione della rete di partenza in base alla conoscenza dei nodi obbiettivo e all'orizzonte temporale scelto per la simulazione. I passaggi principali svolti dall'algoritmo sono i seguenti:

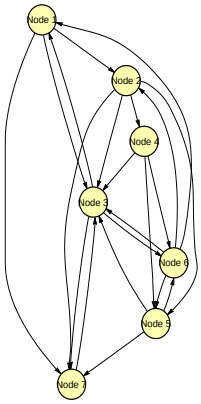
- Genera le matrici $\hat{A}^T + \Lambda$, Λ e Xd della rete di partenza. Salva quindi i dati rispettivamente nei fogli 1,2,5 del file excel.
- Genera le matrici $\hat{A}_r^T + \Lambda_r$, Λ_r e Xd_r della rete ridotta Salva quindi i dati rispettivamente nei fogli 3,4,7 del file excel.
- Visualizza il grafo della rete di partenza e della rete ridotta.

Vediamo un semplice esempio:

esempio 6.6. *Consideriamo la rete di partenza nella figura 6.10(a). In rosso viene identificato il nodo o i nodi target. Impostiamo come orizzonte temporale K=1, di conseguenza tutti i nodi che distano più di K passi verranno scartati. I nodi scartati vengono identificati dal colore blu, in seguito alla riduzione della rete di partenza si ottiene il grafo in figura 6.10(b).*



(a) Rete di partenza



(b) Rete ridotta

Figura 6.10: Rete di partenza e rete ridotta

Capitolo 7

Simulazioni

7.1 Introduzione

Nel seguente capitolo analizzeremo i risultati sui tre simulatori implementati in OPL. Per ognuno di essi analizzeremo i tempi di calcolo richiesti per risolverli. Vedremo quindi i risultati ottenuti attraverso i metodi di ottimizzazione approssimati cercando di dare una valutazione dal punto di vista delle prestazioni. Particolare importanza viene data alle simulazioni sul primo problema 1 del quale vedremo sostanzialmente tre approcci risolutivi. Nella parte finale delle simulazioni ci si occupa del problema 2 in cui valuteremo l'impatto sul tempo computazionale della riduzione delle rete, infine faremo dei semplici esempi per l'ottimizzatore dedicato al problema 3.

7.2 Complessità computazionale del problema 1

Abbiamo più volte rimarcato che il problema 1 risulta essere di tipo
NP hard ossia il tempo computazionale non dipende in maniera lineare dalla dimensione dei dati da processare, questo comporta che il problema risulta assolutamente non trattabile per istanze reali per il quale la rete da processare può contenere milioni di nodi e archi. La complessità del problema dipende da:

- Il numero di nodi n della rete
- Dalla dimensione del set di innovatori di partenza r

Il numero di possibili combinazioni di r innovatori su una rete di n elementi è data dalla seguente relazione:

$$\left(\begin{array}{c} n \\ r \end{array} \right) = \frac{n!}{r!(n-r)!}$$

Per capire l'impatto che può avere sul tempo computazionale possiamo graficare l'andamento del numero di combinazioni al variare del numero di innovatori r su una rete di dimensione n.

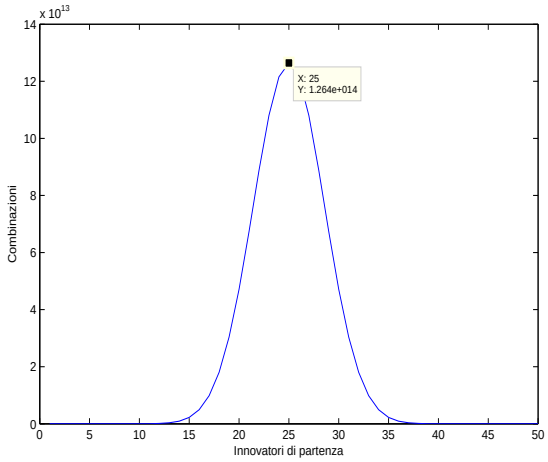


Figura 7.1: Possibili combinazioni di set di innovatori

La figura 7.1 pone in evidenza come per una rete con soli 50 nodi il numero di possibili soluzioni può essere dell'ordine di 10^{14}. Un'altro dato che influenza le prestazioni è l'orizzonte temporale K, infatti la dimensione della funzione obbiettivo dipende pesantemente da questo fattore oltre che dalla dimensione della rete stessa. La dimensione della funzione obbiettivo è data dalla seguente relazione:

$$n(K+1)$$

Quindi se per esempio prendessimo in considerazione una rete con $n = 1000$ e un orizzonte temporale $K = 20$ si dovrebbe risolvere un problema di ottimizzazione discreto la cui funzione obbiettivo è composta da 21000 variabili. Per rimarcare questi concetti vediamo due simulazioni, nella prima vedremo come varia il tempo computazionale al variare dell'orrizzonte temporale mentre nella seconda simulazione fisseremo l'orizzonte temporale e varieremo il numero di innovatori di partenza. Consideriamo una rete composta da 50 nodi, imponiamo che il set di innovatori di partenza abbia una cardinalità pari a r=2. Risolviamo quindi il problema di ottimizzazione variando l'orizzonte temporale K. In figura 7.2 viene mostrato l'andamento del tempo computazionale rispetto al valore di K.

Si nota chiaramente che il tempo per risolvere il problema di ottimizzazione cresca notevolmente al variare dell'orizzonte temporale.

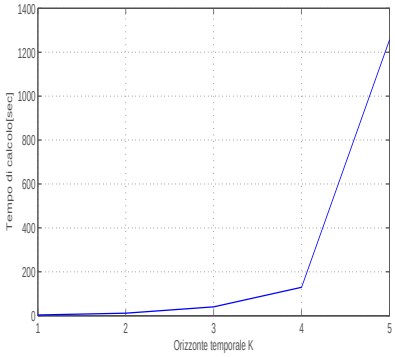


Figura 7.2: Tempo computazionale al variare di K

Il comportamento mostrato al variare di K si presenta anche al variare del numero di innovatori di partenza. Consideriamo a tal proposito la seguente simulazione:

Consideriamo la stessa rete della precedente simulazione e fissiamo l'orizzonte temporale al valore K=2. Risolviamo quindi il problema di ottimizzazione al variare del numero di innovatori di partenza. L'andamento dei tempi di calcolo è mostrato in figura 7.3.

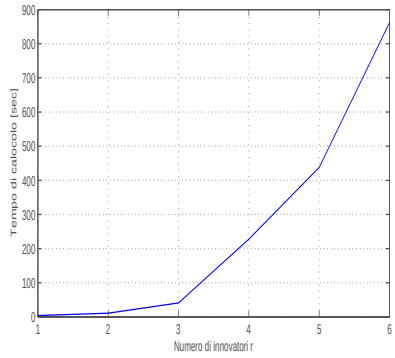


Figura 7.3: Tempo computazionale al variare di r

Anche in tal caso il tempo computazionale cresce sensibilmente e in modo non lineare rispetto alla variabile r. Quindi riassumendo il problema principale è il tempo computazionale, nonché le risorse in termini di memoria. La soluzione migliore sarebbe quindi quella che consente di slegare il tempo computazionale dalla dimensione dei dati in ingresso anche a discapito dell'ottimalità della soluzione.

7.3 Test preliminare problema 1

In questa sezione mostreremo un semplice test condotto su dieci reti così configurate:

- Numero di utenti n=30
- Orizzonte temporale K=1...10
- Set di innovatori di partenza r=1...10

La scelta di effettuare un test su un gruppo di utenti così piccolo nasce dalla necessità di ricavare la soluzione ottima da poter confrontare con quella ottenuta modificando il comportamento dell'algoritmo branch and cut. Nelle simulazioni effettuate con reti di grandi dimensioni abbiamo riscontrato problemi con l'occupazione di memoria e dei tempi di calcolo molto elevati. Le reti vengono generate in modo random tramite un codice scritto in Matlab che salva i dati in formato Excel. Questi dati vengono poi passati al simulatore CPLEX che risolve il problema di ottimizzazione vincolato. In

	Tempo di calcolo [sec]
Rete 1	743
Rete 2	291
Rete 3	428
Rete 4	2123
Rete 5	516
Rete 6	1175
Rete 7	190
Rete 8	956
Rete 9	848
Rete 10	423
Media	796

Tabella 7.1: Tempi di calcolo simulatore 1

tabella 7.1 vengono mostrati i tempi di calcolo per ciascuna rete. Notiamo immediatamente che i tempi di calcolo sono piuttosto variabili nonostante le reti siano di dimensioni uguali e siano state testate con le stesse condizioni. Questo deriva dallo specifico algoritmo che CPLEX impiega per risolvere i problemi di ottimizzazione discreti, infatti il risolutore non testa tutte le possibili soluzioni dello spazio delle soluzioni ammissibili, piuttosto costruisce un serie di versioni rilassate del problema di partenza ognuna con una differente regione delle soluzioni ammissibili costruita attraverso degli algoritmi di taglio. Ogni specifica rete può presentare per il risolutore una

differente difficoltà nell'isolare la regione delle soluzioni ammissibili che rispetta il vincolo sulla funzione obbiettivo e quello sulle variabili e questo spiega come i tempi di calcolo possano variare notevolmente.

In figura 7.4 viene mostrato l'andamento del numero di utenti che adottano l'innovazione al variare dell'orizzonte temporale K e al numero di innovatori di partenza. Il grafico è stato ottenuto dalla media sulle dieci reti testate.

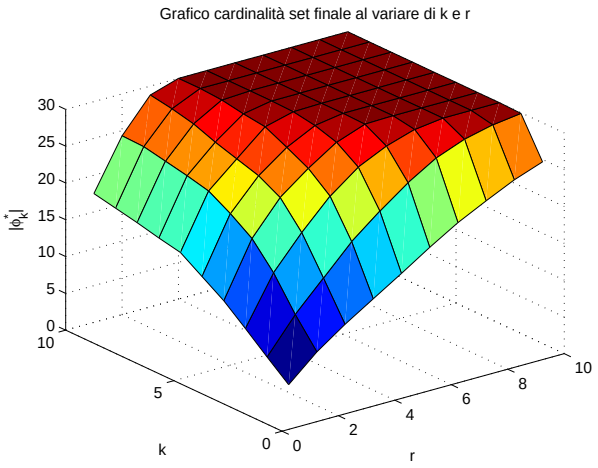


Figura 7.4: Cardinalità del set di innovatori finali al variare di K e r

Come mostrato in [23] aumentando il numero di innovatori di partenza il numero di utenti che alla fine del processo di diffusione adottano l'innovazione cresce. Questo fenomeno si riscontra anche all'aumentare dell'orizzonte temporale anche se in maniera molto più lenta.

7.4 Test di taratura del risolutore CPLEX

Nella sezione precedente abbiamo accennato come il risolutore CPLEX effettua la risoluzione dei problemi di programmazione lineari interi e misti. Per garantire che venga effettivamente trovata la soluzione ottima si può tarare il risolutore in modo tale che questo costruisca un numero più o meno elevato di versioni rilassate del problema di partenza. Ci siamo concentrati in particolare sui limiti di funzionamento dell'algoritmo *Branch* & cut del risolutore CPLEX, in particolare è possibile tarare:

- Il numero di tagli di Gomory che l'algoritmo applicca durante la risoluzione.
- Il numero di sottoproblemi rilassati generati da un sottoproblema radice.

- Il limite globale di sottoproblemi generati.
- Limite sulla memoria assegnata all'albero costituito dai sottoproblemi rilassati.

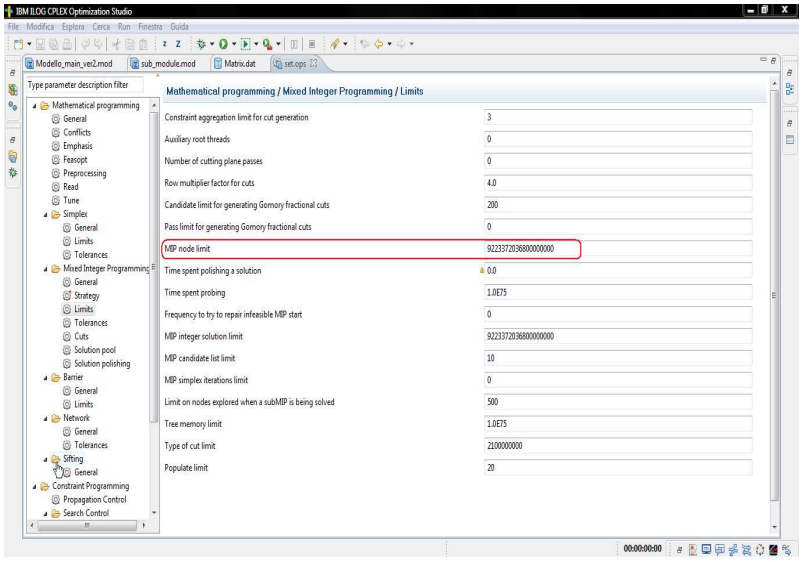
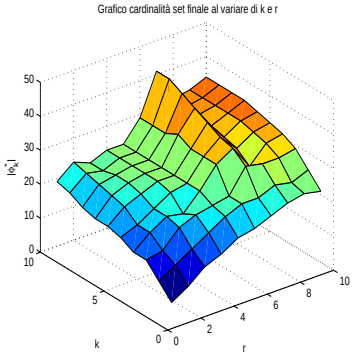


Figura 7.5: Opzioni taratura del risolutore

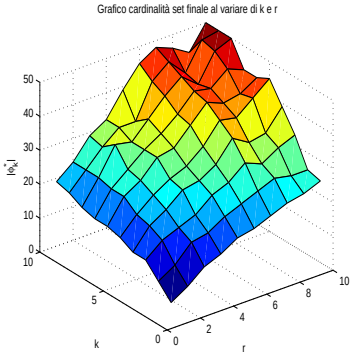
Sono stati effettuati dei test andando a modificare il numero di nodi che globalmente vengono generati per risolvere il problema binario di partenza. In figura 7.5 vengono mostrate le opzioni possibili che consentono di tarare il risolutore. Il test preliminare consiste nel testare una rete con differenti regolazioni partendo da valori molto bassi di nodi globalmente generati per risolvere il problema binario fino a raggiungere dei valori limite oltre il quale la memoria del nostro calcolatore risulta insufficiente. In questa sezione mostreremo un semplice test condotto su una rete con le seguenti caratteristiche e impostazioni di CPLEX :

- Numero di utenti n=50
- Orizzonte temporale K=1...10
- Set di innovatori di partenza r=1...10
- limite globale dei sottoproblemi rilassati:9,100,2000,4000,40000,160000.

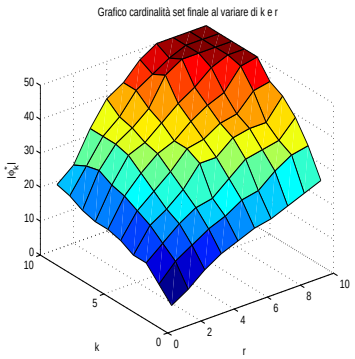
La figura 7.6 evidenzia qualitativamente come la soluzione divenga via via migliore all'aumentare del numero di sottoproblemi rilassati risolti. Questo ci suggerisce che tarando opportunamente il risolutore si possa ottenere un risultato, con una approssimazione ragionevole, del problema di ottimizzazione discreto con dei tempi di calcolo non eccessivamente elevati.



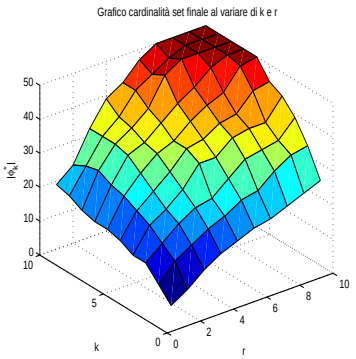
(a) Mip limit 9



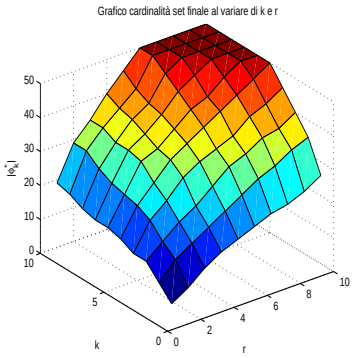
(b) Mip limit 100



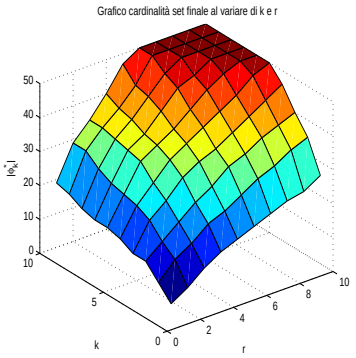
(c) Mip limit 2000



(d) Mip limit 4000



(e) Mip limit 40000



(f) Mip limit 160000

Figura 7.6: Utenti finali al variare della tarartura del risolutore

	Tempo di calcolo [sec]
Mip limit 9	226
Mip limit 100	246
Mip limit 2000	395
Mip limit 4000	589
Mip limt 40000	3631
Mip limit 160000	18009

Tabella 7.2: Tempi di calcolo del test di taratura del risolutore

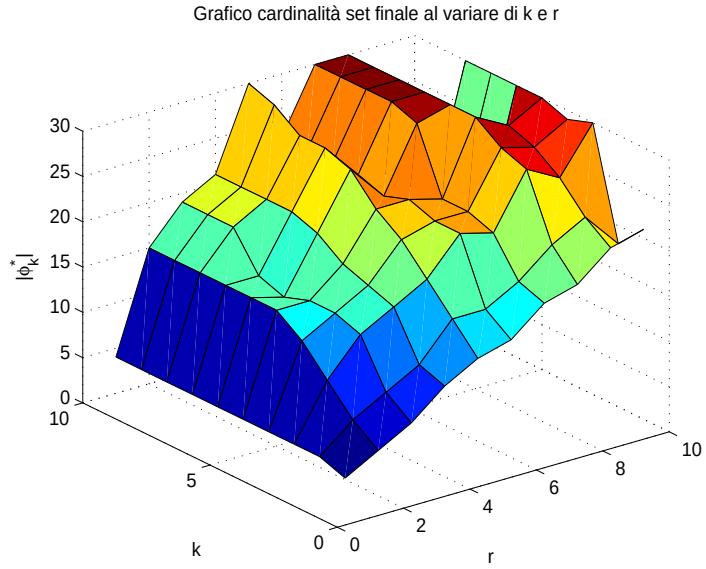
Questa ipotesi viene avvalorata dai tempi di calcolo in tabella 7.2, infatti il tempo di calcolo ottenuto risolvendo 160000 sottoproblemi rilassati è decisamente alto se paragonato con quello ottenuto risolvendone 4000 inoltre si nota come quest'ultima soluzione si discosti poco da quella con maggiore precisione. Vedremo nella sezione successiva la validità di tale approccio.

