Alma Mater Studiorum - Università di Bologna

SCUOLA DI INGEGNERIA E ARCHITETTURA
Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

Analisi e sviluppo di un sistema per l'ottimizzazione di reti publish/subscribe

Relatore:
Chiar.mo Prof.
Vittorio Maniezzo

Presentata da:
Giovanni Mormone

Sessione II
2020/2021

Abstract

Lo sviluppo di tecnologie che sfruttano il paradigma *Internet of Things*(IoT) ha portato all'utilizzo sempre più diffuso di reti formate da diversi *devices* disseminati nel territorio, con il paradigma *Publish/Subscribe* come uno dei paradigmi di comunicazione più utilizzati per applicazioni di questo tipo. L'adozione di questi sistemi porta però anche al sorgere di vari problemi e limitazioni, ad esempio dovuti alla limitata capacità delle batterie dei *devices* utilizzati, alla limitata capacità di banda della connessione all'interno della rete o alle limitate capacità di memoria e di numero di connessioni che è possibile gestire all'interno della rete. Questo lavoro si concentra sullo sviluppo di un sistema in grado di trovare la configurazione di rete più efficiente per permettere ad un'applicazione IoT che sfrutta il paradigma *Publish/Subscribe* di massimizzare il numero di connessioni e ridurre lo spreco di risorse all'interno di essa, a partire dalla definizione dei *devices* e delle connessioni tra gli stessi presenti nel sistema. Per raggiungere lo scopo viene sfruttata la programmazione lineare intera, proponendo delle formulazioni matematiche in grado di risolvere ed ottimizzare il problema della configurazione di rete in contesti dove le risorse a disposizione dei *devices* sono limitate.

Keywords: Publish/Subscribe, Internet of Things, Programmazione Lineare Intera, pub/sub brokers

Indice

Elenco delle figure

Elenco delle tabelle

Introduzione

Lo sviluppo di tecnologie che sfruttano il paradigma *Internet of Things*(IoT) ha portato all'utilizzo di reti formate da diversi *devices* disseminati nel territorio sempre più diffuso. Negli ultimi anni infatti reti formate da *devices* distribuiti nel territorio hanno acquisito sempre più importanza, con lo scopo di aiutare gli utenti a prendere decisioni in fretta quando utilizzate in sistemi ed applicazioni per la gestione di emergenze, come ad esempio servizi di monitoraggio di eventi meteorologici estremi, sistemi per monitoraggio di terremoti oppure sistemi per il monitoraggio di foreste alla ricerca di possibili incendi [38]. In questo tipo di applicazioni l'obbiettivo della disseminazione e distribuzione dei dati e dei sensori è quello di notificare gli utenti riguardo ai dati a cui sono interessati in tempi brevi [38].

Data la natura distribuita di questo tipo di sistemi, è necessario definire dei paradigmi per mettere in comunicazione i vari *devices* che compongono questo tipo di reti. Uno dei paradigmi di comunicazione più utilizzati per applicazioni di questo tipo è il paradigma *Publish/Subscribe*, paradigma che è stato oggetto negli ultimi anni ad una sempre maggiore attenzione come paradigma di comunicazione adatto a disseminare informazioni su sistemi distribuiti con area di utilizzo molto ampia [20], dato che grazie alla sue caratteristiche di *Time, Space e Flow Decoupling* può supportare in maniera efficiente un complesso *event matching* e permette al sistema di essere esteso ed utilizzato su ampia scala [38].

Tuttavia l'adozione di sistemi IoT oltre a dei vantaggi porta anche al sorgere di vari problemi e limitazioni. Alcuni dei problemi principali di questo

tipo di approcci sono la limitata capacità delle batterie dei *devices* utilizzati, che non rende efficiente mandare dati ed aprire connessioni con un grande numero di nodi che compongono la rete [37], la limitata capacità di banda della connessione all'interno della rete o le limitate capacità di memoria e del numero di connessioni che è possibile gestire all'interno della rete. La gestione e lo sviluppo di tecniche e sistemi che risolvano questi problemi sono dunque fondamentali quando si progettano applicazioni e reti costruite sfruttando tecniche IoT.

Questo lavoro si concentra sullo sviluppo di un sistema in grado di trovare la configurazione di rete più efficiente per permettere ad un'applicazione IoT di massimizzare il numero di connessioni e ridurre lo spreco di risorse all'interno di essa, a partire dalla definizione dei *devices* e delle connessioni tra gli stessi presenti in un sistema. Per raggiungere lo scopo viene sfruttata la programmazione lineare intera, proponendo delle formulazioni matematiche in grado di risolvere ed ottimizzare il problema della configurazione di rete in contesti dove le risorse a disposizione dei *devices* sono limitate.

Questa tesi è organizzata nel seguente modo: nel capitolo 1 viene effettuata un'introduzione del paradigma *Publish/Subscribe*, nel capitolo 2 viene descritto il sistema di partenza per lo sviluppo del lavoro e le tecnologie utilizzate da esso, nel capitolo 3 viene introdotta la programmazione lineare e vengono proposte delle formulazioni matematiche per risolvere il problema, nel capitolo 4 viene descritto il sistema implementato e le tecnologie utilizzate per costruirlo e nel capitolo 5 vengono presentati alcuni casi d'uso del sistema implementato. Infine sono presenti le conclusioni ed i possibili sviluppi futuri del lavoro svolto.