7.4.1 Valutazione della soluzione subottima

Nella sezione 7.3 abbiamo mostrato un test preliminare sul problema 1 nel quale sono state ricavate le soluzioni ottime su dieci reti di pari caratteristiche al variare di K ed r. In relazione alla taratura del risolutore CPLEX è stata ripetuta questa simulazione in modo da valutare l'approccio mostrato nella precedente sezione. In particolare è stato tarato il risolutore in modo tale da ridurre al minimo i tempi di calcolo per poi verificare la bontà della soluzione fornita. In quest'ottica è stato impostato come numero limite di sottoproblemi rilassati un valore pari a 80. Tale valore è decisamente basso se paragonato a quello di default, quindi è lecito attendersi che la soluzione fornita dall'ottimizzatore possa discostarsi sensibilmente dalla soluzione ottima. Un esempio di questa ipotesi è mostrata in figura 7.7. Il grafico in particolare si riferisce alla rete 6. È evidente che per alcune combinazioni di r e K la soluzione subottima sia sensibilmente peggiore rispetto a quella ottima.

Vogliamo ora valutare in maniera puntuale la prestazione derivante dalla taratura del risolutore. A tal proposito valutiamo per ciascuna rete e per ogni combinazione di r e K il numero di utenti che adottano l'innovazione sia per la soluzione ottima che per quella subottima, quindi effettueremo la media sulle dieci reti testate. Dato che il test della sezione 7.3 è stato

(a) Soluzione ottima



(b) Soluzione subottima

Figura 7.7: Rete 6 soluzione ottima e subottima

condotto su dieci valori differenti di r e K, otteniamo complessivamente cento differenti combinazioni per ciascuna rete che verranno mediate.

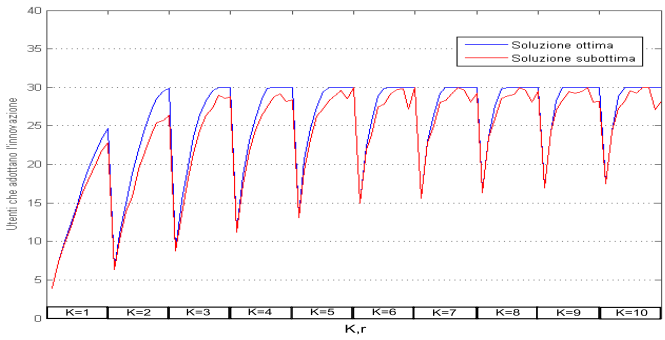


Figura 7.8: Confronto tra soluzione ottima e subottima

In figura 7.8 viene mostrato l'esito di questa valutazione, in particolare in ascissa viene indicato per ogni step un valore specifico di orizzonte temporale K. All'interno di ciascuno di essi il numero di innovatori di partenza r varia da 1 a 10, infatti è facile verificare che all'aumentare di questo fattore aumenta il numero di utenti che adottano l'innovazione. Dal punto di vista qualitativo si nota chiaramente che mediamente la soluzione subottima non si discosta eccessivamente da quella ottima. Per quantificare meglio queste differenze si faccia riferimento alla figura 7.9. Il grafico pone in evidenza la differenza, in termini di utenti che adottano l'innovazione, tra soluzione ottima e subottima. Nel caso peggiore riscontriamo che la soluzione ottima si discosta da quella subottima di 3.7 unità, ossia la soluzione ottima consente a 3.7 utenti in più di adottare l'innovazione.

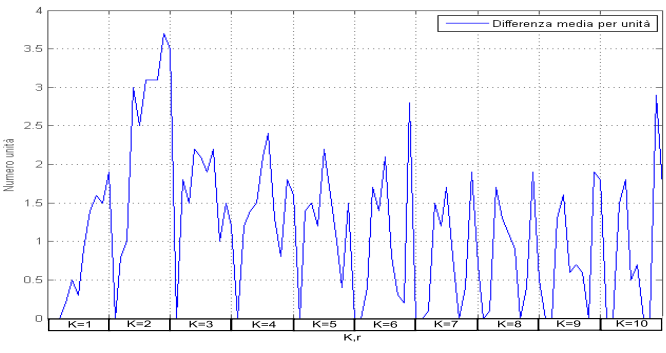


Figura 7.9: Differenza in termini di unità

In tabella 7.3 vengono quantificate le differenze in termini percentuali e di unità tra le due soluzioni.

	errore %	Unità
Minimo	0	0
Massimo	12.3	3.7

Tabella 7.3: Errore di approssimazione del risolutore

È stato riscontrato che nel 20% dei casi la soluzione subottima coincide con quella ottima. Questo non è certo un risultato deludente se posto in relazione con i tempi di calcolo del risolutore in seguito alla sua taratura.

	Tempo di calcolo [sec]	
	Soluzione ottima	Soluzione subottima
Rete 1	743	241
Rete 2	291	227
Rete 3	428	227
Rete 4	2123	217
Rete 5	516	235
Rete 6	1175	233
Rete 7	190	232
Rete 8	956	228
Rete 9	848	231
Rete 10	423	238
Media	796	231

Tabella 7.4: Tempi di calcolo soluzione ottima e subottima

Consideriamo a tal proposito i dati forniti dalla tabella 7.4. Da questa ricaviamo due informazioni, in primo luogo possiamo dire che mediamente la taratura del risolutore porta ad un riduzione dei tempi di calcolo pari a circa il 70%

Notiamo inoltre che il tempo computazionale è contraddistinto da una minore variabilità rispetto a quella riscontrata calcolando la soluzione ottima. Questo comportamento è ovviamente dovuto al limite imposto al numero di sottoproblemi rilassati dell'algoritmo branch and cut. Rispetto ai tempi di calcolo abbiamo voluto indagare un pò più a fondo per verificarne la dipendenza rispetto ai parametri r e K. In quest'ottica è stata effettuata una simulazione su una rete composta da 50 nodi impostando il limite del risolutore a 4000 sottoproblemi. In figura 7.10 viene mostrato l'andamento del tempo computazionale rispetto a K, mentre la figura 7.11 pone in relazione il tempo di calcolo col parametro r. Per quanto riguarda la prima simulazione è stato impostato r=2 mentre per la seconda è stato imposto un orizzonte temporale K=2. In entrambi i casi viene posto in evidenza come

non vi sia più una crescita sensibile dei tempi di calcolo al variare di K ed r. Questa caratteristica è decisamente interessante perché consente di risolvere problemi di ottimizzazione di maggiore dimensione e complessità rispetto a quelli che potevamo trattare lasciando le impostazioni di default.

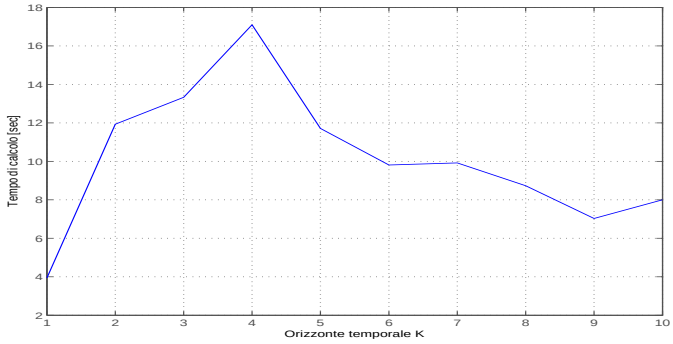


Figura 7.10: Tempo di calcolo al variare di K

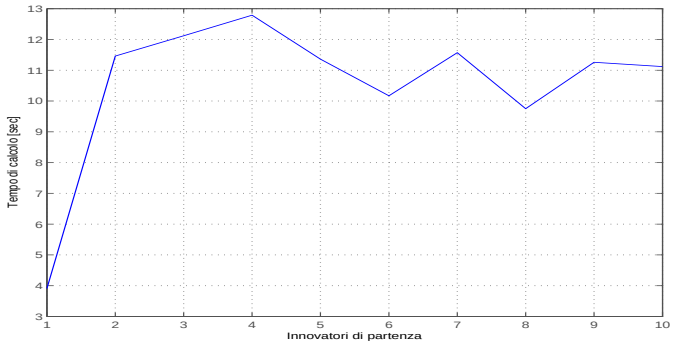


Figura 7.11: Tempo di calcolo al variare di r

7.4.2 Test sull'orizzonte temporale

Nel capitolo 4 abbiamo visto che il problema di massimizzazione dell'influenza in tempo finito rappresenta una generalizzazione del problema classico della massimizzazione dell'influenza. Ciò significa che se valutiamo la soluzione fornita dall'ottimizzatore 1 per valori grandi del parametro K, questa coinciderà con la soluzione che fornirebbe il problema classico. In quest'ottica è stata effettuato un semplice test per valutare questo aspetto visto nella teoria.

Consideriamo la rete in figura 7.12 composta da 30 nodi. Idealmente dovremmo risolvere il problema di ottimizzazione imponendo un valore di

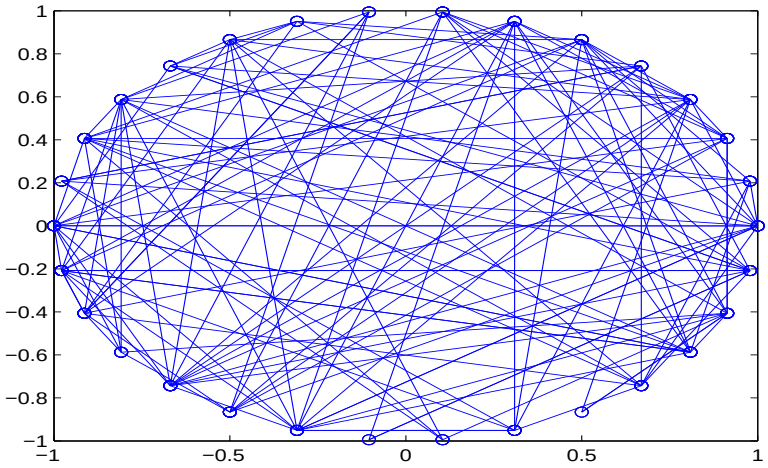


Figura 7.12: Grafo con n=30

K=∞, in realtà è sufficiente imporre che l'orizzonte temporale sia ragionevolmente elevato da considerare concluso il processo di diffusione. In tal senso il test è stato effettuato per un valore di K pari a 5000 che risulta più che sufficiente per una rete di così ridotte dimensioni. La scelta di testare una rete così piccola nasce dal fatto che per valori elevati di K il tempo computazionale e le risorse in termini di memoria occupata per ottenere la soluzione ottima crescono notevolmente come visto nella sezione 7.1. Abbiamo infine risolto il problema di ottimizzazione richiedendo un set di innovatori di partenza composto da 2 unità. L'ottimizzatore 1 fornisce come innovatori di partenza il seguente set:

$$S = \{5, 28\}$$

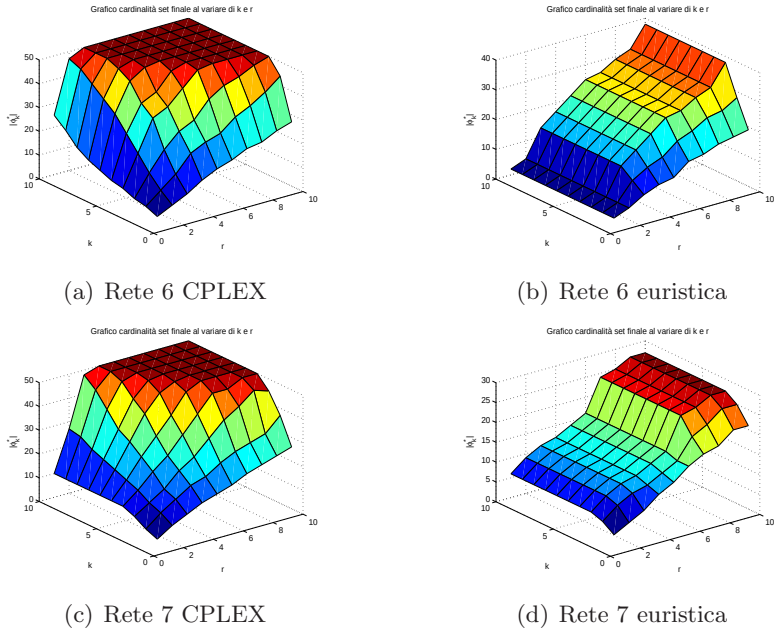
Per un'orizzonte temporale così esteso tutti i nodi della rete hanno adottato l'innovazione. Ora che conosciamo il set di innovatori di partenza possiamo valutare il set di utenti finali che adottano l'innovazione per il problema classico della massimizzazione dell'influenza. Per ottenere questa informazione è stato utilizzato l'ottimizzatore 3 per la determinazione del set coesivo massimale. Ricordiamo che noto il set coesivo massimale del complemento degli innovatori di partenza è possibile determinare lo stato finale della rete in termini di diffusione dell'innovazione. L'ottimizzatore ha fornito come set coesivo massimale l'insieme vuoto $\mathcal{M} = \emptyset$, ciò significa che tutti gli utenti adottano l'innovazione come mostrato anche dall'ottimizzatore 1 per K=5000.

7.5 Valutazione dell'euristica

In alternativa alla taratura del risolutore CPLEX è stata sviluppata un estensione dell'euristica proposta in [11]. In particolare abbiamo valutato se questa euristica rappresentasse una valida alternativa rispetto alla strategia mostrata nella sezione precedente. Il test in particolare è stato condotto ponendo in relazione i risultati ottenuti dall'euristica e dal risolutore CPLEX tarandolo con un numero di sottoproblemi pari a 4000. Analogamente alla simulazione della precedente sezione, valuteremo la prestazione media dei due approcci. In maniera specifica è stato effettuato una simulazione su dieci reti con le seguenti caratteristiche:

- Numero di utenti n=50
- Orizzonte temporale K=1...10
- Set di innovatori di partenza r=1...10

Per tutte le simulazioni è stato impostato $\theta = \frac{1}{80}$. In figura 7.13 vediamo un esempio di confronto tra le due strategie in relazione al numero di utenti finali che adottano l'innovazione al variare di K ed r per la rete 6 e 7.



(a) Rete 6 CPLEX

(b) Rete 6 euristica

(c) Rete 7 CPLEX

(d) Rete 7 euristica

Figura 7.13: Confronto tra euristica e soluzione subottima di CPLEX

Risulta già evidente come la soluzione del risolutore CPLEX sia migliore rispetto a quella ottenuta con l'euristica. Vediamo comunque di quantificare queste differenze di prestazione e di commentare ulteriormente la figura 7.13,

notiamo infatti come la soluzione fornita dall'euristica non cambia superato un certo valore di orizzonte temporale, questo comportamento deriva dal fatto che per un certo valore di K non vengono generati grafi aciclici differenti rispetto al passo K precedente. Ciò è del tutto normale se consideriamo che la rete è piuttosto piccola, ricordiamo che la costruzione dei grafi aciclici prevede che questi siano costituiti da nodi che distano K passi dalla radice, questo comporta che per un certo valore di K il grafo aciclico non possa crescere ulteriormente e che la soluzione non possa cambiare.

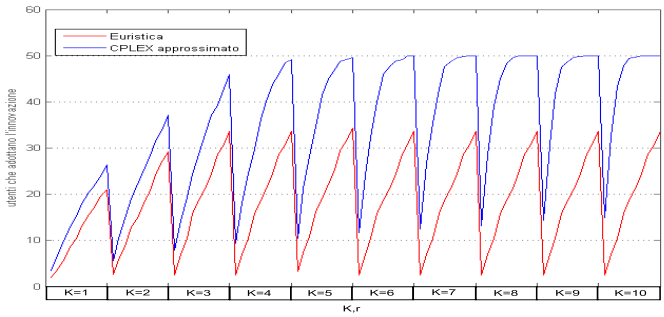


Figura 7.14: Confronto tra euristica e CPLEX approssimato

Vediamo ora un confronto delle prestazioni medie dei due approcci, in figura 7.14 viene mostrato in rosso il numero di utenti che mediamente adottano l'innovazione per ciascuna combinazione di r e K in relazione all'euristica, mentre la curva in blu rappresenta la medesima caratteristica in relazione al risolutore CPLEX. È evidente come vi siano forti differenze tra i due approcci. In figura 7.15 vengono quantificate tali differenze in termini di unità che adottano l'innovazione. In alcuni casi riscontriamo delle differenze che si aggirano intorno a trenta unità ovvero la soluzione trovata in CPLEX consente di avere trenta utenti in più, che adottano l'innovazione all'ultimo passo dell'orizzonte temporale, rispetto alla soluzione fornita dall'euristica.

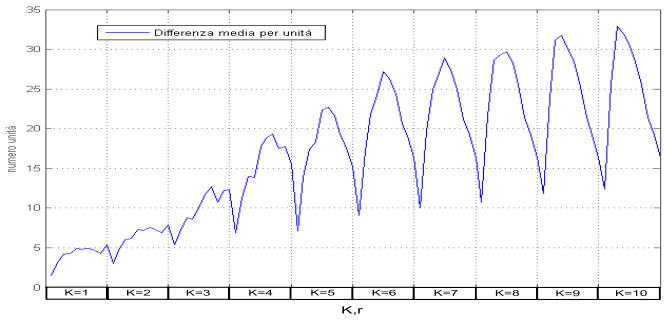


Figura 7.15: Differenza in termini di unità

	differenza%	Differenza per Unità
Minimo	3	1.5
Massimo	64	32.9

Tabella 7.5: Confronto tra l'ottimizzatore CPLEX e l'euristica

	CPLEX	Euristica
Rete 1	589 sec	1221 sec
Rete 2	471 sec	1197 sec
Rete 3	515 sec	1237 sec
Rete 4	457 sec	1137 sec
Rete 5	517 sec	1198 sec
Rete 6	473 sec	1285 sec
Rete 7	480 sec	1359 sec
Rete 8	458 sec	1236 sec
Rete 9	440 sec	1223 sec
Rete 10	490 sec	1258 sec
Media	489 sec	1255 sec

Tabella 7.6: Tempi di calcolo dell'euristica e dell'ottimizzatore CPLEX

In tabella 7.5 viene mostrato un resoconto della differenza di prestazione tra i due approcci in termini di unità e in termini percentuali. Per valutare completamente la prestazione dell'euristica abbiamo confrontato i tempi di calcolo per verificare se almeno ci sia un vantaggio rispetto all ottimizzatore CPLEX. La tabella 7.6 mostra i risultati confermando ancora una volta l'elevata efficienza di CPLEX che permette di ottenere una soluzione approssimata con dei tempi di calcolo mediamente inferiori del 60%.

Sarebbe comunque interessante implementare l'euristica con un linguaggio di programmazione più efficiente, ricordiamo che questa è stata scritta in Matlab che non consente di sfruttare in maniera efficiente i processori installati sul proprio calcolatore al contrario di CPLEX che sfrutta pienamente tutte le recenti tecnologie multiprocessore. Abbiamo visto nella sezione sulla complessità computazionale come la dipendenza tra il tempo di calcolo e il numero di innovatori di partenza non sia lineare. Il vantaggio dell'euristica risiede proprio nella relazione tra tempo computazionale e numero di innovatori di partenza, infatti questa è praticamente lineare. Consideriamo a tal proposito una semplice simulazione su una rete composta da 50 nodi e fissiamo l'orizzonte temporale a un valore pari a K=2. Effetuiamo quindi una simulazione al variare del numero di innovatori di partenza.

La figura 7.16 pone in evidenza l'andamento pressoché lineare del tempo di calcolo al variare del numero di innovatori. Questa caratteristica unita

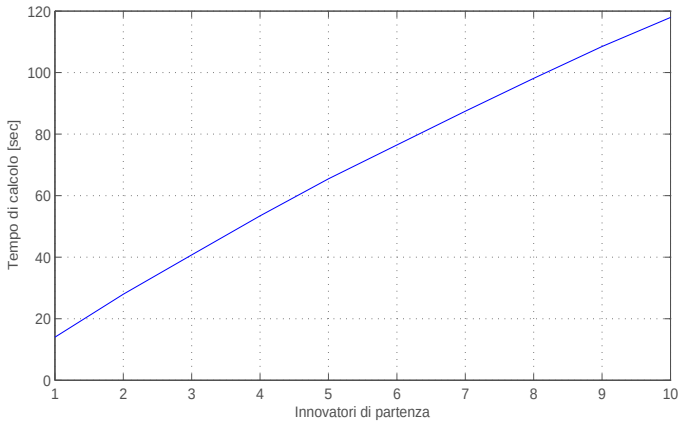


Figura 7.16: Tempo di calcolo al variare di r

ad un'implementazione più efficiente ci consente di dire comunque che questa soluzione non sia da scartare a priori. È altresì vero che le prestazioni ottenute tarando il risolutore CPLEX sono decisamente superiori anche in relazione al fatto che la simulazione appena mostrata è stata condotta con un limite sui sottoproblemi rilassati pari a 4000. Avremmo potuto ridurre questo limite comportando un peggioramento della soluzione a fronte di una riduzione dei tempi di calcolo ma anche accettando di avvicinarsi alla precisione della soluzione mostrata dall'euristica si avrebbe comunque un enorme vantaggio in termini di tempo computazionale comportando che la soluzione ottenuta con CPLEX è decisamente efficiente. In definitiva possiamo dire che l'euristica non ha mostrato dei grossi vantaggi rispetto all'approccio risolutivo implementato in CPLEX, però possiamo dire che per orizzonti temporali molto limitati può rappresentare un'alternativa valida dato che è stata riscontrata una differenza di precisione tra la soluzione dell'euristica e la soluzione subottima di CPLEX che va dal 3% al 5% con tempi di calcolo paragonabili.

7.6 Test preliminare algoritmo di riduzione della rete

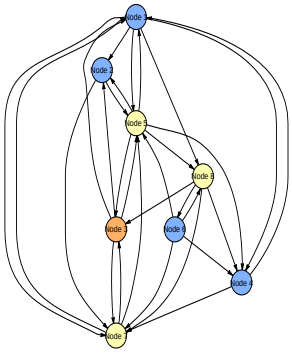
Nel capitolo 5 abbiamo visto come sia possibile ridurre la dimensione di una rete nell'ambito della risoluzione del problema 2. Ricordiamo che quest'ultimo estrae dalla rete il set di innovatori di partenza che diffondono l'innovazione su un target di utenti in K passi temporali. Il fatto di dover puntare su un target di utenti consente di scartare alcuni nodi che di fatto non sono in grado di raggiungere i nodi obbiettivo nel tempo prefissato. La possibilità o meno di ridurre la rete dipende dal numero di utenti facenti parte il set di nodi obbiettivo e dall'orizzonte temporale scelto, infatti aumentando la dimensione del set obbiettivo e dell'orizzonte temporale si riducono le possibilità che la rete possa essere ridotta. È comunque comodo sviluppare questa fase preliminare perché non comporta un dispendio elevato di tempo e risorse e anzi consente nelle fasi successive di ridurre i tempi di calcolo e la memoria occupata durante le fasi di risoluzione del problema 2. La riduzione della rete è stata implementata tramite un codice Matlab riportato in appendice B, questo consente di determinare se la rete può essere ridotta e in tal caso costruisce i dati necessari da passare all'ottimizzatore. Il codice consente inoltre di visualizzare il risultato della riduzione della rete tramite un grafo. In definitiva a seconda dei parametri del problema 2 non sempre è possibile ottenere una rete ridotta. Vediamo in tal senso alcune simulazioni per dare una valutazione qualitativa dell'algoritmo.

Consideriamo la riduzione di tre semplici reti e fissiamo la cardinalità del set di utenti obbiettivo pari a uno. Per ciascuna rete consideriamo i seguenti orizzonti temporali:

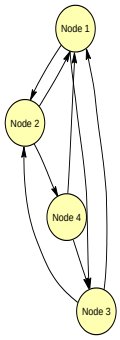
- Rete 1: Orizzonte temporale $\mathbf{K} = 1$
- Rete 2: Orizzonte temporale $\mathbf{K} = 1$
- Rete 3: Orizzonte temporale $\mathbf{K} = 3$

Date le dimensioni ridotte possiamo visualizzare l'esito della riduzione dal punto di vista grafico, vedi figura 7.17. In rosso vengono indicati i nodi obbiettivo mentre in blu indichiamo i nodi che vengono scartati. Notiamo subito che le prime due reti contraddistinte dal parametro K=1 hanno subito entrambe una riduzione della loro dimensione di partenza. La terza rete invece con parametro K=3 non ha subito alcuna riduzione dato che tutti i nodi distano dal nodo obbiettivo non più di tre passi. Quindi l'esito della riduzione dipende strettamente dalla struttura della rete, in particolare dal numero di archi dato che all'aumentare del loro numero aumentano i possibili percorsi verso il nodo o i nodi obbiettivo.

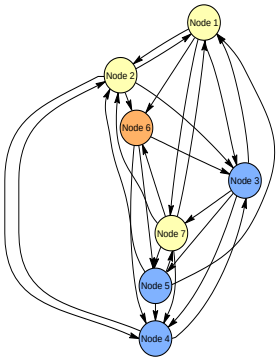
La valutazione puntuale dell'algoritmo andrebbe effettuata su reti reali per capire la sua efficacia. Queste sono tipicamente composte da un numero



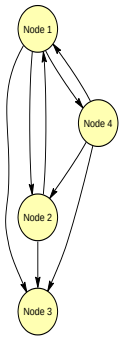
(a) Rete 1 di partenza



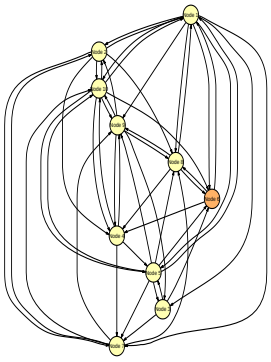
(b) Rete 1 ridotta K=1



(c) Rete 2 di partenza



(d) Rete 2 ridotta K=1



(e) Rete 3 non riducibile per K=3

Figura 7.17: Esempi di reti ridotte

di archi non troppo elevato ossia in relazione al numero di nodi della rete ciascun nodo non ha un numero elevato di archi entranti e uscenti. Quindi in fase di generazione dei dati si è cercato di rispettare questa caratteristica. Vediamo una semplice simulazione che mostra come varia la risposta dell'algoritmo in funzione del parametro K. Consideriamo una rete composta da n=1000 nodi e valutiamo il numero di nodi che vengono scartati. Il set di utenti target è composto da tre utenti. In tabella 7.7 viene mostrato l'esito di questa simulazione.

K	nodi scartati
1	968
2	752
3	162
4	4
5	0

Tabella 7.7: Riduzione della rete al variare di K

Quindi in definitiva l'applicazione dell'algoritmo di riduzione dipende strettamente da:

- La struttura della rete
- L'orizzonte temporale K
- La dimensione del set obbiettivo

Vedremo nelle sezioni successive quale sia l'impatto sul tempo computazionale, per il problema 2, dell'applicazione preliminare di questo algoritmo.

7.7 Complessità computazionale problema 2

Nei precedenti capitoli abbiamo visto come il problema 2 sia stato modellato attraverso un problema di ottimizzazione discreto. Anche in tal caso il problema risulta non trattabile per dimensioni elevate dei dati. Sono state quindi effettuate alcune simulazioni per capire quali fossero i fattori che principalmente determinano un elevato tempo compuatzionale e occupazione di memoria. I principali fattori che influenzano le prestazioni dell'ottimizzatore sono:

- Il numero di nodi n della rete
- L'orizzonte temporale K
- La cardinalità $|X_d|$ del set di utenti target

Per quanto riguarda il primo fattore è ovviamente chiaro che questo sia determinante e in quest'ottica abbiamo sviluppato l'algoritmo presentato nella sezione precedente. Concentriamoci invece sugli ultimi due aspetti.

Consideriamo una rete composta da n=30 nodi, fissiamo quindi un set di nodi obbiettivo di cardinalità $|X_d|$. Effettuiamo quindi una simulazione al variare di K.

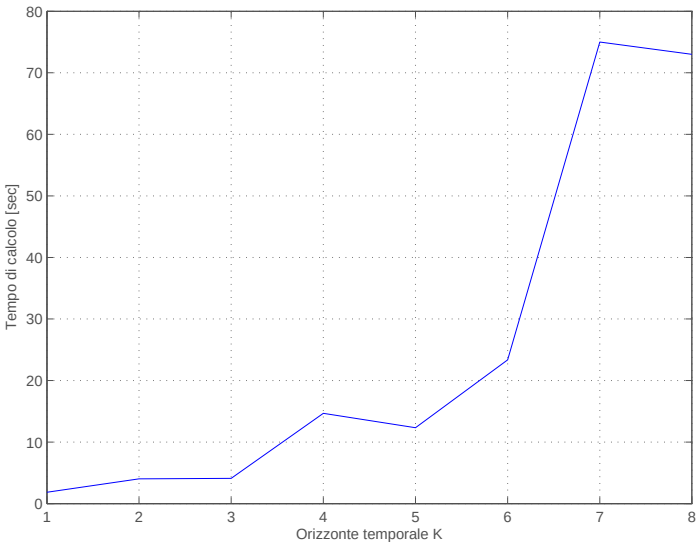


Figura 7.18: Tempo di calcolo al variare di K

Dai dati mostrati si nota che non sempre il tempo computazionale cresce all'aumentare del parametro K. Questo effettivamente è un comportamento atteso perché spesso tra due iterazioni successive la soluzione può rimanere invariata e nonostante l'aumento della dimensione della funzione obbiettivo, il tempo di calcolo non viene influenzato. Consideriamo invece la risposta rispetto alla cardinalità del set di utenti target. Consideriamo una rete composta da 250 nodi. Fissiamo il valore del parametro K=2. Quindi valutiamo i tempi di calcolo al variare di $|X_d|$. In tal caso è stato riscontrato, come mostrato in tabella 7.8, un repentino aumento del tempo di calcolo in funzione della cardinalità del set di utenti target. Questo è assolutamente normale se consideriamo che comporta una maggiore complessità dei vincoli del problema.

$\|X_d\|$	Tempo di calcolo[sec]
3	11,73
6	13,02
8	16,75
12	19,43
16	82
20	1667

Tabella 7.8: Tempo computazionale al variare di $|X_d|$

7.8 Test preliminare del problema 2

In questa sezione viene mostrato un semplice esempio del problema 2 in particolare vedremo la soluzione fornita dal problema di ottimizzazione discreto, vedremo quindi che la versione rilassata di quest'ultimo fornisce un limite inferiore della cardinalità del set di innovatori di partenza. Il test preliminare è stato condotto su una rete con le seguenti caratteristiche:

- Numero di utenti n=30
- Orizzonte temporale K=1...10
- Cardinalità del set obbiettivo $|Xd| = 8$

Risolvendo il problema di ottimizzazione otteniamo il seguente risultato:

```
K=1  0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 1 0 1 0 1 0 0 0
K=2  0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 0
K=3  0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0
K=4  0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0
K=5  0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 1 0 0 0
K=6  0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
K=7  0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
K=8  0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
K=9  0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
K=10 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
```

Per comodità sono state indicate solo le prime 30 variabili della funzione obbiettivo che identificano come si configura la soluzione al variare dell'orizzonte temporale. In particolare le variabili contraddistinte dal valore 1 rappresentano i nodi che vengono scelti come innovatori di partenza per diffondere l'innovazione sule set di utenti obbiettivo. Si nota come il numero di innovatori tenda a decrescere all'aumentare di K, questo aspetto è reso

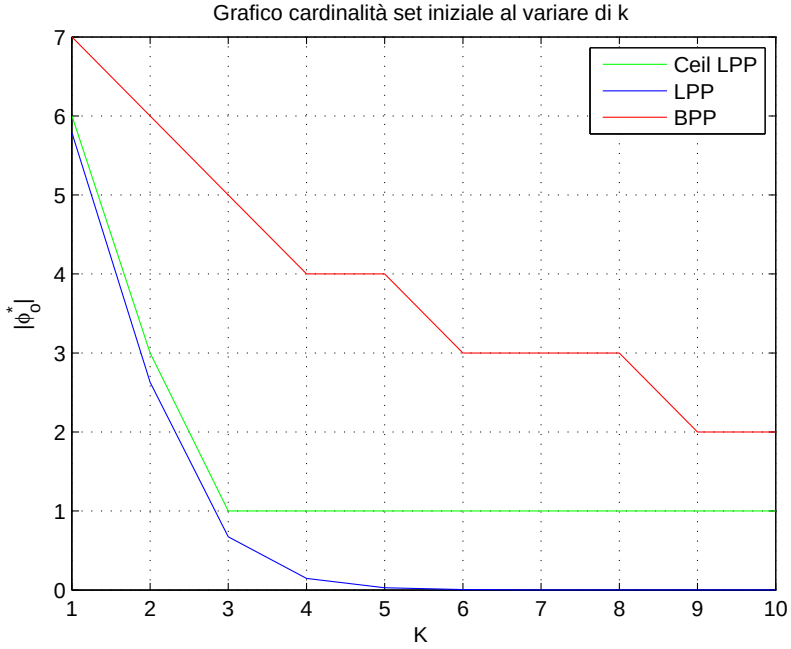


Figura 7.19: Cardinalità del set di innovatori di partenza

evidente dalla figura 7.19 nel quale viene riportato l'andamento della cardinalità del set di innovatori di partenza al variare dell'orizzonte temporale. In rosso vediamo l'andamento ottenuto dalla soluzione ottima fornita dal problema di ottimizzazione discreto, si nota chiaramente che man mano che aumenta il valore di K il set di innovatori scelto dall'ottimizzatore decresce, inoltre si nota che il set di innovatori di partenza non supera mai in numero il set obbiettivo, al più può avere la stessa cardinalità. In verde vediamo invece come il problema rilassato fornisca un limite inferiore della cardinalità del set di innovatori di partenza. Questo tracciato si ottiene arrotondando all'intero superiore la cardinalità fornita dal problema rilassato che necessariamente fornirà un risultato frazionario come mostrato dalla curva in blu.

7.8.1 Valutazione dell'algoritmo di riduzione della rete

Ora focalizzeremo la nostra attenzione sulle prestazioni ottenute riducendo le dimensioni della rete di partenza da testare. In particolare testeremo dieci reti ognuna con le stesse caratteristiche, i parametri salienti della simulazione sono i seguenti:

- Numero di utenti n=250
- Orizzonte temporale K=2
- Cardinalità del set obbiettivo $|Xd| = 3$

Come detto nelle sezioni precedenti la soluzione ottenuta riducendo la rete continua ad essere quella ottima quindi focalizzeremo l'attenzione sui tempi di calcolo ottenuti dalle simulazioni.