Capitolo 1

Paradigma Publish/Subscribe

Quando si parla di paradigma *Publish/Subscribe* si fa riferimento ad un pattern per lo scambio di messaggi in maniera asincrona. In questo paradigma il mittente dei messaggi (*Publisher*) non li invia direttamente ai destinatari (*Subscribers*) come avviene in altri paradigmi di comunicazione, bensì categorizza ogni messaggio in determinate classi e li rende disponibili pubblicandoli (*publish)* sulla rete di cui fa parte. I destinatari si sottoscrivono (*subscribe*) alle classi di messaggi di cui hanno interesse e, senza avere necessità di conoscere i *Publishers*, ricevono i messaggi a cui sono sottoscritti ogni volta che ne viene pubblicato uno.

Questo paradigma è discusso in letteratura da molti anni ed uno dei primi lavori che ne introduce l'utilizzo risale al 1987 ed in particolare al sottosistema "news" facente parte del sistema ISIS [21].

Questo capitolo ha come obiettivo l'introduzione del paradigma *Publish/-Subscribe*.

1.1 Paradigma Publish/Subscribe

Sin dall'inizio della diffusione dell'utilizzo di internet la scala dei sistemi distribuiti è cambiata ed è sempre in costante crescita, con sistemi composti potenzialmente da migliaia di entità eterogenee e con comportamenti variabili

[25, 32]. È necessario utilizzare dei modelli di comunicazione più flessibili e dinamici per gestire flussi di informazioni provenienti dalle più svariate sorgenti, dato che sistemi *point-to-point* e sincroni sono troppo statici e rigidi e non si adattano a reti dinamiche in costante evoluzione [25].

Il paradigma *Publish/Subscribe* è stato oggetto negli ultimi anni di una sempre maggiore attenzione come paradigma di comunicazione adatto a disseminare informazioni su sistemi distribuiti con area di utilizzo molto ampia [20], caratteristica importante considerando che al giorno d'oggi è fondamentale avere dei sistemi distribuiti in rete in grado di reagire rapidamente ad eventi rilevanti [29].

In un sistema con comunicazione *Publish/Subscribe* lo scambio di messaggi non avviene direttamente tra due componenti del sistema distribuito in rete, bensì dei nodi produttori di dati specifici pubblicano questi dati in rete e li rendono disponibili a nodi consumatori, nodi che si sono dichiarati interessati ai dati e che li ricevono non appena sono disponibili.

L'atto di connettere tra loro produttori e consumatori è detto *Brokering* [32] e per mettere in comunicazione i diversi *client* che compongono una rete con comunicazione di tipo *Publish/Subscribe*, *client* che possono essere sia produttori che consumatori di dati [29], è necessario definire un mezzo attraverso cui scambiare i messaggi: il componente del sistema che mette in comunicazione *Publisher* con *Subscribers* è detto *Broker*. Nella figura 1.1 è presente un esempio astratto di architettura *Publish/Subscribe*: in questo caso due *publisher* pubblicano messaggi attraverso un *broker*, messaggi che saranno poi distribuiti ai quattro *subscriber* che sono iscritti alla rete.

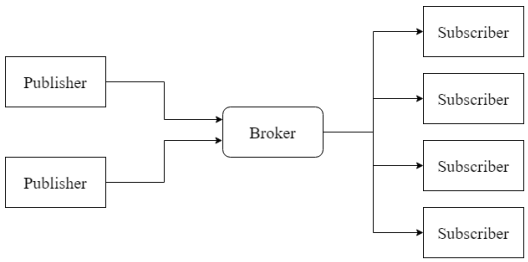

Figura 1.1: Paradigma Publish/Subscribe

1.2 Caratteristiche Publish/Subscribe

1.2.1 Space, Time and Flow Decoupling

Il paradigma *Publish/Subscribe* si adatta bene ad essere utilizzato da sistemi composti da molte entità che producono ed hanno bisogno di scambiarsi dati, in particolar modo quando la rete può essere molto dinamica e composta da entità eterogenee tra loro. Le principali caratteristiche dei sistemi *Publish/Subscribe* sono le seguenti [20, 25, 29, 32]:

- *Decoupling in Space*: i *client* che partecipano ad una comunicazione di tipo *Publish/Subscribe* non hanno nessuna informazione rispetto alla rete. Ogni *Publisher* non sa dunque quanti *Subscriber* sono presenti nella rete ne se effettivamente sono presenti; lo stesso vale per i *Subscribers*. Questa caratteristica aiuta nella flessibilità della rete, non essendo necessario memorizzare gli altri nodi *client*, nodi che possono arrivare ed andarsene dalla rete in qualsiasi momento senza nessun problema. In figura 1.2 è presente un esempio di *space decoupling* in cui un *Publisher* pubblica un messaggio, senza conoscere se e quanti *Subsribers* sono presenti, e tre *Subscribers* lo ricevono.

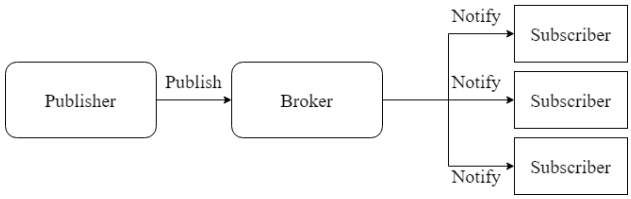

Figura 1.2: Space Decoupling in Publish/Subscribe

- *Decoupling in Time*: non è necessario che i *client* che partecipano alla comunicazione siano attivi contemporaneamente. Un *Subscriber* potrebbe ad esempio aggiungersi alla rete, sottoscriversi ad una tipologia di messaggi e ricevere messaggi pubblicati prima del suo arrivo da *Publisher* che non ne fanno più parte; allo stesso modo un *Publisher* potrebbe pubblicare messaggi a cui nessun *Subscriber* è interessato e

disconnettersi dalla rete senza nessun problema. In figura 1.3 è presente un esempio di *time decoupling*: in questo caso una freccia barrata è usata per rappresentare il fatto che un *client* non è connesso alla rete in un determinato istante di tempo, tempo che scorre dall'alto al basso; in questo scenario un *Publisher* pubblica un messaggio quando non ci sono dei *Subscriber* presenti e si disconnette, mentre un *Subscriber* si connette alla rete quando non ci sono *Publisher* e riceve il messaggio pubblicato in precedenza.