K=1	Rete di partenza		Rete ridotta	
	Nodi	Tempo di calcolo[sec]	Nodi	Tempo di calcolo[sec]
Rete 1	250	3.73	151	2.97
Rete 2	250	2.97	114	2.67
Rete 3	250	2.86	107	1.95
Rete 4	250	3.57	131	2.16
Rete 5	250	3.37	141	3.33
Rete 6	250	3.60	105	2.40
Rete 7	250	3.27	117	2.83
Rete 8	250	3.81	117	2.77
Rete 9	250	3.29	106	2.79
Rete 10	250	3.43	123	3.43
Media	250	3.39	121.2	2.73

Tabella 7.9: Confronto dei tempi di calcolo per K=1

In tabella 7.9 viene mostrato il confronto dei tempi di calcolo del problema di ottimizzazione discreto effettuato con le reti di partenza con quelli ottenuti riducendone le dimensioni. Si nota chiaramente che in ogni caso la riduzione della dimensione della rete di partenza produce un tempo di calcolo inferiore. I tempi di calcolo sono dell'ordine dei secondi perchè il set target e l'orizzonte temporale risultano limitati proprio per far in modo che per ogni rete avvenisse con successo la riduzione della rete. Consideriamo ora l'impatto sulle prestazioni dell'orizzonte temporale K che come sappiamo influenza l'efficacia dell'algoritmo di riduzione.

Consideriamo 10 reti composte da 1000 utenti per ciascuna di essa fissiamo 10 utenti obbiettivo. Risolviamo quindi il problema di ottimizzazione col parametro K=2 e successivamente col valore K=3. Valutiamo quindi quale sia mediamente l'impatto dei tempi di calcolo per effetto dell'applicazione dell'algoritmo di riduzione. Dalla prima simulazione, come mostrato in tabella 7.10, notiamo che l'algoritmo di riduzione delle reti consente se applicato preliminarmente di ridurre i tempi di calcolo in fase di ottimizzazione di circa il 50%. Notiamo inoltre che la riduzione della rete porta ad ottenere reti con un numero di nodi di circa la meta rispetto al numero di partenza, questo dato pare evidenziare che la relazione tra i tempi di calcolo e la dimensione della rete sia di tipo lineare. La stessa simulazione effettuata per K=3 evidenzia come la riduzione del tempo di calcolo in seguito all'applicazione preliminare dell'algoritmo di riduzione comporti una diminuzione dei tempi di calcolo di ottimizzazione dell'ordine del 10%. Questo è ovviamente naturale e dipende dal fatto che le reti subiscono una riduzione inferiore rispetto a quella riscontrata nella simulazione precedente. Le reti subiscono un decremento della loro dimensione in termini di nodi pari a circa il 10%,come evidenziato nella tabella 7.11 comportando, come nella simulazione precedente, che il tempo di calcolo e la dimensione delle reti siano legate linearmente.

K=2	Rete di partenza		Rete ridotta	
	Nodi	Tempo di calcolo[sec]	Nodi	Tempo di calcolo[sec]
Rete 1	1000	36	558	17.64
Rete 2	1000	35.4	543	18.22
Rete 3	1000	34.7	522	18.34
Rete 4	1000	34.2	537	17.5
Rete 5	1000	36.2	526	18.64
Rete 6	1000	34.7	573	17.63
Rete 7	1000	37.3	548	18.12
Rete 8	1000	37.8	562	17.15
Rete 9	1000	36.15	559	18.9
Rete 10	1000	36.9	565	19.2
Media	1000	35.9	549.3	18.13

Tabella 7.10: Confronto dei tempi di calcolo per K=2

K=3	Rete di partenza		Rete ridotta	
	Nodi	Tempo di calcolo[sec]	Nodi	Tempo di calcolo[sec]
Rete 1	1000	43.61	942	39.8
Rete 2	1000	42.70	948	38.7
Rete 3	1000	43.51	936	38.12
Rete 4	1000	42.76	941	39.16
Rete 5	1000	41.81	938	39.98
Rete 6	1000	43.77	933	39.18
Rete 7	1000	45.39	943	39.74
Rete 8	1000	42.50	958	38.26
Rete 9	1000	44.16	923	39.42
Rete 10	1000	43.12	945	38.56
Media	1000	43.3	940.7	39.1

Tabella 7.11: Confronto dei tempi di calcolo per K=3

Parallelizzazione dati

L'analisi del problema di partenza consente di ottenere anche altri approcci risolutivi che comportino un vantaggio in termini di prestazioni. In particolare in alcuni casi è possibile applicare una parallelizzazione sui dati. Questa prevede l'esecuzione parallela di una medesima istruzione in contemporanea su più di un dato. In casi particolari infatti la riduzione della rete può generare due o più reti che risultano disgiunte ossia tra esse non collegate. Questo comporta che la soluzione del problema di ottimizzazione di partenza possa essere risolto effettuando l'ottimizzazione su due o più reti contemporaneamente. Per capire meglio consideriamo il seguente esempio. Consideriamo la rete in figura 7.20, i nodi in rosso rappresentano i nodi target. Le soglie assumono i seguenti valori: $\lambda_1 = \lambda_2 = \lambda_3 = \lambda_4 = \lambda_5 = \lambda_6 = \lambda_7 = \lambda_8 = \lambda_9 = \lambda_{10} = 0.5$

Se svolgessimo l'operazione di ottimizzazione con K=2 otterremmo la seguente soluzione:

```
1 2 3 4 5 6 7 8 9 10
1 0 0 1 0 0 0 0 0 0
```

Abbiamo indicato i primi 10 elementi della funzione obbiettivo che indicano quali siano gli innovatori di partenza. In seguito alla riduzione della rete con orizzonte temporale K=2 si otterrebbero due reti disgiunte caratterizzate

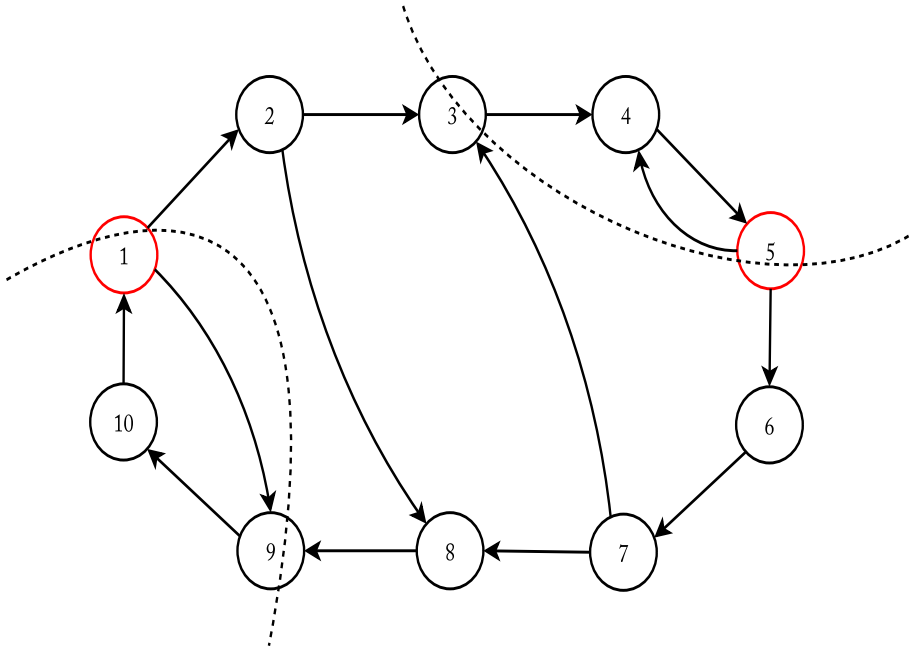


Figura 7.20: Rete da ridurre

dalle seguenti matrici di adiacenza:

$$A_1 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \qquad A_2 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Risolvendo il problema di ottimizzazione su queste reti si ottiene rispettivamente:

```
1 9 10          3 4 5
1 0 0           0 1 0
```

Quindi abbiamo ottenuto la medesima soluzione del problema di ottimizzazione di partenza. In definitiva si ottengono due vantaggi:

1. Si riduce il tempo di calcolo in percentuale di un fattore che dipende dalla percentuale riduzione delle dimensioni della rete di partenza.

2. Si divide il tempo di calcolo ottenuto in seguito alla riduzione della rete di partenza di un fattore pari al numero di sottoreti ottenibili da quella ridotta. Questa affermazione è valida se si ottengono sottoreti di dimensioni tutte uguali, ovviamente questo non è sempre vero.

In relazione all'esempio si otterrebbe una riduzione dei tempi di calcolo di circa il 70%. Nel lavoro di tesi non è stato approfondito questo approccio,

comunque è stata scritta una semplice estensione dell'algoritmo di riduzione delle reti nel quale viene segnalata la possibilità di svolgere l'ottimizzazione in parallelo visualizzando i grafi delle reti disgiunte che derivano dal processo di riduzione della rete di partenza. Per maggiori dettagli si faccia riferimento alla sezione B.2.2 nell'appendice B.

7.9 Test determinazione set coesivo massimale

In questa sezione vedremo un semplice esempio di calcolo del set coesivo massimale. Vedremo in particolare come possa essere calcolato secondo i due metodi visti nel capitolo 4.

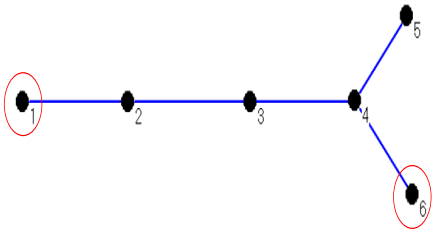


Figura 7.21: Configurazione iniziale della rete

Prendiamo in considerazione la rete in figura 7.21, i nodi cerchiati in rosso sono quelli appartenenti al set di innovatori di partenza. A partire da questo set la configurazione finale della rete ossia gli utenti che alla fine adottano l'innovazione può essere estrapolato dalla conoscenza del set coesivo massimale.

Risolvendo il problema col metodo di ottimizzazione discreto otteniamo la seguente soluzione: $x = [001110]$ quindi il set coesivo massimale ha la seguente composizione: $\mathcal{M} = \{3, 4, 5\}$ di conseguenza il set di utenti che alla fine del processo di diffusione adottano l'innovazione assume la seguente configurazione: $\phi^* = \mathcal{V} - \mathcal{M} = \{1, 2, 6\}$ quindi in seguito all'introduzione del set di innovatori di partenza soltanto il nodo 2 adotta l'innovazione.

La configurazione finale della rete è mostrata in figura 7.22. Vediamo ora un'altro esempio sulla stessa rete considerando il seguente insieme di innovatori di partenza: $\phi(0) = \{3, 4\}$. Risolvendo il problema di ottimizzazione otteniamo il seguente set coesivo massimale: $\mathcal{M} = \{\emptyset\}$, quindi in questo caso la rete assume la seguente configurazione finale: $\phi^* = \mathcal{V} - \mathcal{M} = \{1, 2, 3, 4, 5, 6\}$, ciò significa che tutti gli utenti adottano l'innovazione alla fine del processo di diffusione dell'innovazione. I semplici esempi finora proposti possono essere risolti attraverso la versione rilassata dell'ottimizzatore discreto che abbiamo visto nella sezione 4.3.1. Ricordiamo che questa si basa sulla risoluzione di una serie di problemi rilassati costruiti a partire dal problema

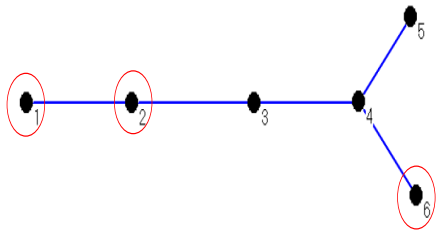


Figura 7.22: Configurazione finale della rete

binario di partenza. La soluzione in tal caso viene ottenuta risolvendo due problemi di programmazione lineare. In relazione al primo esempio risolto attraverso il problema discreto la risoluzione del problema rilassato si snoda attraverso i seguenti semplici passi:

1. Viene risolto il problema lineare di partenza con seed set $y_0 = [0, 0, 1, 1, 0, 0]^T$
2. La soluzione del 1 problema è il seguente: $x_0 = [1, 0.98, 0, 0, 0, 0]^T$
3. Viene risolto il 2 problema col seguente set di partenza: $y_1 = \lceil 1 - x_0 \rceil = [0, 1, 1, 1, 1, 1]$
4. La soluzione ottenuta è la seguente: $x_1 = [0, 0, 0, 0, 0, 0]$

Si nota che la soluzione del primo problema contiene una variabile frazionaria, l'algoritmo non si arresta finché non viene trovata una soluzione intera. Il numero di passi effettuati è pari a $k = 2$ quindi rispetta il limite inferiore imposto dalla proposizione 4.3.3, infatti risulta:

$$2 \leq n - \mid \phi_0 \mid - \mid \mathcal{M} \mid + 1 = 6 - 2 - 3 + 1 = 2$$

.

Capitolo 8

Conclusioni

In questa tesi ci siamo proposti di studiare delle tecniche di ottimizzazione per l'analisi della diffusione delle innovazioni nei social networks. In particolare sono stati trattati i seguenti problemi:

- Massimizzazione dell'influenza in tempo finito.
- Diffusione dell'innovazione attraverso un set di utenti obbiettivo in tempo finito
- Determinazione del set coesivo massimale a partire da un set di innovatori.

Ognuno di questi problemi è modellizzabile attraverso un problema di ottimizzazione discreto di cui è stata proposta un'implementazione in linguaggio OPL. Data la loro natura combinatoria i problemi sopracitati non sono trattabili per istanze reali dei dati, a tal proposito sono state sviluppate delle tecniche di ottimizzazione basate su approcci risolutivi approssimati o sullo studio del problema di partenza. Per il problema 1 è stata estesa l'euristica proposta da [11] basata sulle proprietà del modello di diffusione a soglia lineare e delle reti acicliche. È stato inoltre proposto un metodo risolutivo basato sulla taratura del risolutore CPLEX. Quest'ultimo approccio è certamente quello che ha mostrato i migliori risultati in termini di velocità di calcolo e di qualità della soluzione fornita. Per quanto riguarda il problema 2 è stato proposto un metodo risolutivo basato sullo studio del problema di partenza, in particolare si è mostrato come sia possibile risolvere il problema di ottimizzazione riducendo preliminarmente la dimensione dei dati di partenza ottenendo comunque la soluzione ottima. Il terzo problema, già ampiamente trattato, è stato implementato anch'esso in linguaggio OPL per completare il set di strumenti software. Possiamo dire certamente che vi possano essere numero sviluppi futuri, da un lato è possibile definire nuovi modelli che siano più aderenti alla realtà, ad esempio che tengano conto della diffusione di più innovazioni concorrenti, dall'altro è possibile migliorare le

prestazioni degli strumenti software sviluppati, infatti è possibile analizzare i primi due problemi dal punto di vista delle operazioni parallelizzabili. In particolare l'euristica relativa al problema 1 può essere scritta in modo tale che, nella fase preparatoria, le reti acicliche possano essere generate non sequenzialmente bensì in parallelo. Anche la fase di selezione degli innovatori di partenza può essere resa più efficiente se si pensa di aggiornare le strutture dati relative a ciascun grafo aciclico in modo parallelo. Un discorso analogo può essere fatto per il problema 2, infatti in alcuni casi particolari non solo è possibile ridurre le dimensioni della rete ma è possibile ottenere due o più reti disgiunte trattabili quindi in parallelo dall'ottimizzatore.

Appendice A

ILOG OPL e CPLEX

Introduzione

CPLEX rappresenta uno dei pacchetti software più efficienti, tra quelli disponibili sul mercato, per la risoluzione di problemi di programmazione lineare e di programmazione lineare intera o mista.

In particolare, CPLEX consente:

- la risoluzione di problemi lineari, anche di notevoli dimensioni (migliaia di variabili e vincoli), tra cui i problemi di ottimizzazione su reti (problemi di flusso), mediante l'utilizzo dell'algoritmo del simplesso primale e duale.
- la risoluzione di problemi di programmazione lineare intera e mista mediante procedure basate sull'enumerazione implicita delle soluzioni (procedure ad albero tipo Branch and Bound). In particolare, il software consente la risoluzione di istanze di notevoli dimensioni (soprattutto in confronto ad altri software concorrenti) per problemi cosiddetti difficili (NP-hard).

CPLEX è un valido strumento per la risoluzione di problemi reali in disparati campi applicativi quali:

- Problemi nel settore dei trasporti. Esempi delle principali classi di problemi analizzabili in termini di programmazione lineare e intera sono:
  - Problemi di percorso e instradamento ottimo di veicoli. Tali problemi rivestono notevole interesse applicativo in quanto compaiono frequentemente in problemi di attribuzione ottima di risorse ed in problemi di distribuzione, dove più veicoli sono impiegati per eseguire consegna e/o raccolta di determinati beni a clienti distribuiti sul territorio:

  - Problemi di sequenziamento e schedulazione.
  - Problemi di traffico aereo.

- Problemi di ottimizzazione della produzione in sistemi produttivi complessi (problemi di sequencing e scheduling). Tra questi rientrano i classici problemi di taglio: aziende (ad esempio del settore del legno) che utilizzano materiali pregiati devono spesso cercare di minimizzare lo sfrido. La ricerca di schemi di taglio ottimali che permettano di soddisfare gli ordini minimizzando il valore delle parti scartate è una classe interessante di problemi che richiede la soluzione di modelli di programmazione intera.

- Sviluppo di sistemi logistici complessi. Il problema della pianificazione e della gestione di tali sistemi richiede l'adozione di decisioni che riguardano principalmente la localizzazione di impianti ed infrastrutture (centri di produzione, depositi...) necessari per la realizzazione dei servizi da espletare e l'organizzazione di servizi orientati a soddisfare una certa domanda.

- Problemi nel settore delle telecomunicazioni. I problemi riguardano il progetto di reti di telecomunicazione affidabili (in particolare quelle con l'impiego di fibre ottiche) e la loro gestione nella fase di instradamento del traffico. Un altro caso di particolare interesse è quello dell' assegnamento di frequenze per stazioni radio-base di telefonia cellulare. Tutti i problemi indicati possono essere affrontati applicando modelli di programmazione matematica.

La teoria dei grafi rappresenta un classico esempio di settore comune a più discipline. Molti problemi appartenenti alla teoria dei grafi sono formulabili in termini di programmazione lineare e risolvibili con l'utilizzo di CPLEX. Rientrano in tale settore diversi problemi di carattere ingegneristico (ad esempio la navigazione robotica) che richiedono la soluzione di problemi di cammino minimo su grafi. Inoltre, frequenti sono i problemi basati su reti di tipo ingegneristico (flusso di traffico), di tipo economico (flusso rappresentato dalla merce nei canali di scambio) o di tipo informatico (flusso di informazioni telematiche), in cui è richiesto di massimizzare il flusso spedito da un origine ad una destinazione o di minimizzare i costi associati ai flussi spediti.