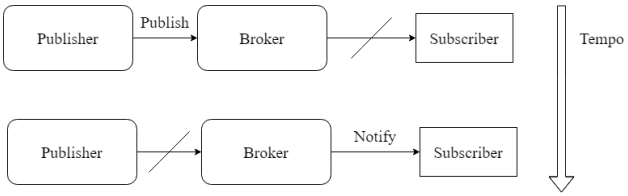

Figura 1.3: Time Decoupling in Publish/Subscribe

- *Decoupling in Flow*: la comunicazione avviene in maniera asincrona e non bloccante. Un *Publisher* non rimane bloccato in attesa che il suo messaggio venga ricevuto da un *Subscriber*, *Subscriber* che riceve ogni messaggio in maniera asincrona e anch'essa non bloccante. In questo modo la comunicazione è più fluida e si riducono i tempi in cui i *client* sono in attesa di qualche evento senza fare niente. In figura 1.4 è presente un esempio di *flow decomoposition*: in questo caso i due *client* durante il passare del tempo pubblicano o ricevono messaggi in maniera asincrona, senza necessità di attenderli e dunque non bloccandosi.

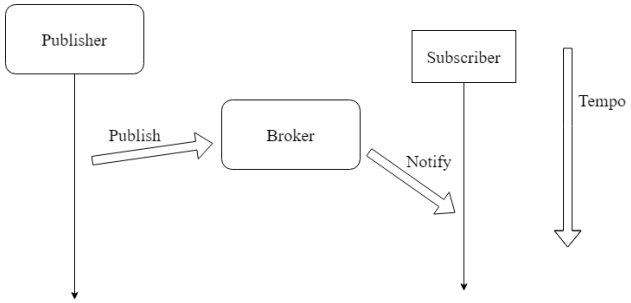

Figura 1.4: Flow Decoupling in Publish/Subscribe

1.2.2 Tipologia di messaggi Publish/Subscribe

Oltre alla definizione di *Publishers*, *Subscribers* e del mezzo per cui vengono scambiati i messaggi, un sistema *Publish/Subscribe* deve definire anche il tipo di messaggi che vengono scambiati. I sistemi *Publish/Subscribe* possono dunque essere divisi in due categorie principali in base al tipo di messaggi ed a come vengono effettuate le sottoscrizioni da parte dei *Subscribers* [20, 24, 25, 32]:

- *Topic-Based*: In un sistema *Topic-Based* i *Subscribers* si dichiarano interessati a dei *Topic* o gruppi di messaggi, gruppi che sono caratterizzati da parole chiave. I gruppi rappresentano distinti canali logici *many-to-many*, ognuno caratterizzato da metodi per classificare i contenuti. È possibile creare delle gerarchie di *Topic* ed un *Subscriber* riceve tutti i messaggi associati ad i *Topic* a cui è sottoscritto, con la possibilità di ricevere anche i messaggi che fanno parte della gerarchia a cui è interessato. In figura 1.5 è presente un esempio di *Topic-Based Publish/Subscribe*: i diversi *Publishers* pubblicano messaggi relativi ai *Topic* A,B,C ed i *Subscribers* ricevono i messaggi a cui sono sottoscritti; in questo caso il *Subscriber1* è iscritto ai *Topic* A e C mentre il *Subscriber2* solo al *Topic* B.

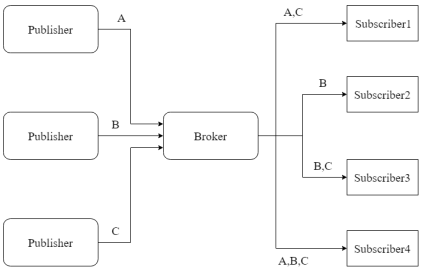

Figura 1.5: Topic-Based Publish/Subscribe

- *Content-Based*: In un sistema *Content-Based* i *Subscribers* dichiarano interesse e si sottoscrivono a determinate proprietà dinamiche dei messaggi pubblicati. Questo tipo di sottoscrizione aumenta la flessibilità del sistema, in quanto i *Subscribers* non sono più legati a specifici *Topic* ma solo a determinate caratteristiche del messaggio, rendendo possibile modellare ogni combinazione di informazioni come un singolo canale dinamico. Per decidere a chi inoltrare un messaggio si fa ricorso a *query* definite dai *Subscribers* quando manifestano interesse ad un determinato tipo di messaggi. In figura 1.6 è presente un esempio di *Content-Based Publish/Subscribe*: in questo caso il *Publisher* pubblica dei messaggi e, supponendo che i messaggi abbiamo un campo di nome *Tmp* utilizzato per descrivere una temperatura, il *Subscriber1* mostra interesse verso i messaggi con una temperatura minore di 100 mentre il *Subscriber2* mostra interesse verso messaggi con una temperatura maggiore di 60.

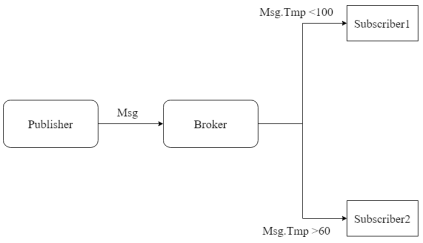

Figura 1.6: Content-Based Publish/Subscribe

1.3 Architetture Publish/Subscribe

Il paradigma *Publish/Subscribe* è un paradigma utilizzato per mettere in comunicazione diversi *client* che producono o consumano dati, *client* che possono essere molto eterogenei e numerosi. È possibile applicare il paradigma *Publish/Subscribe* a vari tipi di architettura di rete [32]:

- *Client-Server* centralizzata: un sistema basato su questa architettura fa affidamento su un unico server che agisce come *broker* per i diversi *client*. Questo tipo di architettura non scala bene all'aumentare dei *client* presenti e specialmente quando si tratta di utilizzare una comunicazione di tipo *Content-Based* [20]. In figura 1.7 è presente un esempio di architettura centralizzata, in cui i *Publisher* ed i *Subscriber* sono connessi tutti allo stesso server.

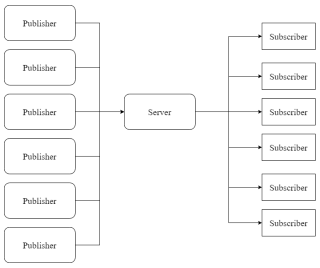

Figura 1.7: Architettura server centralizzata Publish/Subscribe

- *Client-Server* gerarchica: un sistema basato su questa architettura è composto da diversi server che hanno tra loro una relazione gerarchica e dai *client* che fanno parte della rete. Ogni server può essere connesso ad un certo numero di *client* e la comunicazione *server-server* e *server-client* segue lo stesso protocollo, facendo in modo che un *server* non distingua gli altri *server* dai propri *client*. L'obiettivo di questo tipo di architettura è la scalabilità, in quanto ogni *parent server* riceve messaggi solo dai suoi *client* e dai *server* che sono suoi figli, ma li inoltra solo verso la parte di sottoalbero che effettivamente richiede i dati. In figura 1.8 è presente un esempio di architettura gerarchia: Ogni *client*

è collegato ad un *server* mentre i server sono collegati tra loro in modo gerarchico. In questo caso il *Server(Root)* è la radice della gerarchia, il *Server1* è figlio del *Server(Root)* ed è padre del *Server2*.

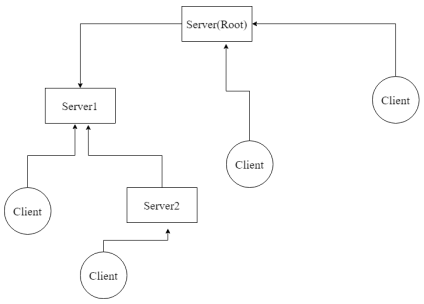

Figura 1.8: Architettura server gerarchica Publish/Subscribe

- *Client-Server* ad anello: in un sistema basato su questa architettura i *server* sono in relazione *peer-to-peer* l'uno con l'altro e le loro connessioni possono essere rappresentate graficamente come un anello. La comunicazione *server-server* è bidirezionale ed è diversa rispetto alla comunicazione *server-client*: mentre i *server* mantengono le informazioni gli uni degli altri, per una connessione *client-server* un *client* può generare sottoscrizioni o essere il destinatario di una richiesta mentre i *server* si occupano solo di inoltrare i messaggi assolvendo al loro ruolo di *broker*. In figura 1.9 è presente un esempio di architettura ad anello: in questo caso i *quattro* server sono collegati tra loro in maniera *peer-to-peer* mentre ogni *client* è collegato ad un solo server che agirà per lui come broker.

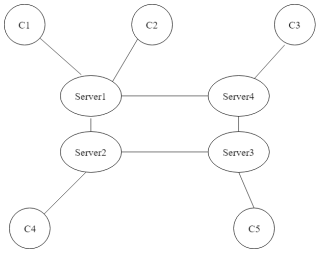

Figura 1.9: Architettura server ad anello Publish/Subscribe

- *Client-Server* irregolare: questo tipo di sistema è quasi del tutto equivalente ad un sistema ad anello, con la differenza che i *server* non sono collegati solo a due altri *server* in una topologia ad anello. In figura 1.10 è presente un esempio di architettura irregolare: in questo caso la struttura è simile a quelle della figura 1.9 con la differenza che è presente un collegamento tra *Server2* e *Server4* ed è presente un *server* aggiuntivo collegato con il *Server1*.

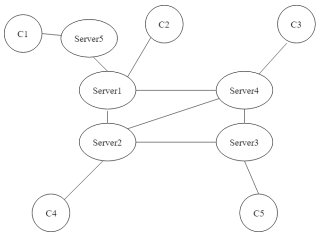

Figura 1.10: Architettura server irregolare Publish/Subscribe

- *Peer-to-Peer*: in un sistema con architettura *peer-to-peer* non sono presenti distinzioni tra *client* e *server*: ogni nodo può agire da *Publisher*, *Subscriber* e *broker*. In figura 1.11 è presente un esempio di architettura *peer-to-peer*.

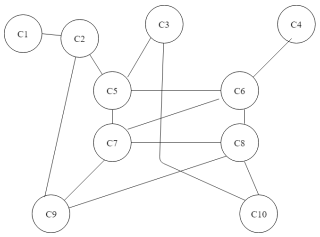

Figura 1.11: Architettura Peer-to-Peer Publish/Subscribe

Capitolo 2

Problema Affrontato

Con lo sviluppo e la diffusione dell'*Internet of Things*(IoT), il numero di applicazioni e servizi che ne fanno uso è sempre maggiore. L'IoT è un paradigma che connette tra loro utilizzando varie *networking technologies* diversi tipi di *smart devices*, *devices* che possono essere molto eterogenei tra loro.