A.1 Componenti di ILOG CPLEX:

CPLEX può essere utilizzato essenzialmente in tre modi, come mostrato in [16] garantendo un ampio spettro di possibilità per la definizione di problemi di programmazione lineare:

	Microsoft Windows	Unix
C++	`ilocplex.lib.concert.lib`	`libilocplex.a libconcert.a`
Java	`cplex.jar`	`cplex.jar`
.NET	`ILOG.CPLEX.dll, ILOG.Concert.dll`	

Tabella A.1: Librerie di Concert technology

- **CPLEX interactive optimizer:** È un programma eseguibile che permette la lettura da files, in formati standard, di problemi di programmazione. Il software restituisce la soluzione in un foglio di testo.
- **Concert technology:** È un set di librerie scritte in C++, Java e .NET che mettono a disposizione un interfaccia che consente di incapsulare l'ottimizzatore CPLEX in codice scritto in C++, Java e .NET. In tabella 1.1 vengono indicate le librerie su piattaforma Windows e Unix.
- **CPLEX callable library:** È un set di librerie scritte in linguaggio C che consente di incapsulare l'ottimizzatore CPLEX in codici scritti in linguaggio C, Visual basic, Fortran o qualsiasi codice in grado di richiamare librerie C.

A.2 ILOG OPL Development studio

ILOG Development Studio fornisce un completo ambiente di sviluppo per modelli di ottimizzazione costituito da:

1. Il linguaggio di modellazione(OPL). Costituisce un vero e proprio linguaggio specifico per la modellazione di problemi di ottimizzazione vincolata.
2. Un strumento riga di comando (oplrun) che permette l'esecuzione di modelli anche senza l'impiego di interfaccia grafica.
3. Un ambiente integrato (IDE) per lo sviluppo e testing dei modelli.
4. IBM ILOG ODM Enterprise fornisce una piattaforma scalabile per lo sviluppo e la distribuzione di soluzioni di pianificazione analitiche, estremamente efficienti e basate sull'ottimizzazione per i responsabili del supporto decisionale di business.

ILOG OPL è uno strumento che offre numerosi vantaggi dal punto di vista della modellazione permettendo la creazione in tempi brevi di modelli anche

molto complessi. D'altro canto OPL non è un vero e proprio linguaggio di programmazione e quindi non offre le stesse potenzialità di questi ultimi. Per applicazioni molto complesse è consigliato utilizzare la tecnologia Concert, menzionata nella sezione precedente, che permette di incapsulare CPLEX nei più moderni linguaggi di programmazione.

A.2.1 ILOG OPL Development studio IDE

L'ambiente di sviluppo di OPL fornisce numerosi strumenti che facilitano lo sviluppo di modelli di ottimizzazione in tutte le loro fasi. In figura A.1

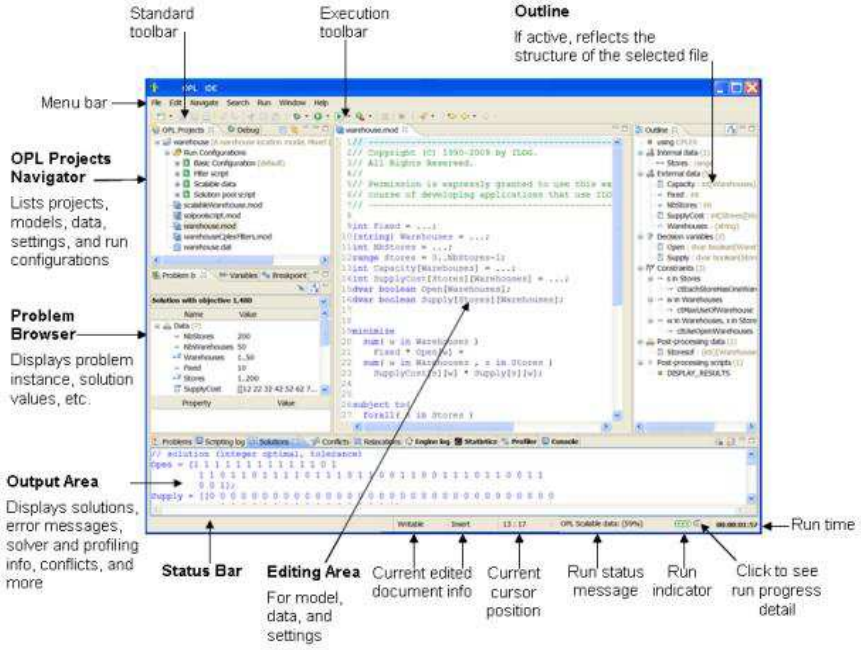


Figura A.1: Interfaccia grafica di OPL Development studio

viene mostrata l'interfaccia grafica di OPL development studio su piattaforma Microsoft Windows. Come mostrato in [17] l'interfaccia è costituita essenzialmente dalle seguenti sezioni:

- **Project navigator:** Consente di effettuare tutte le principali operazioni di gestione dei modelli quali la loro creazione o l'eliminazione, l'inserimento di nuovi dati e la configurazione del motore di ottimizzazione.
- **Editing area:** Fornisce un editor di testo sul quale è possibile modellare dei problemi di ottimizzazione in linguaggio OPL.

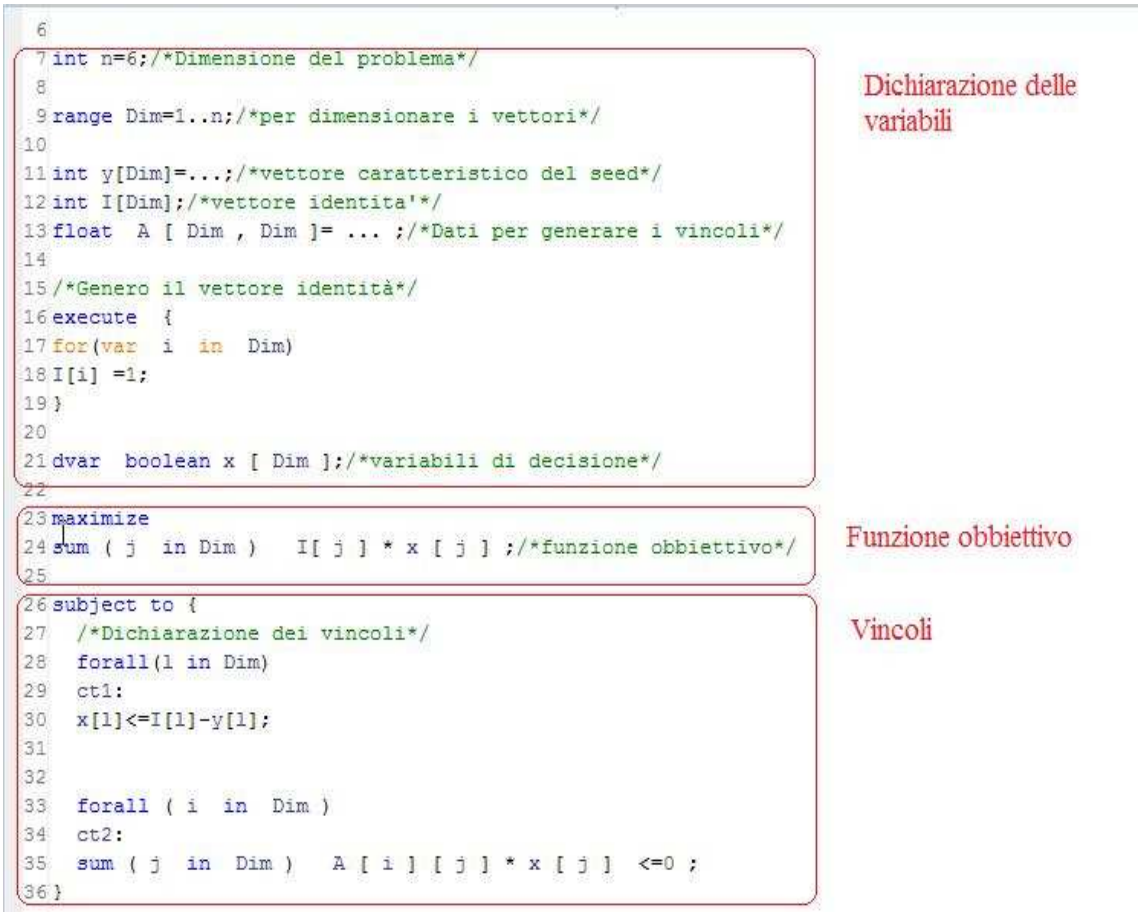
- **Problem browser:** Consente la visualizzazione di un'istanza del modello permettendo anche la verifica della correttezza della costruzione dei vincoli nel problema di ottimizzazione.
- **Outline:** Fornisce una panoramica della struttura del modello implementato visualizzando le strutture dati generate internamente ed esternamente, le variabili di decisione e i vincoli del problema.
- **Output area:** È forse lo strumento più importante che permette di interpretare le soluzioni ottenute. Quest'area è divisa in diverse sottosezioni nel quale è possibile ottenere varie informazioni dettagliate. Una delle più importanti è certamente la sezione detta *profiler* che consente di stabilire l'utilizzo della memoria permettendo un ottimizzazione accurata del codice.

Oltre gli strumenti appena citati l'ambiente di sviluppo fornisce anche un'area interamente dedicata al debug delle applicazioni.

A.3 Il linguaggio di modellazione OPL

I linguaggi di modellazione nascono dalla necessità di esprimere i problemi di programmazione in una formulazione che ci si avvicini il più possibile alla notazione matematica. Questa caratteristica è certamente un pregio ma può costituire un difetto: infatti spesso i linguaggi di modellazione hanno una struttura rigida che di fatto non consente la stessa flessibilità di un vero e proprio linguaggio di programmazione. CPLEX è utilizzabile tramite software di modellazione indipendenti quali: AIMMS, AMPL, GAMS, MPL, OpenOpt, OptimJ e TOMLAB. In questa sezione ci concentreremo sul linguaggio impiegato in questa tesi. La modellazione di un problema di programmazione è organizzata in specifici files che permettono la separazione tra il modello e i dati. Questo approccio modulare consente di organizzare il codice in maniera ordinata ed efficiente. I files principali che costituiscono un'applicazione in OPL sono i seguenti:

- **Model:** I file model contengono l'implementazione del problema di programmazione, quindi al suo interno troviamo la dichiarazione delle strutture dati, della funzione obbiettivo e dei vincoli di ottimizzazione. Tipicamente in questi file non è contenuta nessuna istanza dei dati anche se è possibile dichiarare dei dati internamente al modello. Questo approccio può essere impiegato per modelli molto semplici ma risulta molto scomodo per modelli complessi. L'estensione di questi files è (.mod).
- **Data:** I file dati contengono i dati del problema di ottimizzazione. Questi possono essere dichiarati sia manualmente sia attraverso un'op-




```
 6
 7 int n=6;/*Dimensione del problema*/
 8
 9 range Dim=1..n;/*per dimensionare i vettori*/
10
11 int y[Dim]=...;/*vettore caratteristico del seed*/
12 int I[Dim];/*vettore identita'*/
13 float  A [ Dim , Dim ]= ... ;/*Dati per generare i vincoli*/
14
15 /*Genero il vettore identità*/
16 execute  {
17 for(var  i  in  Dim)
18 I[i] =1;
19 }
20
21 dvar  boolean x [ Dim ];/*variabili di decisione*/
22
23 maximize
24 sum ( j  in Dim )   I[ j ] * x [ j ] ;/*funzione obbiettivo*/
25
26 subject to {
27   /*Dichiarazione dei vincoli*/
28   forall(l in Dim)
29   ct1:
30   x[l]<=I[l]-y[l];
31
32
33   forall ( i  in  Dim )
34   ct2:
35   sum ( j  in  Dim )   A [ i ] [ j ] * x [ j ]  <=0 ;
36 }
```

Figura A.2: Esempio di codice OPL

portuna sintassi che consente di invocare dei database esterni. I file dati (.dat) consentono quindi la separazione tra i dati e il modello.

- **Option:** I file di estensione (.ops) consentono di impostare in maniera accurata i parametri del risolutore.

La figura A.2 mostra un semplice esempio di codice scritto in linguaggio OPL.

A.3.1 OPL Script

Finora abbiamo visto le caratteristiche di OPL che più lo avvicinano ad una notazione prettamente matematica. In realtà spesso è necessario manipolare le soluzioni e costruire strutture di controllo complesse. In OPL questo è reso possibile dal linguaggio di *scripting*. Questo consente di effettuare essenzialmente le seguenti operazioni:

- Operazioni di preprocessing sui dati. Spesso è necessario costruire dei dati internamente al modello. Questo può essere fatto prima che venga definito il modello matematico vero e proprio.
- Tramite gli script è possibile definire delle strutture di controllo complesse. Un esempio tipico è quello di scindere il problema di programmazione in uno o più sotto problemi. Questi vengono coordinati tramite uno script che si cura di invocare i sottoproblemi e di crearne un'istanza in base ai dati in ingresso. Tramite gli script è poi possibile risolvere un problema iterativamente modificando i dati ad ogni iterazione.
- Spesso è necessario isolare dati sensibili che non vengono resi immediatamente disponibili dall'output area. Questa operazione può essere svolta attraverso un opportuno script.

Gli script risiedono in blocchi denominati *execute* questi vengono inseriti prima della definizione del problema di programmazione se devono svolgere operazioni di preprocessing mentre vengono posti nella parte finale del file (.mod) che definisce il modello se devono effettuare delle operazioni di postprocessing. Per maggiori dettagli sugli *script* si faccia riferimento al manuale del linguaggio OPL [18]

A.3.2 Inizializzazione dei dati

Il linguaggio OPL fornisce una vasta gamma di possibilità di inizializzazione dei dati. Questa può essere di tipo interno o di tipo esterno. L'approccio impiegato nell'inizializzazione dei dati riveste un ruolo importante.

Inizializzazione interna L'inizializzazione interna consiste nel dichiarare i dati all'interno del file in cui risiede il modello. Di seguito si mostra un semplice esempio nel quale viene inizializzato un array.

```
int a[1..5] = [b+1, b+2, b+3, b+4, b+5];
```

Questo approccio di inizializzazione viene impiegato per problemi di programmazione molto semplici in cui risulta possibile inserire i dati a mano. Per problemi di programmazione complessi si preferisce inizializzare i dati esternamente perché permette di scrivere del codice più leggibile e facilmente manutenibile. L'aspetto fondamentale risiede nel fatto che l'inizializzazione interna alloca i dati in memoria soltanto quando necessario cioè seguono il flusso di esecuzione del problema di programmazione. Questo garantisce un impiego più efficiente della memoria che viene allocata solo quando necessario.

Tipo database
DB2
MS SQL
ODBC
Oracle 9 e successivi
OLE DB

Tabella A.2: Database supportati da OPL

Inizializzazione esterna L'inizializzazione esterna avviene tramite la dichiarazione dei dati in un file di estensione (.dat). I dati possono essere scritti manualmente oppure attraverso un'opportuna sintassi è possibile instanziare dati che risiedono in file esterni. Questo consente di costruire l'applicazione in maniera modulare comportando una facile manutenzione e separando i dati dal modello. È possibile dichiarare dati provenienti da database relazionali oppure da file excel. I database supportati, come specificato in [19], sono indicati in tabella A.2. La dichiarazione esterna dei dati influenza notevolmente l'utilizzo della memoria infatti l'allocazione dei dati avviene non appena viene generata completamente l'istanza del modello. Questi dati permangono in memoria finché non viene completata l'esecuzione del problema di programmazione. Per evitare quindi la saturazione della memoria è buona norma liberare esplicitamente la memoria. In questa tesi i dati sono stati inizializzati esternamente data la complessità dei modelli implementati. I dati sono stati generati in Matlab e quindi salvati in file Excel. Di seguito mostriamo un semplice esempio di dichiarazione dati esterni in Excel.

```
/* .dat file */
SheetConnection sheet(transport.xls);
```

A.4 Algoritmi di Cplex:

ILOG CPLEX è uno strumento per la risoluzione di problemi di programmazione lineare nella forma:

$$\begin{cases} \textit{Massimizza} & \vec{c} \cdot \vec{x} \\ \textit{tale che} & \mathcal{A} \cdot \vec{x} = \vec{b} \\ & x_j \geq 0 \end{cases}$$

È possibile inoltre risolvere delle estensioni dei problemi di programmazione lineare, come mostrato in [16]:

- **Network flow problems:** È un caso particolare di programmazione lineare che CPLEX risolve in maniera efficiente attraverso l'analisi della struttura del problema.

- **Quadratic programming problems(QP):** La funzione obbiettivo del problema di programmazione lineare viene esteso per contenere termini quadratici.
- **Quadratically constraints programming(QCP):** Rappresentano problemi di programmazione in cui i vincoli possono contenere dei termini quadratici.
- **Mixed integer programming(MIP):** Sono una classe particolare di problemi di programmazione in cui alcuni termini della funzione obbiettivo possono assumere solo valori interi. Questa classe generale comprende la classe dei problemi di programmazione intera (IP) nel quale tutte le variabili della funzione obbiettivo devono assumere valori interi. Se la funzione obbiettivo è caratterizzata da una relazione lineare si parla di *Mixed integer linear programming*(MILP)

I problemi di programmazione lineare utilizzano gli algorimi di *presolve*. Questi vengono richiamati da CPLEX in maniera completamente trasparente all'utente. Essi hanno il compito di analizzare la struttura del problema identificando delle condizioni per il quale sia possibile ridurne la complessità. I principali algoritmi che il prerisolutore può richiamare sono i seguenti:

- Simplesso primale e duale
- Simplesso specifico per reti
- Primal/Dual barrier log

Per quanto riguarda i problemi di programmazione intera o mista vengono utilizzati algoritimi detti di enumerazione implicita basati sul metodo del *branch & cut*. Quest'ultimo è un ibrido che impiega i metodi di *branch & bound* e *cutting planes*. L'algoritmo risolve una serie di problemi lineari andando a rilassare il vincolo di interezza del problema intero di partenza. Se l'esito del rilassamento continuo produce una soluzione in cui anche una sola delle variabili di decisione è frazionaria l'algoritmo aggiunge dei vincoli in modo da ridurre la dimensione della regione dello spazio delle soluzioni ammissibili escludendo quelle aree che contenevano la variabile frazionaria trovata precedentemente. La parte appena descritta è quella che impiega il metodo dei piani di taglio. Quando un sottoproblema continuo produce una soluzione con variabili frazionarie il problema viene ulteriormente suddiviso appunto col metodo del *branch & bound*. CPLEX in particolare costruisce un albero di ricerca nel quale il nodo di root rappresenta il rilassamento continuo del problema intero di partenza. Ogni nodo rappresenta un sottoproblema continuo generato in seguito all'applicazione del metodo dei piani di taglio. Per maggiori dettagli sui metodi del *branch & cut* e dei piani di taglio si faccia riferimento ai richiami sulla programmazione lineare.

A.5 Parametri del risolutore CPLEX

CPLEX di default analizza il modello implementato e cerca di impiegare la migliore strategia di risoluzione. È comunque possibile modificare attraverso i file di estensione (.ops) il comportamento del risolutore. I parametri più importanti sono i seguenti:

- Impostazioni generali: Permette di impostare la cartella di lavoro e fornisce delle opzioni su come visualizzare il tempo di calcolo delle simulazioni.
- Emphasis: È possibile porre un enfasi sulla riduzione della memoria impiegata e sulla precisione numerica.
- Impostazioni di partenza e preprocessamento: Permette di modificare il comportamento del risolutore nella fase di preprocessamento del problema.
- Parametri del Simplesso: È possibile impostare la strategia dell'algoritmo di costo del simplesso primale e duale.
- Parametri di Rete: Permette di gestire l'algoritmo del simplesso per problemi di controllo di flusso.
- Parametri MIP: Permette di modificare il comportamento dell'algoritmo di branch and cut.
- Controllo sulla ricerca della soluzione: Permette di impostare un criterio di arresto temporale della simulazione. Consente inoltre di gestire il numero di biforcazioni negli algoritmi di enumerazione implicita.

Appendice B

Codice Matlab

B.1 Codice per l'ottimizzatore 1

B.1.1 Generazione dei dati per l'ottimizzatore 1

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                     Generazione dati per l'ottimizzatore 1           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Vengono generate le matrici necessarie a risolvere il problema di %
%  ottimizzazione e vengono caricate su un file excel                 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all;
tic;
n=input('Inserisci il numero di utenti della rete:');

%Per ottenere matrici pi sparse abbassa la cifra 6 della prossima
%istruzione
%-------------------------------------------------------------------------%
      % Generazione della matrice di adiacenza n*n %
%-------------------------------------------------------------------------%
A=randi(ceil(n/6),n)-((ceil(n/6))-1);
for i=1:n %controlo sulla diagonale
    for j=1:n
        if(i==j)
            A(i,j)=0;
        end
    end
end

%da attivare per fare confronti con l'euristica IMFTP
```

```
%A=randi(30,n)-(20);
%for i=1:n
    %for j=1:n
        %if(i==j)
            %A(i,j)=0;
        %end
    %end
%end

for i=1:n
    for j=1:n
        if(A(i,j)<0)
            A(i,j)=0;
        end
    end
end
%-------------------------------------------------------------------------%

      % Generazione matrice pesata %
%-------------------------------------------------------------------------%
As=A';
Is=ones(n,1);

    Bs=As*Is;
   Bs_2=A*Is;
for i=1:n %controllo nodi disgiunti
    if(Bs(i,1)==0)
        error('Divide by zero_repeat operation')
    end
end

for i=1:n
    if(Bs_2(i,1)==0)
        error('E stata generata una rete
        non valida ripetere operazione ')
    end
end

 for j=1:n
        Ac(:,j)=A(:,j)/Bs(j);
        At=Ac';
    end
```

```
%-------------------------------------------------------------------------%
       % Generazione matrice Lambda %
%-------------------------------------------------------------------------%
Treshold=rand(1,n);
Lambda=diag(Treshold);
%-------------------------------------------------------------------------%
        % Generazione Matrice At+Lambda %
%-------------------------------------------------------------------------%
Af=At+Lambda;

%------------------------Salvataggio dati-----------------------------%

xlswrite('c:\matlab7\work\provaKc.xlsx',Af,'Foglio1');
xlswrite('c:\matlab7\work\ProvaKc.xlsx',Lambda,'Foglio2');


toc;
```

B.1.2 Utenti finali al variare di K e r

```
clear all

load R.dat;
load K.dat;
load Set_finale.dat;
r=R;
k=K;
A=Set_finale;
[m,n]=size(A);

Is=ones(n,1);
Bs=A*Is;


[Z,padded] = vec2mat(Bs,r);
% il grafico con la funzione surf

%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%
X=1:1:r; % range di variazione r
Y=1:1:k; % range di variazione k
```

```
surf(X,Y,Z); % grafico 3d rispetto a r e k

xlabel('r ')
ylabel('k ')
zlabel('|\phi^*_k|')
title('Grafico cardinalit set finale al variare di k e r')
%shading interp;
```

B.2 Codice per l'ottimizzatore 2

B.2.1 Riduzione della rete per l'ottimizzatore 2

```
%Lo script genera una rete e in base al valore di K e al target
% Xd genera una rete ridotta e ne fornisce una rappresentazione
%grafica

clear all;

n=input('Inserisci il numero di utenti della rete:');
K=input('Inserire il numero di  passi k:');
h=input('Inserire il numero di nodi target da generare:');
tic;
e=1;
f=n-1;
r =ceil( e + (f-e).*rand(1,h));
Xd=r;
Target=zeros(1,n);
Target(:,Xd)=1;
Xd=cat(2,0,r);
%-------------------------------------------------------------------------%
   % Generazione della matrice di adiacenza n*n di partenza %
%-------------------------------------------------------------------------%
A=randi(ceil(n/4),n)-((ceil(n/4))-1);
for i=1:n
    for j=1:n
        if(i==j)
            A(i,j)=0;
        end
    end
end

for i=1:n
    for j=1:n
        if(A(i,j)<0)
```

```
            A(i,j)=0;
        end
    end
end

%-------------------------------------------------------------------------%

      % Generazione matrice pesata di partenza %
%-------------------------------------------------------------------------%
As=A';
Is=ones(n,1);

    Bs=As*Is;
    Bs_2=A*Is;
for i=1:n
    if(Bs(i,1)==0)
        error('Divisione per zero ripetere operazione')
    end
end

for i=1:n
    if(Bs_2(i,1)==0)
        error('E stata generata una rete
        non valida ripetere operazione ')
    end
end




B=A;%salvo A originale


for j=1:n
        Ac(:,j)=A(:,j)/Bs(j);
        At=Ac';
    end


%-------------------------------------------------------------------------%
       % Generazione matrice Lambda di partenza %
%-------------------------------------------------------------------------%
```

```
Treshold=rand(1,n);
Lambda=diag(Treshold);
%-------------------------------------------------------------------------%
        % Generazione Matrice At+Lambda di partenza %
%-------------------------------------------------------------------------%
Af=At+Lambda;
%-------------------------------------------------------------------------%



%>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
%>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
%%>>> riduzione della rete in base a k e target <<<<<<<<<<<<

Redu=Xd;

[m,old]=size(Redu);
[p,new]=size(Redu);

for k=1:K

if(k==1)
for i=1:n
    for j=1:new-1
    a=j+1;
    b=Redu(:,a);
    if(A(i,b)==1)

        Redu=cat(2,Redu,setdiff(i,Redu));

    end
end
    end
    old=new;
    [p,new]=size(Redu);


else
    for i=1:n
    for j=1:new-old

    if (A(i,Redu(:,j+old))==1)
        Redu=cat(2,Redu,setdiff(i,Redu));
```

```
    end
end
    end
    old=new;
    [p,new]=size(Redu);
end
end
Subset=union(Xd,Redu);
Subset(:,1)=[];
All_nodes=[1:1:n];
Deleted_nodes=setdiff(All_nodes,Subset);
A(Deleted_nodes,:)=[];
A(:,Deleted_nodes)=[];
Xd(:,1)=[];

Target_Aux=Target;
Target_Aux(:,Deleted_nodes)=[];
Target_r=Target_Aux;

%--------------------------------------------------------------------%
        % Generazione matrice pesata ridotta %
%--------------------------------------------------------------------%


As_r=A';
[N_redu,x]=size(A);
Is_r=ones(N_redu,1);
Bs_r=(As_r*Is_r);




 for j=1:N_redu
        Ac_r(:,j)=A(:,j)/Bs_r(j);
        At_r=Ac_r';
 end

%--------------------------------------------------------------------%
        % Generazione Matrice Lambda ridotta %
%--------------------------------------------------------------------%

 Redu(:,1)=[];
 Treshold_r=Treshold(:,Redu);
 Lambda_r=diag(Treshold_r);
```

```
%-------------------------------------------------------------------%
        % Generazione Matrice At+Lambda ridotta %
%-------------------------------------------------------------------%

 Af_r=At_r+Lambda_r;
%-------------------------------------------------------------------%
                        %Visualizzazione della rete
%-------------------------------------------------------------------%
% Da attivare se si vuole vedere graficamente la riduzione del grafo
%sub=[1:1:N_redu];
%im_1=view(biograph(B));
%set(im_1.Nodes(Deleted_nodes),'Color',[.5 .7 1]);
%set(im_1.Nodes(Xd),'Color',[1 .7 .4]);
%set(im_1.edges,'LineColor',[0  0  0]);
%set(im_1.Nodes,'LineColor',[0  0  0]);
%set(im_1.Nodes(All_nodes),'Shape','circle');
%im_1.NodeAutosize='off';
%set(im_1.Nodes(All_nodes),'Size',[n*4 n*4]);
%dolayout(im_1);
%im_2=view(biograph(A));
%set(im_2.Nodes(sub),'Shape','circle');
%set(im_2.edges,'LineColor',[0  0  0]);
%set(im_2.Nodes,'LineColor',[0  0  0]);
%set(im_1.Nodes(All_nodes),'Size',[n*4 n*4]);
%dolayout(im_2);

%-------------------------------------------------------------------%
                        %Salvataggio dati
%-------------------------------------------------------------------%


xlswrite('c:\matlab7\work\MyDatat.xlsx',Af,'Foglio1');
xlswrite('c:\matlab7\work\MyDatat.xlsx',Lambda,'Foglio2');
xlswrite('c:\matlab7\work\MyDatat.xlsx',Af_r,'Foglio3');
xlswrite('c:\matlab7\work\MyDatat.xlsx',Lambda_r,'Foglio4');
xlswrite('c:\matlab7\work\MyDatat.xlsx',Target,'Foglio5');
xlswrite('c:\matlab7\work\MyDatat.xlsx',Subset,'Foglio6');
xlswrite('c:\matlab7\work\MyDatat.xlsx',Target_r,'Foglio7');
xlswrite('c:\matlab7\work\MyDatat.xlsx',Xd,'Foglio8');

toc;
```

B.2.2 Riduzione della rete e parallelizzazione

```
%Lo script genera una rete e in base al valore di K e al target
% Xd genera una rete ridotta e ne fornisce una rappresentazione
%grafica

clear all;

n=input('Inserisci il numero di utenti della rete:');
K=input('Inserire il numero di  passi k:');
h=input('Inserire il numero di nodi target da generare:');
tic;
e=1;
f=n-1;
r =ceil( e + (f-e).*rand(1,h));
Xd=r;
Target=zeros(1,n);
Target(:,Xd)=1;
Xd=cat(2,0,r);
%-------------------------------------------------------------------------%
   % Generazione della matrice di adiacenza n*n di partenza %
%-------------------------------------------------------------------------%
A=randi(ceil(n/4),n)-((ceil(n/4))-1);
for i=1:n
    for j=1:n
        if(i==j)
            A(i,j)=0;
        end
    end
end

for i=1:n
    for j=1:n
        if(A(i,j)<0)
            A(i,j)=0;
        end
    end
end

%-------------------------------------------------------------------------%

      % Generazione matrice pesata di partenza %
%-------------------------------------------------------------------------%
As=A';
```

```
Is=ones(n,1);

    Bs=As*Is;
    Bs_2=A*Is;
for i=1:n
    if(Bs(i,1)==0)
        error('Divisione per zero ripetere operazione')
    end
end

for i=1:n
    if(Bs_2(i,1)==0)
        error('E stata generata una rete
        non valida ripetere operazione ')
    end
end




B=A;%salvo A originale
C=B;

for j=1:n
        Ac(:,j)=A(:,j)/Bs(j);
        At=Ac';
    end


%-------------------------------------------------------------------------%
       % Generazione matrice Lambda di partenza %
%-------------------------------------------------------------------------%
Treshold=rand(1,n);
Lambda=diag(Treshold);
%-------------------------------------------------------------------------%
        % Generazione Matrice At+Lambda di partenza %
%-------------------------------------------------------------------------%
Af=At+Lambda;
%-------------------------------------------------------------------------%
```

```
%>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
%>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
%%>>> riduzione della rete in base a k e target <<<<<<<<<<<<

Redu=Xd;

[m,old]=size(Redu);
[p,new]=size(Redu);

for k=1:K

if(k==1)
for i=1:n
    for j=1:new-1
    a=j+1;
    b=Redu(:,a);
    if(A(i,b)==1)

        Redu=cat(2,Redu,setdiff(i,Redu));

    end
end
    end
    old=new;
    [p,new]=size(Redu);


else
    for i=1:n
    for j=1:new-old

    if (A(i,Redu(:,j+old))==1)
        Redu=cat(2,Redu,setdiff(i,Redu));

    end
end
    end
    old=new;
    [p,new]=size(Redu);
end
end
Subset=union(Xd,Redu);
Subset(:,1)=[];
All_nodes=[1:1:n];
```

```
Deleted_nodes=setdiff(All_nodes,Subset);
A(Deleted_nodes,:)=[];%riduco la rete di partenza
B(Deleted_nodes,:)=0;
A(:,Deleted_nodes)=[];
B(:,Deleted_nodes)=0;
Xd(:,1)=[];


Target_Aux=Target;
Target_Aux(:,Deleted_nodes)=[];
Target_r=Target_Aux;

%-----------------------------------------------------------------------%
        % Generazione matrice pesata ridotta %
%-----------------------------------------------------------------------%


As_r=A';
[N_redu,x]=size(A);
Is_r=ones(N_redu,1);
Bs_r=(As_r*Is_r);




 for j=1:N_redu
        Ac_r(:,j)=A(:,j)/Bs_r(j);
        At_r=Ac_r';
 end

%-----------------------------------------------------------------------%
        % Generazione Matrice Lambda ridotta %
%-----------------------------------------------------------------------%

 Redu(:,1)=[];
 Treshold_r=Treshold(:,Redu);
 Lambda_r=diag(Treshold_r);

%-----------------------------------------------------------------------%
        % Generazione Matrice At+Lambda ridotta %
%-----------------------------------------------------------------------%

 Af_r=At_r+Lambda_r;
```

```
%-----------------------------------------------------------------------%
              %identificazione paralellizzazione
%-----------------------------------------------------------------------%
sort_vec=[];

for i=1:h
 [sort_vect]=topologically_sort_r(B,Xd(:,i),n);
 sort_vec=union(sort_vec,sort_vect);
end
Insieme=intersect(All_nodes,sort_vec);
parallel=[];
if(isempty(Insieme))
     parallel=1;
 else
     parallel=0;
end

 %-----------------------------------------------------------------------%
                 %Visualizzazione della rete
%-----------------------------------------------------------------------%
% Da attivare se si vuole vedere graficamente la riduzione del grafo
%sub=[1:1:N_redu];
%im_1=view(biograph(C));
%set(im_1.Nodes(Deleted_nodes),'Color',[.5 .7 1]);
%set(im_1.Nodes(Xd),'Color',[1 .7 .4]);
%set(im_1.edges,'LineColor',[0  0  0]);
%set(im_1.Nodes,'LineColor',[0  0  0]);
%set(im_1.Nodes(All_nodes),'Shape','circle');
%im_1.NodeAutosize='off';
%set(im_1.Nodes(All_nodes),'Size',[n*4 n*4]);
%dolayout(im_1);


%-----------------------------------------------------------------------%
                         %Salvataggio dati
%-----------------------------------------------------------------------%


xlswrite('c:\matlab7\work\MyDatat.xlsx',Af,'Foglio1');
xlswrite('c:\matlab7\work\MyDatat.xlsx',Lambda,'Foglio2');
xlswrite('c:\matlab7\work\MyDatat.xlsx',Af_r,'Foglio3');
xlswrite('c:\matlab7\work\MyDatat.xlsx',Lambda_r,'Foglio4');
xlswrite('c:\matlab7\work\MyDatat.xlsx',Target,'Foglio5');
xlswrite('c:\matlab7\work\MyDatat.xlsx',Subset,'Foglio6');
```

```
xlswrite('c:\matlab7\work\MyDatat.xlsx',Target_r,'Foglio7');
xlswrite('c:\matlab7\work\MyDatat.xlsx',Xd,'Foglio8');

toc;
```

B.2.3 Limite inferiore set utenti di partenza

```
clear all
load Set_iniziale_1.dat
load Set_iniziale.dat

Relaxed= Set_iniziale;
Bpp= Set_iniziale_1;

[m,n]=size(Relaxed);
Is=ones(n,1);
Bs_1=Relaxed*Is;
Bs_2=Bpp*Is;

for i=1:m
    Bs_r(i)=ceil(Bs_1(i));
end

plot(Bs_r,'g');
hold on;
plot(Bs_1,'b');
plot(Bs_2,'r');
xlabel('K ');
ylabel('|\phi^*_o|');
grid;
title('Grafico cardinalit set iniziale al variare di k ');
 LEGEND('Ceil LPP','LPP','BPP');
```

B.3 Euristica problema 1

B.3.1 Modulo di controllo