Uno dei possibili utilizzi di questa tecnologia è il monitoraggio di ampie aree geografiche tramite l'utilizzo di reti sensori distribuiti nel territorio e collegati tra loro. Alcuni problemi di questo tipo di approccio sono la limitata capacità delle batterie dei sensori utilizzati che non rende efficiente mandare dati ed aprire connessioni con un grande numero di nodi che compongono la rete [37], la limitata capacità di banda della connessione all'interno della rete o le limitate capacità di memoria e del numero di connessioni che è possibile gestire all'interno della rete.

In questo capitolo si introdurrà il problema utilizzato come punto di partenza per lo sviluppo e la modellazione del sistema obiettivo di questo lavoro, descrivendone le caratteristiche, le strutture e le tecnologie utilizzate.

2.1 Descrizione del problema

Sono presenti diversi sensori distribuiti in un territorio che producono dati taggati in base al loro contenuto secondo una gerarchia di tipo *arg/subar-*

g/subsubarg/... (e.g. umidità/Bologna/stazione). I sensori sono collegati a *client* che si occupano di raccogliere i dati provenienti da diversi sensori e pubblicarli in rete. La rete è composta dai vari *client* che raccolgono dati dai sensori e da *server* intermedi che si occupano di mettere in collegamento i diversi *client*, utilizzando una comunicazione di tipo *Publish/Subscribe*.

Riprendendo la terminologia introdotta nel capitolo 1 i *client* si comportano sia come *Publisher* che *Subscriber* mentre i *server* si comportano da *broker*. I *client* che accedono alla rete si possono connettere a uno o molti dei *broker* presenti e attraverso di esso pubblicare e ricevere dati. I dati oltre che dal tag a loro associato sono caratterizzati dalla banda richiesta per la loro trasmissione in rete. Un *client* per richiedere un tipo di dato deve sottoscriversi allo specifico tag di interesse, sottoscrizione che ammette anche *wildcard* (e.g. *umidità/Bologna/#*).

Tutte le connessioni esistenti nella rete, sia tra *client* e *broker che* tra *broker* e *broker*, hanno una banda passante limitata e sono basate sulle tecnologie LoRa e LoRaWAN, descritte nella sezione 2.2, mentre il protocollo di livello applicazione utilizzato dai componenti della rete per comunicare tra loro è il protocollo MQTT, descritto nella sezione 2.3. Esiste anche la possibilità che alcuni nodi che compongono la rete siano connessi ad internet, caso che comporta la presenza di una banda in uscita sostanzialmente non limitata.

I *broker* sono limitati dal numero di connessioni che possono accettare ed aprire verso altri *broker* o *client* e, quando il numero di richieste in arrivo ad un *broker* raggiunge il limite, un *broker* non può accettare ulteriori connessioni e le rifiuta. I *client* non sono vincolati a connettersi ad un solo *server* ma possono scegliere di connettersi ad uno qualunque dei *server* visibili, cambiando *server* di riferimento in maniera dinamica in base al carico di connessioni a cui è sottoposto un *broker*. In figura 2.1 è presente un esempio di rete: in questo caso il *Client1* vede il *Broker1* ed il *Broker2*, il *Client2* vede il *Broker1* ed il *Broker4* ed il *Client3* vede il *Broker4* ed il *Broker7*; qualora le richieste ad uno dei *broker* si saturassero, i *Client* potrebbero de-

cidere di aprire una connessione verso un altro dei *Broker* da loro visibili. La rete può inoltre cambiare in maniera dinamica, con la possibilità che un nodo sparisca o che ne appaiano di nuovi, così come le connessioni tra i nodi possono apparire o scomparire.

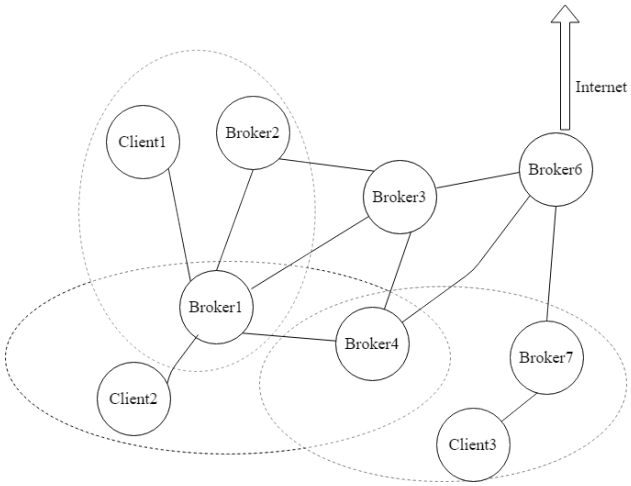

Figura 2.1: Esempio Rete Client/Broker

I *broker* non memorizzano dati ma solo i tag richiesti dai *client* o da altri *broker* a loro connessi e si occupano di inoltrare dati compatibili con le richieste memorizzate. Inoltre i *broker* possono essi stessi richiedere dati ad altri nodi, in particolare i nodi connessi ad internet possono richiedere agli altri nodi i dati che devono essere inoltrati in rete. In questo modo emergeranno cammini dinamici tra produttori di dati e nodi connessi con l'esterno.

Dunque il problema può essere descritto come un problema di network design, il cui obiettivo è trovare la configurazione di rete in grado di massimizzare il numero di connessioni accettate dai *broker* e quindi la banda allocata dai *client*, minimizzando il numero di richieste che non vengono soddisfatte dalla rete.