```
% Coordina la simulazione
clear all

n=input('Inserisci il numero di utenti della rete:');
K=input('Inserire orizzonte temporale K:');
r=input('Inserire la cardinalit del set di partenza:');
```

```
%<<<<<<<<<<<<<<<<FASE PREPARATORIA>>>>>>>>>>>>>>>>>>>>>>>


IncInf=zeros(1,n);
ap_v=zeros(n,n);
S=[];
%chiamo l'algoritmo 3 che genera i grafi aciclici
[DAGs,A]= DAGS(n,K);



[Alpha_v,IncInf]=Coefficienti_alpha(DAGs,n);




%<<<<<<<<<<<<<LOOP PRINCIPALE>>>>>>>>>>>>>>>>>>>>>>>

    for iter=1:r

    disp(['Seed numero: ', num2str(iter)]);
     [Y,s] = max(IncInf);

    %Eseguo il loop principale
    [IncInf,ap_v,Alpha_v]=Loop_principale
    DAGs,Alpha_v,ap_v,IncInf,n,s,S);

    S=union(S,s);
    IncInf(:,S)=0;

end
```

B.3.2 Algoritmo di generazione dei grafi aciclici

```
%Data la dimensione n della rete e il parametro K la funzione
%genera i grafi aciclici

function [DAGs,A] =DAGS(n,K)
Tetha=1/80;


%-----------------------------------------------------------------%
```

```
      % Generazione della matrice di adiacenza n*n di partenza %
%-----------------------------------------------------------------%
A=randi(30,n)-(20);
for i=1:n
    for j=1:n
        if(i==j)
            A(i,j)=0;
        end
    end
end

for i=1:n
    for j=1:n
        if(A(i,j)<0)
            A(i,j)=0;
        end
    end
end

%-----------------------------------------------------------------%

 % Generazione matrice pesata di partenza e controllo sulla rete%
%-----------------------------------------------------------------%
As=A';
Is=ones(n,1);

    Bs=As*Is;
    Bs_2=A*Is;
for i=1:n
    if(Bs(i,1)==0)
        error('Divisione per zero ripetere operazione')
    end
end

for i=1:n
    if(Bs_2(i,1)==0)
        error('E stata generata una rete
         non valida ripetere operazione ')
    end
end

%%val=0;
%%for i=1:n
    %%for j=1:n
```

```
        %%if A(j,i)~=0
        %%val=val+1;
        %%end

    %%end
     %%Bs(i,:)=val;
    %%val=0;
%%end


for j=1:n
        Ac(:,j)=A(:,j)/Bs(j);

end


DAGs=cell(n,1);

for d=1:n


    Xd=[0 d];
    Redu=Xd;

%-------------------------------------------------------------------%

% Costruzione dei grafi con nodi distanti k_passi dal root%
%-------------------------------------------------------------------%


[m,old]=size(Redu);
[p,new]=size(Redu);

for k=1:K

if(k==1)
for i=1:n
    for j=1:new-1
    a=j+1;
    b=Redu(:,a);
    if(Ac(i,b)~=0)

        Redu=cat(2,Redu,setdiff(i,Redu));
```

```
    end
end
    end
    old=new;
    [p,new]=size(Redu);


else
    for i=1:n
    for j=1:new-old

    if (Ac(i,Redu(:,j+old))~=0)
        Redu=cat(2,Redu,setdiff(i,Redu));

    end
end
    end
    old=new;
    [p,new]=size(Redu);
end
end


Subset=union(Xd,Redu);
Subset(:,1)=[];

All_nodes=[1:1:n];
Deleted_nodes=setdiff(All_nodes,Subset);
%-----------------------------------------------------------------%

              % Costruzione dei grafi aciclici%
%-----------------------------------------------------------------%
Inf=zeros(1,n);
Aux=Ac;

Aux(Deleted_nodes,:)=0;
Aux(:,Deleted_nodes)=0;

Aux(d,:)=0;
X=d;
Inf(:,d)=1;

Nin=[];
for i=1:n
```

```
    if(Aux(i,d)~=0)
        Nin=cat(2,Nin,i);
        Inf(:,i)=Aux(i,d);
    end
end

[M,size_subset]=size(Subset);

for j=1:size_subset-1
%processing_nodes=setdiff(Subset,X);
Inf_par=Inf;
Inf_par(:,X)=0;o

[N,x]=max(Inf_par);
if(N<Tetha)
    break;
end
All_out=[];
Nout=[];
for g=1:n
    if(Aux(x,g)~=0)
        All_out=cat(2,All_out,g);
    end
end

Nout=intersect(X,All_out);

Aux(x,setdiff(All_out,Nout))=0;
Nin=[];
for f=1:n
    if(Aux(f,x)~=0)
        Nin=union(Nin,f);
        Inf(:,f)=Inf(:,f)+Aux(f,x)*Inf(:,x);
    end
end
%X1=0;
X=cat(2,X,x);
%X1=union(X1,X);
%X1(:,1)=[];%cancello lo zero

end
cut=setdiff(Subset,X);
Aux(:,cut)=0;
Aux(cut,:)=0;
```

```
[v,Size_X]=size(X);
X=cat(2,X,zeros(1,n-Size_X));



Aux=cat(2,Size_X*ones(n,1),Aux);
Aux=cat(2,X',Aux);


DAGs{d,:}=Aux;

end
```

B.3.3 Generazione dei coefficienti α

```
% Dati i grafi aciclici la funzione genera:
% 1)I coefficienti alpha per la fase preliminare
% 2)L'indice di influenza di ciascun nodo
function [Alpha_v,IncInf]=Coefficienti_alpha(DAGs,n)

Alpha_v=zeros(n,n);
IncInf=zeros(1,n);
for d=1:n
    Alpha_dag=zeros(1,n);

    Alpha_dag(:,d)=1;

    dag=DAGs{d,:};
    Size_dag=dag(1,2);
    F=dag(:,3:end);
    s=d;
    [Sort_vect]=topologically_sort_r(F,s,n);

    Rho=Sort_vect;
    Rho(:,1)=[];

    Rho=Rho';
    rho=[];

    for i=1:Size_dag-1
        rho=Rho(i,:);
        Alpha_dag(:,rho)=dag(rho,3:end)*Alpha_dag';
    end
```

```
    Alpha_v(d,:)=Alpha_dag;

    for j=1:n
        IncInf(:,j)=sum(Alpha_v(:,j));
    end
end
```

B.3.4 Modulo loop principale

```
%Dati i grafi aciclici, i coefficienti alpha, le probabilit di attivazione
%l'incremento di influenza,il nuovo innovatore se i precedenti innovatori S
%aggiorna:
% 1)L'incremento di influenza IncInf
% 2)Le probabilit di attivazione
% 3)I coefficienti alpha
function [IncInf,ap_v,Alpha_v]=Loop_principale(
DAGs,Alpha_v,ap_v,IncInf,n,s,S)

name=[];
for i=1:n
    dag=DAGs{i,:};
    dim=dag(1,2);
    seq=dag(1:dim,1);
    Intersect=intersect(seq,s);

    if(~isempty(Intersect))
        name=union(name,i);
    end
end

S_plus_s=union(s,S);
[f,size_S_plus]=size(S_plus_s);
name=setdiff(name,S_plus_s);


[N,dim_name]=size(name);
for i=1:dim_name




Delta_alpha=zeros(1,n);
%Delta_alpha=Alpha_v(name(:,i),:);
```

```
Delta_alpha(s)=-Alpha_v(name(:,i),s);
Delta_alpha(:,S)=0;


g=DAGs{name(:,i),:};
Size_dag=g(1,2);

F=g(:,3:end);
[Sort_vect]=topologically_sort_r(F,s,n);


[z,Size_sort_vect]=size(Sort_vect);
ind=[];

for f=1:size_S_plus
    for d=1:Size_sort_vect
    if(Sort_vect(:,d)==S_plus_s(:,f))
       ind=union(ind,d);
    end
  end
end

Rho=Sort_vect;
Sort_vect(:,ind)=[];
[e,Size_seq]=size(Sort_vect);
Seq=Sort_vect;



rho=[];

for l=1:Size_seq
      Nout=[];
      rho=Seq(:,l);
      for t=1:n
          if(g(rho,t+2)~=0)
              Nout=union(Nout,t);
          end
          Nout=intersect(Nout,Rho);
      end
      alpha_aux=Delta_alpha;
      alpha_aux(:,setdiff(Rho,Nout))=0;
      W=g(rho,3:end);
      W(:,setdiff(Rho,Nout))=0;
```

```
      Delta=W*alpha_aux';
      Delta_alpha(:,rho)=Delta;
      Alpha_v(name(:,i),rho)=Alpha_v(name(:,i),rho)+Delta;
      IncInf(:,rho)=IncInf(:,rho)+(1-ap_v(name(:,i),rho))*Delta;

end
Delta_alpha_aux=Delta_alpha;
Delta_alpha_aux(:,Seq)=0;
Alpha_v(name(:,i),:)=Alpha_v(name(:,i),:)+Delta_alpha_aux;



Delta_alpha_aux=Delta_alpha.*(1-ap_v(name(:,i),:));
Delta_alpha_aux(:,Seq)=0;
Inf_aux=Delta_alpha_aux;
IncInf=IncInf+Inf_aux;




 Delta_ap=zeros(1,n);
 %Delta_ap=ap_v(name(:,i),:);
 Delta_ap(:,s)=1-ap_v(name(:,i),s);
 Delta_ap(:,S)=0;

 [Sort_vect]=topologically_sort(F,s,n);

  ind=[];
  [z,Size_sort_vect]=size(Sort_vect);


for f=1:size_S_plus
    for d=1:Size_sort_vect
    if(Sort_vect(:,d)==S_plus_s(:,f))
       ind=union(ind,d);
    end
  end
end

Rho=Sort_vect;
Sort_vect(:,ind)=[];
[e,Size_seq]=size(Sort_vect);
Seq=Sort_vect;
```

```
rho=[];
  for l=1:Size_seq
      Nin=[];
      rho=Seq(:,l);
      for t=1:n
          if(g(t,rho+2)~=0)
              Nin=union(Nin,t);
          end
          Nin=intersect(Nin,Rho);
      end
      ap_aux=Delta_ap;

      ap_aux(:,setdiff(Rho,Nin))=0;
      W=g(:,rho+2);
      W(setdiff(Rho,Nin),:)=0;

      Delta=ap_aux*W;
      Delta_ap(:,rho)=Delta;
      ap_v(name(:,i),rho)=ap_v(name(:,i),rho)+Delta;
      IncInf(:,rho)=IncInf(:,rho)-Alpha_v(name(:,i),rho)*Delta;

  end

  Delta_ap_aux=Delta_ap;
  Delta_ap_aux(:,Seq)=0;
  ap_v(name(:,i),:)=ap_v(name(:,i),:)+Delta_ap_aux;

Delta_ap_aux=Delta_ap.*(Alpha_v(name(:,i),:));
Delta_ap_aux(:,Seq)=0;
Inf_aux=-Delta_ap_aux;
IncInf=IncInf+Inf_aux;

  end
```

B.3.5 Ordinamento topologico dei nodi raggiungibili dal seed

```
function [Sort_vect]=topologically_sort(F,s,n)

%la funzione genera una sequenza di nodi raggiungibili dal seed

Xd=[0 s];
Redu=Xd;

[m,old]=size(Redu);
[p,new]=size(Redu);


for k=1:n
if(k==1)
for i=1:n
    for j=1:new-1
    a=j+1;
    b=Redu(:,a);
    if(F(b,i)~=0)
        Redu=cat(2,Redu,setdiff(i,Redu));

    end
end
    end
    old=new;
    [p,new]=size(Redu);


else
    for i=1:n
    for j=1:new-old

    if (F(Redu(:,j+old),i)~=0)
        Redu=cat(2,Redu,setdiff(i,Redu));

    end
end
    end
    old=new;
    [p,new]=size(Redu);
end
end
```

```
Sort_vect=Redu(:,2:end);
```

B.3.6 Ordinamento topologico nodi che raggiungono s

```
function [Sort_vect]=topologically_sort_r(F,s,n)

%la funzione genera una sequenza di nodi che  raggiungono  il seed

Xd=[0 s];
Redu=Xd;

[m,old]=size(Redu);
[p,new]=size(Redu);


for k=1:n
if(k==1)
for i=1:n
    for j=1:new-1
    a=j+1;
    b=Redu(:,a);
    if(F(i,b)~=0)

        Redu=cat(2,Redu,setdiff(i,Redu));

    end
end
    end
    old=new;
    [p,new]=size(Redu);


else
    for i=1:n
    for j=1:new-old

    if (F(i,Redu(:,j+old))~=0)
        Redu=cat(2,Redu,setdiff(i,Redu));

    end
end
```

```
    end
    old=new;
    [p,new]=size(Redu);
end
end


Sort_vect=Redu(:,2:end);
```

B.4 Dati set coesivo massimale

```
%Lo script genera i dati per l'ottimizzatore sul set coesivo massimale%
clear all;

n=input('Inserisci il numero di utenti della rete:');
h=input('Inserire la cardinalit del seed set:');
tic;
e=1;
f=n-1;
r =ceil( e + (f-e).*rand(1,h));% genero k numeri nel range di r
Target=zeros(1,n);
Target(:,r)=1;
y=Target;
%-------------------------------------------------------------------------%
  % Generazione della matrice di adiacenza n*n di partenza %
%-------------------------------------------------------------------------%
%Per ottenere matrici pi sparse abbassa la cifra 5 della prossima
%istruzione

A=randi(ceil(n/5),n)-((ceil(n/5))-1);
for i=1:n %controlo sulla diagonale
    for j=1:n
        if(i==j)
            A(i,j)=0;
        end
    end
end

for i=1:n %elimino i valori negativi
    for j=1:n
        if(A(i,j)<0)
```

```
            A(i,j)=0;
        end
    end
end

%--------------------------------------------------------------------%

      % Generazione matrice pesata di partenza %
%--------------------------------------------------------------------%
As=A';
Is=ones(n,1);

    Bs=As*Is;
    Bs_2=A*Is;
for i=1:n
    if(Bs(i,1)==0)
        error('Divisione per zero ripetere operazione')
    end
end

for i=1:n
    if(Bs_2(i,1)==0)
        error('E stata generata una rete
        non valida ripetere operazione ')
    end
end


B=A;
for j=1:n
        Ac(:,j)=A(:,j)/Bs(j);
        At=Ac';
    end


%--------------------------------------------------------------------%
       % Generazione matrice Lambda di partenza %
%--------------------------------------------------------------------%
Treshold=rand(1,n);
Lambda=diag(Treshold);
%--------------------------------------------------------------------%
        % Generazione Matrice I-Lambda-At di partenza %
%--------------------------------------------------------------------%
Af=ones(n,n)-Lambda-At;
```

```
%-------------------------------------------------------------------%


%-------------------------------------------------------------------%
                          %Salvataggio dati
%-------------------------------------------------------------------%
xlswrite('c:\matlab7\work\ProvaKc.xlsx',Af,'Foglio1');
xlswrite('c:\matlab7\work\ProvaKc.xlsx',y,'Foglio2');
```

Appendice C

Codice OPL

C.1 Simulatore problema 1

C.1.1 Modulo di controllo di flusso

```
 main{

var KTests =10;
var rtest=10;
var source;
var def;
var theModel;

var ofiler=new IloOplOutputFile(" R.dat");
var ofilek=new IloOplOutputFile(" K.dat");
var ofile2 = new IloOplOutputFile("Set_finale.dat");
var ofile3 = new IloOplOutputFile("Set_iniziale.dat");
ofiler.write(rtest);/*salvo k e r massimi*/
ofilek.write(KTests);

    for (var i=1; i<= KTests; i++){

var ofile1=new IloOplOutputFile("K="+i+ "Vettore_evoluzione.dat");

writeln ("valore attuale K=",i);

for(var j=1; j<=rtest; j++){
writeln ("valore attuale r=",j);

source = new IloOplModelSource("sub_module.mod");
def = new IloOplModelDefinition(source);
theModel = new IloOplModel(def,cplex);
```

```
var ofilen = new IloOplOutputFile(i+".dat");
        ofilen.writeln("K=",i,";");

var data = new IloOplDataSource(i+".dat");
var data1= new IloOplDataSource("Matrix.dat");

theModel.addDataSource(data);
theModel.addDataSource(data1);

theModel.generate();

        theModel.ctone.UB=j;
/*<<<<<<<<<<<<<<<<<<<<ISTANZA COMPLETATA>>>>>>>>>>*/


 cplex.solve(); /*SOLVE THE PROBLEM*/
/*ofile1.writeln(cplex.getObjValue());*/

 /*Salvo per ogni valore di k tutto il vettore x*/

 ofile1.writeln(" ");
 for ( var  a  in  theModel.Cols )
 {
   ofile1.write(theModel.x[a]," ");
 }

 /*La configurazione finale  la seguente*/

 ofile2.writeln(" ");

 for ( var  c in  theModel.Last_elm )

 {
  ofile2.write(theModel.x[c]," ");
 }

/*la configurazione iniziale  la seguente*/

 ofile3.writeln(" ");
 for ( var  b  in  theModel.Colonne )
 {
```

```
   ofile3.write(theModel.x[b]," ");
 }
}
}
}
```

C.1.2 Modello del problema

```
int  r=...; /*cardinalit r*/
int  K=...;
int  n=30; /*Da impostare manualmente*/
int  N=n*(K+1);/***E' la dimensione n(k+1)***/
int  N_minus_n=N-(n-1);


range   Par  =1..K ;
range   Cols  =  1 .. N ;
range   Righe =  1..n;
range   Colonne= 1..n;
range   Last_elm =N_minus_n..N ;


float I_O[Cols];
float INK[Cols];
float  A [ Righe , Colonne ]  =   ... ;
float  Lambda [ Righe , Colonne ]  =   ... ;

/********Generare la I_O*******************/

execute  {
for(var  i  in  Cols)

  if(i<=n)

I_O[i] =1;
else
I_O[i]=0;
}
/****************************************/
/********Generare la INK*******************/

execute  {
for(var  i  in  Cols)
if(i<=n*K)
```

```
INK[i] =1;
else
INK[i]=n*K;
}

dvar   boolean  x [ Cols ] ;

maximize
sum ( j  in  Cols )   INK[ j ] * x [ j ] ;

subject  to
 {

    ctone:
    sum ( j  in  Cols )   I_O[ j ] * x [ j ] == r;

    forall(f in Par)
    forall(i in Righe)
    cttwo:
    sum(p in Colonne)  A[i][p]*x[(n*(f-1)+1)+(p-1)] >=
    sum(p in Colonne) Lambda[i][p]*x[(n*(f-1)+1)+(p-1+n)] ;
  }
```

C.1.3 Ridefinizione della matrice Lambda

```
int  r=...; /*cardinalit r*/
int  K=...;
int  n=7; /*Da impostare manualmente*/
int  N=n*(K+1);/***E' la dimensione n(k+1)***/
int  N_minus_n=N-(n-1);
/*int  rmax=n-1;*/

range   Par  =1..K ;
range   Cols  =  1 .. N ;
range   Righe =  1..n;
range   Colonne= 1..n;
range   Last_elm =N_minus_n..N ;


float I_O[Cols];
float INK[Cols];
float  A [ Righe , Colonne ]  =   ... ;
```

```
float  Lambda [ Colonne ]  =   ... ;

/********Generare la I_O*********************/

execute  {
for(var  i  in  Cols)

  if(i<=n)

I_O[i] =1;
else
I_O[i]=0;
}
/*****************************************/
/********Generare la INK*********************/

execute  {
for(var  i  in  Cols)
if(i<=n*K)
INK[i] =1;
else
INK[i]=n*K;
}

dvar   boolean  x [ Cols ] ;

maximize
sum ( j  in  Cols )   INK[ j ] * x [ j ] ;

subject  to
 {


ctone:
sum ( j  in  Cols )   I_O[ j ] * x [ j ] == r;


forall(f in Par)
forall(i in Righe)

cttwo:
sum(p in Colonne) A[i][p]*x[(n*(f-1)+1)+(p-1)] >=
Lambda[i]*x[(n*(f-1)+(i+n))] ;
```

```
}
```

C.1.4 Dichiarazione dei dati

```
r=1;
SheetConnection sheet("Rete_4.xlsx");

A from SheetRead(sheet,"'Foglio1'!A1:AD30");
Lambda from SheetRead(sheet,"'Foglio2'!A1:AD30");
```

C.2 Simulatore problema 2

C.2.1 Modulo di controllo di flusso

```
main {


    var KTests =10;
var source;
var def;
var theModel;
var ofile1 = new IloOplOutputFile("Set_iniziale.dat");
    var ofile2 = new IloOplOutputFile("Set_finale.dat");



for (var i=1; i<= KTests; i++){
writeln ("valore attuale K=",i);
var ofile3=new IloOplOutputFile("K="+i+ "Vettore_evoluzione.dat");

source = new IloOplModelSource("Sub_Model.mod");
def = new IloOplModelDefinition(source);
theModel = new IloOplModel(def,cplex);


var ofilen = new IloOplOutputFile(i+".dat");
ofilen.writeln("K=",i,";");

var data = new IloOplDataSource(i+".dat");
var data1= new IloOplDataSource("A_+_Lambda.dat");
```

```
theModel.addDataSource(data);
theModel.addDataSource(data1);

theModel.generate();/*creo una nuova istanza del modello*/
/*<<<<<<<<<<<<<<<<<<ISTANZA COMPLETATA>>>>>>>>>>>>>>>>>>>>>*/


 cplex.solve(); /*SOLVE THE PROBLEM*/

 /*Il set di partenza ottimale */
 /*ofile1.writeln(cplex.getObjValue())*/


 /*<<<<<<<<<<<<<<<<<<<<<<SAVE RESULT>>>>>>>>>>>>>>>>>>>>>>*/

 /*Salvo per ogni valore di k tutto il vettore x*/
 ofile3.writeln(" ");
 for ( var  a  in  theModel.Cols )
 {
   ofile3.write(theModel.x[a]," ");
 }


 /*la configurazione iniziale  la seguente*/

 ofile1.writeln(" ");
 for ( var  b  in  theModel.Colonne )
 {

   ofile1.write(theModel.x[b]," ");
 }


 /*La configurazione finale  la seguente*/

 ofile2.writeln(" ");
 for ( var  c in  theModel.Last_elm )

 {
  ofile2.write(theModel.x[c]," ");
 }


}
```

```
}
```

C.2.2 Modello del problema

```
int  K=...;
int  n=30;
int  N =n*(K+1) ;
int  N_minus_n=N-(n-1);

/****DICHIARAZIONE DIMENSIONE STRUTTURE DATI*************/

range   Par  =1..K ;
range   Cols  =  1 .. N ;
range   Righe =  1..n;
range   Colonne= 1..n;
range   Last_elm =N_minus_n..N ;
/*range   One=1..1;*/
/*******DICHIARAZIONE STRUTTURE DATI PRINCIPALI*********/
int  Xd[Righe]=...;
/*float I_O[Cols];*/
float INK[Cols];
float  Lambda [ Righe , Colonne ]=...;
/***A+ lambda la dichiaro pi avanti***/

/*****************CARICAMENTO DATI*****************/


/********Generare la INK*****************************/

execute  {
for(var  i  in  Cols)
if(i<=n)
INK[i] =1;
else
INK[i]=0;
}



float A[ Righe, Colonne]=...;
```

```
/*<<<MODELLO PROBLEMA DI PROGRAMMAZIONE BINARIO>>>*/

dvar   boolean  x [ Cols ] ;

minimize
sum ( j  in  Cols )   INK[ j ]*x [ j ] ;

subject  to
 {

  forall(l in Last_elm)
    ct:
    x[l]>=Xd[l-n*K];

   forall(f in Par)
   forall(i in Righe)

cttwo:
sum(p in Colonne) A[i][p]*x[(n*(f-1)+1)+(p-1)] >=
sum(p in Colonne) Lambda[i][p]*x[(n*(f-1)+1)+(p-1+n)] ;
  }
```

C.2.3 versione rilassata

```
int  K=...;
int  n=30;
int  N =n*(K+1) ;
int  N_minus_n=N-(n-1);

/****DICHIARAZIONE DIMENSIONE STRUTTURE DATI*********/

range   Par  =1..K ;
range   Cols  =  1 .. N ;
range   Righe =  1..n;
range   Colonne= 1..n;
range   Last_elm =N_minus_n..N ;
/*range   One=1..1;*/
/*******DICHIARAZIONE STRUTTURE DATI PRINCIPALI*******/
int  Xd[Righe]=...;
/*float I_O[Cols];*/
float INK[Cols];
float  Lambda [ Righe , Colonne ]=...;
/***A+ lambda la dichiaro pi avanti***/
```

```
/*****************CARICAMENTO DATI*******************/


/********Generare la INK*****************************/

execute  {
for(var  i  in  Cols)
if(i<=n)
INK[i] =1;
else
INK[i]=0;
}



float A[ Righe, Colonne]=...;


/*<<<<<MODELLO PROBLEMA DI PROGRAMMAZIONE BINARIO>>>>*/

dvar  float+ x [ Cols ] ;

minimize
sum ( j  in  Cols )   INK[ j ]*x [ j ] ;

subject  to
 {

  forall(l in Last_elm)
    ct:
    x[l]>=Xd[l-n*K];

forall(f in Par)
   forall(i in Righe)

cttwo:
sum(p in Colonne) A[i][p]*x[(n*(f-1)+1)+(p-1)] >=
sum(p in Colonne) Lambda[i][p]*x[(n*(f-1)+1)+(p-1+n)];


}
\\
```

C.2.4 Ridefinizione della matrice Lambda

```
/*****************DICHIARAZIONE DELLE VARIABILI***********/
int  K=...;     /***E'il numero di passi dell'algoritmo***/
int  n=7;
int  N =n*(K+1) ;   /***E' la dimensione n(k+1)***/
int  N_minus_n=N-(n-1);

/******DICHIARAZIONE DIMENSIONE STRUTTURE DATI*****/

range   Par  =1..K ;
range   Cols  =  1 .. N ;
range   Righe =  1..n;
range   Colonne= 1..n;
range   Last_elm =N_minus_n..N ;
/*range   One=1..1;*/
/***DICHIARAZIONE STRUTTURE DATI PRINCIPALI********/
int  Xd[Righe]=...;
/*float I_O[Cols];*/
float INK[Cols];
float  Lambda [ Righe ]=...;
/***A+ lambda la dichiaro pi avanti***/

/*****************CARICAMENTO DATI*****************/


/********Generare la INK**************************/

execute  {
for(var  i  in  Cols)
if(i<=n)
INK[i] =1;
else
INK[i]=0;
}



float A[ Righe, Colonne]=...;




/*<<MODELLO PROBLEMA DI PROGRAMMAZIONE BINARIO>>>>*/
```

```
dvar   boolean  x [ Cols ] ;

minimize
sum ( j  in  Cols )   INK[ j ]*x [ j ] ;

subject  to
 {

  forall(l in Last_elm)
    ct:
    x[l]>=Xd[l-n*K];

forall(f in Par)
   forall(i in Righe)

cttwo:
sum(p in Colonne) A[i][p]*x[(n*(f-1)+1)+(p-1)] >=
Lambda[i]*x[(n*(f-1)+(i+n))] ;
}
```

C.2.5 Dichiarazione dei dati

```
SheetConnection sheet("test_preliminare.xlsx");

A from SheetRead(sheet,"'Foglio1'!A1:AD30");
Lambda from SheetRead(sheet,"'Foglio2'!A1:AD30");
Xd from SheetRead(sheet,"'Foglio5'!A1:AD1");
```

C.3 Determinazione set coesivo massimale

C.3.1 Modello del problema binario

```
int n=50;/*Dimensione del problema*/

range Dim=1..n;/*per dimensionare i vettori*/

int y[Dim]=...;/*vettore caratteristico del seed*/
int I[Dim];/*vettore identita'*/
float  A [ Dim , Dim ]= ... ;
```

```
/*Genero il vettore identit*/
execute  {
for(var  i  in  Dim)
I[i] =1;
}

dvar  boolean x [ Dim ];

maximize
sum ( j  in Dim )   I[ j ] * x [ j ] ;

subject to {
  /*Dichiarazione dei vincoli*/
  forall(l in Dim)
  ct1:
  x[l]<=I[l]-y[l];


  forall ( i  in  Dim )
  ct2:
  sum ( j  in  Dim )   A [ i ] [ j ] * x [ j ]  <=0 ;
}
execute{

var ofile1 = new IloOplOutputFile("Set_finale.dat");

/*Il set di partenza ottimale */
writeln (" Il set finale  : ") ;
 write (" x = [ " ) ;

 ofile1.writeln(" ");

 for ( var  a  in  Dim )
 {
 write ( x [ a] , "  " ) ;
 ofile1.write(x[a]," ");
 }

}
```

C.3.2 Dichiarazione dei dati

```
SheetConnection sheet("Net_6.xlsx");
```

```
A from SheetRead(sheet,"'Foglio1'!A1:AX50");
y from SheetRead(sheet,"'Foglio2'!A1:F1");
```

C.3.3 Modulo di controllo versione rilassata

```
main{

var source;
var def;
var theModel;
var Curr;
var i=1;
source = new IloOplModelSource("Sub_Model_1.mod");
def = new IloOplModelDefinition(source);
theModel = new IloOplModel(def,cplex);

var ofilen = new IloOplOutputFile(i+".dat");

var data = new IloOplDataSource(i+".dat");
var data1= new IloOplDataSource("Matrix.dat");

theModel.addDataSource(data);
theModel.addDataSource(data1);
theModel.generate();

cplex.solve(); /*SOLVE THE FIRST PROBLEM*/
i=i+1;
var ofilel = new IloOplOutputFile(i+".dat");

ofilel.writeln("y=[");

 for ( var  a in  theModel.Dim)

 {
  ofilel.write(theModel.x[a]," ");
 }
 ofilel.writeln("];");

Curr=cplex.getObjValue();



while(Math.ceil(Curr)!=Curr)
```

```
{
source = new IloOplModelSource("Sub_Model_K.mod");
def = new IloOplModelDefinition(source);
theModel = new IloOplModel(def,cplex);

var ofiler = new IloOplOutputFile(i+".dat");

var data3 = new IloOplDataSource(i+".dat");
var data4= new IloOplDataSource("Matrix.dat");

theModel.addDataSource(data3);
theModel.addDataSource(data4);
theModel.generate();

cplex.solve(); /*SOLVE THE PROBLEM*/
i=i+1;

Curr=cplex.getObjValue();

var ofiles = new IloOplOutputFile(i+".dat");
ofiles.writeln("y=[");

 for ( var  b in  theModel.Dim)

 {
  ofiles.write(theModel.x[b]," ");
 }
 ofiles.writeln("];");
}
}
```

C.3.4 Modello prima iterazione

```
int n=6;/*Dimensione del problema*/

range Dim=1..n;/*per dimensionare i vettori*/

int y[Dim]=...;/*vettore caratteristico del seed*/
float I[Dim];/*vettore identita'*/
float  A [ Dim , Dim ]= ... ;

/*Genero il vettore identit*/
execute  {
```

```
for(var  i  in  Dim)
I[i] =1;
}

/*Bound del primo vincolo*/

float b[i in Dim]=I[i]-y[i];


dvar   float+ x [ Dim ];

maximize
sum ( j  in Dim )   I[ j ] * x [ j ] ;

subject to {
  /*Dichiarazione dei vincoli*/
  forall(l in Dim)
  ct1:
  x[l]<=b[l];


  forall ( i  in  Dim )
  ct2:
  sum ( j  in  Dim )   A [ i ] [ j ] * x [ j ]  <=0 ;
}
```

C.3.5 Modello iterazioni successive

```
int n=6;/*Dimensione del problema*/

range Dim=1..n;/*per dimensionare i vettori*/

float y[Dim]=...;
float I[Dim];
float  A [ Dim , Dim ]= ... ;

float b[Dim];

/*Genero il vettore identit*/
execute  {
for(var  i  in  Dim)
I[i] =1;
}
```

```
float C[p in Dim]=ceil(I[p]-y[p]);



execute  {
for(var  f  in  Dim)

 b[f]=I[f]-C[f];

}


dvar   float+ x[ Dim ];

maximize
sum ( j  in Dim )   I[ j ] * x [ j ] ;

subject to {
  /*Dichiarazione dei vincoli*/
  forall(l in Dim)
  ct1:
  x[l]<=b[l];


  forall ( i  in  Dim )
  ct2:
  sum ( j  in  Dim )   A [ i ] [ j ] * x [ j ]  <=0 ;
}
```

C.3.6 Dichiarazione dati

```
SheetConnection sheet("Mydata.xlsx");

A from SheetRead(sheet,"'Foglio1'!A1:F6");
```

Bibliografia

[1] M.Granovetter. "Treshold models of collective behavior", *American journal of sociology*, pp.1420-1443, 1978.

[2] T. Schelling, "C.(1978). micromotives and macrobehavior."

[3] G. Ellison, "Learning, local interaction, and coordination," *Econometrica: Journal of the Econometric Society*, pp. 10471071, 1993.

[4] L. Blume et al., "The statistical mechanics of strategic interaction," *Games and economic behavior*, vol. 5, no. 3, pp. 387424, 1993.

[5] S. Morris, "Contagion," *The Review of Economic Studies*, vol. 67, no. 1, p. 57, 2000.

[6] J. Wortman, "Viral marketing and the diffusion of trends on social networks," 2008.

[7] D. Acemoglu, A. Ozdaglar, and E. Yildiz, "Diffusion of innovations in social networks," in 50th IEEE *Conference on Decision and Control*, 2011.

[8] D. Kempe, J. Kleinberg, and É. Tardos, "Maximizing the spread of influence through a social network," *in Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM, 2003, pp. 137146.

[9] P. Domingos and M. Richardson, Mining the network value of customers," *in Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM, 2001, pp. 5766.

[10] M. Richardson and P. Domingos, "Mining knowledge-sharing sites for viral marketing," *in Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM, 2002, pp. 6170.

[11] W. Chen, Y. Yuan, and L. Zhang, "Scalable influence maximization in social networks under the linear threshold model," *in Data Mining (ICDM), 2010 IEEE 10th International Conference on.* IEEE, 2010, pp. 8897.

[12] W. Chen, C. Wang, and Y. Wang, "Scalable influence maximization for prevalent viral marketing in large-scale social networks," *in Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM, 2010, pp. 10291038.

[13] E. Rogers, *Diffusion of Innovations*. New York: Free Press, 1995.

[14] Zuddas P.(2004)*Ricerca operativa*. Dipartimento di Ingegneria idraulica. Università di Cagliari

[15] Malucelli F.*Appunti di introduzione alla ricerca operativa*. Dipartimento di elettronica e informazione. Politecnico di Milano.

[16] ILOG CPLEX 11.2.

[17] IBM ILOG OPL From or to OPL and ODM.

[18] IBM ILOG OPL Language user's manual.

[19] IBM ILOG OPL Language reference manual.

[20] F. M. Bass. A new product growth for model consumer durables. *Management Science*, 15(5):215227, 1969.

[21] J. J. Brown and P. H. Reingen. Social ties and word-of-mouth referralbehavior. *The Journal of Consumer Research*, 14(3):350362, 1987.

[22] V. Mahajan, E. Muller, and F. M. Bass. New product diffusion models in marketing: A review and directions for research. *The Journal of Marketing*, 54(1):126, 1990.

[23] D.Rosa, A. Giua. *On the spread of innovation in social network*. 51st IEEE Conference on Decision and Control , CDC 2012.

[24] Jacob Goldenberg, Barak Libai, and Eitan Muller. Using complex systems analysis to advance marketing theory development: Modeling heterogeneity effects on new product growth through stochastic cellular automata. *Academy of Marketing Science Review*, [Online] 1(9), 2001.

[25] Charu C.Aggarval(2011)-*Social network data analytics*, Springer, New York, pp502.