2.2 LoRa e LoraWan

2.2.1 Origini della tecnologia LoRa

LoRa, il cui nome è un riferimento al lungo raggio delle connessioni che permette (*long-range*), è una tecnologia per la modulazione di frequenze radio (*RF modulation*) per *low-power, wide area networks(LPWANs)* [11]. LoRa è stato sviluppato nel 2010 dalla società francese *Cycleo* utilizzando la tecnologia *Chirp Spread Spectrum (CSS)*, tecnologia largamente utilizzata sia nell'industria marittima per i sonar che nell'aviazione per i radar [12]. Convinta dalle potenzialità della nuova tecnologia, nel 2012 Semtech ha acquisito Cycleo e nel 2015 ha fondato la LoRa Alliance, un'organizzazione no-profit con lo scopo di promuovere l'utilizzo dello standard LoRaWAN nelle *low-power, wide area networks(LPWANs)* [10].

2.2.2 Caratteristiche LoRa

LoRa si basa su una tecnica proprietaria derivata dalla tecnologia *Chirp Spread Spectrum (CSS)* e offre la possibilità di effettuare comunicazioni a lungo raggio tra i dispositivi che la utilizzano, con distanze fino a 5 chilometri in aree urbane e 15 chilometri in aree più rurali. Una delle caratteristiche principali di LoRa è la bassissima richiesta di energia per permettere ai dispositivi di comunicare, fattore che permette di creare dispositivi che possono operare anche per 10 anni senza avere necessità di sostituire la batteria.

La tecnologia LoRa, come mostrato in figura 2.2, si colloca nel livello fisico del modello OSI ed utilizza l'aria invece che dei cavi come mezzo di trasporto delle onde radio tra i dispositivi [11].

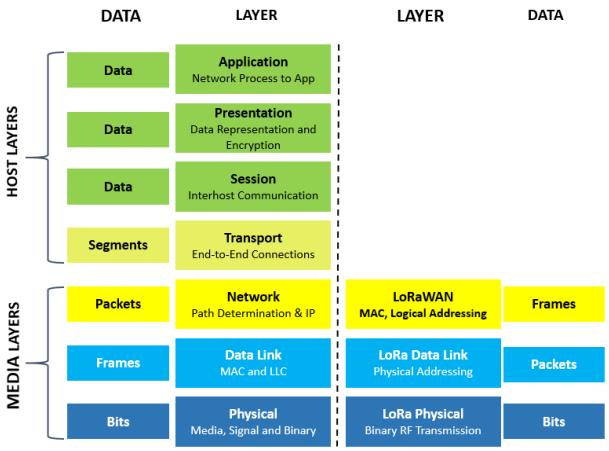

Figura 2.2: Stack OSI LoRa

2.2.3 LoRaWAN

LoRaWAN, come detto nella sezione 2.2.1, è un protocollo standard aperto per la comunicazione di tipo *LPWAN* gestito dalla LoRa Alliance. Il protocollo LoRaWAN si adatta ad essere utilizzato da applicazioni che richiedono comunicazioni a lunga distanza tra un grande numero di dispositivi con capacità energetiche limitate e che utilizzano piccole quantità di dati [11].

LoRaWAN definisce il protocollo di comunicazione e l'architettura di sistema per reti che utilizzano la tecnologia di comunicazione LoRa.

In una rete LoRaWAN i nodi non sono associati ad uno specifico *gateway* bensì i dati trasmessi da ogni nodo sono tipicamente ricevuti da multipli *gateway*, che inoltreranno i dati ricevuti attraverso la rete.

I nodi che fanno parte di una rete LoRaWAN comunicano tra loro in maniera asincrona, inviando dati solo quando sono disponibili. Eliminare la sincronizzazione nella comunicazione permette di ridurre il consumo energetico della rete, dato che non è necessario che i nodi controllino periodicamente la presenza di nuovi messaggi per sincronizzarsi.

Le reti LoRaWAN hanno una grande capacità di comunicazione ottenuta tramite l'utilizzo di velocità di trasferimento dati adattive ed utilizzando ricetrasmittori multicanale nei *gateway*, in modo da poter ricevere messaggi

in maniera simultanea su diversi canali. [19]

2.3 MQTT

Message Queuing Telemetry Transport (MQTT) è un protocollo standard per lo scambio di messaggi per l'*Internet of Things*, progettato per essere estremante leggero utilizzando il Paradigma *Publish/Subscribe* ed ideale per connettere dispositivi remoti utilizzando una bassa larghezza di banda [13]. Viene introdotto nel 1999 da IBM e nel 2013 diventa uno standard *Organization for the Advancement of Structured Information Standards* (OASIS).

Il protocollo MQTT si basa sul modello *Client-Server* ed utilizza TCP/IP o altri protocolli di rete che forniscono connessioni ordinate, senza perdite e bidirezionali, con un basso *overhead* per il trasporto dei dati e scambio di dati minimizzati per ridurre il traffico sulla rete [14].

Come ogni protocollo basato sul paradigma *Publish/Subscribe*, MQTT definisce ed utilizza tre componenti principali nelle reti che lo utilizzano: *Publishers*, *Subscribers* e *Brokers*. Facendo riferimento al modello *Client-Server*, MQTT definisce gli *MQTT Client* come qualsiasi oggetto *IoT* che genera dati (*Publisher*) o che li riceve (*Subscriber*); in MQTT qualsiasi dispositivo o programma può agire come *Client* [14, 39]. Un *MQTT Server* è un qualsiasi dispositivo o programma che agisce come intermediario tra i *Client* che pubblicano messaggi ed i *Client* che sono sottoscritti ad un determinato tipo di messaggio, agendo dunque da *Broker* [14].

I messaggi che vengono scambiati in MQTT contengono un *payload*, che è il contenuto informativo del messaggio, il tipo di modalità secondo cui il messaggio sarà consegnato[1], un insieme di proprietà ed il nome del *topic* a cui fa riferimento [33]. Il nome del *topic* identifica il canale su cui il messaggio è pubblicato e viene utilizzato dai *Client* per pubblicare e ricevere messaggi [14].

[1] la modalità fa riferimento alla *Quality of Service* spiegata più avanti in questa sezione

I compiti principali che sono assegnati ad un *MQTT Client* sono i seguenti [14]:

- Aprire la connessione verso il *Server*
- Pubblicare messaggi a cui altri *Client* potrebbero essere interessati, quando agisce come *Publisher*.
- Effettuare sottoscrizioni ai *topic* di interesse, quando agisce come *Subscriber*
- Cancellare le sottoscrizioni ai *topic* che non sono più di proprio interesse, quando agisce come *Subscriber*
- Chiudere la connessione verso il *Server*

I compiti principali che sono assegnati ad un *MQTT Server* sono i Seguenti [14]:

- Accettare le richieste di connessione provenienti dai *Client*
- Accettare i messaggi pubblicati dai *Client/Publishers*
- Processare le richieste di sottoscrizione e di cancellazione della sottoscrizione riceveute dai *Client/Subscribers*
- Inoltrare i messaggi di interesse ai *Client* che sono sottoscritti al *topic* relativo ai messaggi.
- Chiudere le connessioni dei *Client*

In figura 2.3 è presente una rappresentazione di questa struttura: in questo caso abbiamo un *MQTT Client* che pubblica dati riguardo il *topic* temperatura verso un *MQTT Broker*; allo stesso modo abbiamo due *MQTT Client* che sono sottoscritti al *topic* temperatura e che dunque ricevono i dati pubblicati dal primo *Client* attraverso il *Broker* a cui sono connessi.

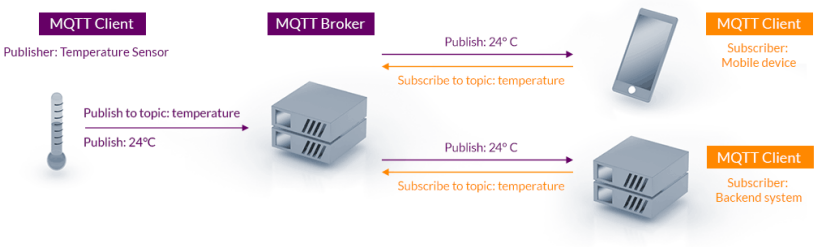

Figura 2.3: MQTT publish/subscribe

In MQTT è possibile fare in modo che un *Broker* memorizzi alcuni messaggi per poi inoltrarli a *Client* che in futuro si sottoscriveranno al *topic* di riferimento, opzione utile quando i *Publishers* pubblicano messaggi in maniera irregolare [14].

Per consegnare i messaggi MQTT offre la possibilità di utilizzare tre modalità o livelli diversi, detti *Quality of Service (QoS)* [14, 33, 36, 39]:

- *Quality of Service 0*: il livello zero è definito come *at most once*, prevede dunque che il messaggio sia consegnato una sola volta, senza garanzie che il messaggio sia stato effettivamente ricevuto.
- *Quality of Service 1*: il livello uno è definito come *at least once*, prevede dunque che i messaggio sia ricevuto almeno una volta ed offre la possibilità che i messaggi siano duplicati settando un opportuno *flag* nel messaggio inviato.
- *Quality of Service 2*: il livello due è definito come *exactly once*, prevede dunque che il messaggio sia consegnato esattamente una volta. Per applicare questo tipo di garanzie è necessario definire un protocollo aggiuntivo di controllo, per fare in modo che la conferma dell'effettiva ricezione del messaggio da parte del destinatario sia trasmessa e confermata al mittente.

Capitolo 3

Formulazione Matematica

Come descritto nel capitolo 2 e più in particolare nella sezione 2.1, il problema da risolvere si può ridurre ad un problema di network design il cui obiettivo è massimizzare il numero di connessioni accettate dai *broker* e dunque la quantità di informazioni che vengono trasmesse nella rete tra i vari *Client*. Questo problema può essere modellato come un problema di Programmazione Lineare Intera e più in particolare come un problema di Flusso.

In questo capitolo si procederà ad introdurre la Programmazione Lineare Intera, i problemi di Flusso e si presenteranno delle formulazioni matematiche adatte a risolvere il problema analizzato.

3.1 Programmazione Lineare

La programmazione lineare è un caso speciale della programmazione matematica, programmazione matematica che è definita come la branca della matematica che utilizza e sviluppa tecniche per massimizzare o minimizzare una funzione obiettivo soggetta a vincoli lineari, non lineari e interi sulle variabili [23]. Un problema di programmazione lineare può essere definito come il problema di massimizzare o minimizzare una funzione lineare soggetta a vincoli lineari, vincoli che possono essere uguaglianze o disuguaglianze [23, 26]. In seguito è presente un esempio di problema di programmazione lineare:

maximize

$$z = x_1 + x_2 \tag{3.1}$$

subject to

$$x_1 + 2x_2 \leq 4 \tag{3.2}$$

$$4x_1 + 2x_2 \leq 12 \tag{3.3}$$

$$-x_1 + x_2 \geq 1 \tag{3.4}$$

$$x_1, x_2 \geq 0 \tag{3.5}$$

- Le variabili x_1, x_2 sono dette variabili decisionali e rappresentano i valori finali che si vogliono trovare ottimizzando il problema.
- L'equazione 3.1 è chiamata funzione obiettivo, che rappresenta l'obiettivo dell'ottimizzazione che si sta eseguendo. In questo caso si sta cercando la coppia di numeri x_1, x_2 la cui somma sia massima e che rispetti i vincoli successivi.
- Le equazioni 3.2, 3.3 e 3.4 sono definiti vincoli principali. I vincoli presenti in questo problema sono disuguaglianze in diverse funzioni lineari delle variabili.
- Infine l'equazione 3.5 rappresenta i vincoli di non-negatività, vincoli che sono spesso presenti in problemi di programmazione lineare e impongono che le variabili decisionali assumano solo valori positivi.

3.1.1 Risoluzione di problemi di programmazione lineare

Per poter risolvere problemi di programmazione lineare è possibile procedere in vari modi. Quando il problema presenta pochi vincoli e variabili decisionali, è possibile risolvere il problema in maniera grafica, come mostrato in

figura 3.1. In questo caso si mostra la soluzione al problema di massimizzazione rappresentato dalla funzione obiettivo 3.1 presentata nella sezione 3.1.

Tuttavia non è sempre possibile risolvere questo tipo di problemi utilizzando una soluzione grafica, data la possibilità che il problema sia composto da un gran numero di variabili decisionali e vincoli. Uno dei metodi utilizzati per risolvere problemi di programmazione lineare è l'algoritmo del Simplesso, che è una procedura molto efficiente per risolvere grandi problemi di programmazione lineare [23]. L'algoritmo del simplesso utilizza una soluzione iniziale per il problema e trova una soluzione ottima se presente. Qualora la soluzione non sia presente l'algoritmo termina indicando che non esiste una soluzione [23].

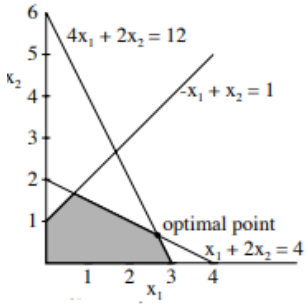

Figura 3.1: Soluzione grafica di un problema di programmazione lineare

Rilassamento Lagrangiano

In presenza di un problema complicato, caratterizzato dalla presenza di vincoli difficili e che rendono il problema arduo da risolvere, è possibile semplificare la sua risoluzione applicando un rilassamento lagrangiano. Il rilassamento lagrangiano permette di risolvere un sottoproblema del problema originale, ottenuto rilassando i vincoli difficili ed inserendoli nella funzione obiettivo con un moltiplicatore, moltiplicatore che rappresenta una penalità aggiunta alla soluzione che non soddisfa il vincolo [31].

In seguito è presente un esempio di applicazione di rilassamento lagrangiano: è data la funzione obiettivo 3.6 soggetta al vincolo "semplice" 3.7 ed al vincolo "difficile" 3.8 che rende il problema complicato da ottimizzare.

minimize

$$z = \sum_j c_j x_j \tag{3.6}$$

subject to

$$\sum_j x_j \leq a_j \tag{3.7}$$

$$\sum_j x_j \leq b_j \tag{3.8}$$

$$x_j \geq 0 \tag{3.9}$$

Applicando un rilassamento lagrangiano si ottiene un nuovo problema più semplice da risolvere, con il vincolo "difficile" 3.8 rimosso e che è stato aggiunto alla funzione obiettivo 3.10 moltiplicato per un fattore λ_j.

minimize

$$z = \sum_j c_j x_j + \sum_j \lambda_j x_j - \sum_j \lambda_j b_j \tag{3.10}$$

subject to

$$\sum_j x_j \leq a_j \tag{3.11}$$

$$x_j \geq 0 \tag{3.12}$$

Ottimizzando il secondo problema la soluzione sarà trovata in maniera più rapida e semplice, con il risultato che sarà un'approssimazione del risultato del primo problema. [31].

3.1.2 Programmazione Lineare Intera

Quando in un problema di programmazione lineare sono presenti degli ulteriori vincoli che impongono ad una o più variabili decisionali di assumere solo

valori interi, il problema viene definito come un problema di programmazione lineare intera. Se solo alcune delle variabili devono rispettare questi vincoli il problema è definito come un problema di programmazione lineare mista intera.

Dato che questo tipo di problema presenta ulteriori vincoli, la sua soluzione è più complessa rispetto alla soluzione di problemi di programmazione lineare. Una delle tecniche più utilizzate per risolvere problemi di programmazione lineare mista intera è l'applicazione di metodi *branch-and-cut*, che sono algoritmi esatti che consistono nella combinazione del metodo dei piani di taglio con un algoritmo *branch-and-bound*, risolvendo una sequenza di rilassamenti lineari del problema di partenza [23, 34].

A partire da un problema di programmazione lineare intera è possibile generare un problema di programmazione lineare utilizzando la stessa funzione obiettivo e gli stessi vincoli ma sostituendo il vincoli di interezza delle variabili con opportuni vincoli continui (e.g. il vincolo $x_i = [0, 1]$ può essere sostituito con due vincoli continui $x_i \leq 1$ e $x_i \geq 0$) [17]. Il problema così ottenuto è chiamato rilassamento lineare del problema originale.

L'applicazione del metodo dei piani di taglio per problemi di programmazione intera è stata proposta per la prima volta da Gomory nel 1963 [28]. Questi metodi hanno l'obiettivo di migliorare i rilassamenti lineari per approssimare il problema in maniera via via più precisa.

Gli algoritmi *branch-and-bound* sono una tipologia di algoritmi utilizzati per risolvere problemi di programmazione lineare intera che si basano su una strategia *divide et impera*. Un algoritmo *branch-and-bound* ricerca la migliore soluzione di un problema effettuando una analisi completa del suo spazio di soluzioni [22].

Gli algoritmi di tipo *branch-and-bound* sono caratterizzati dalla scomposizione del problema in sottoproblemi più semplici, che sono risolti e la cui soluzione parziale è mantenuta in una struttura ad albero. I nodi inesplorati dell'albero generano dei nodi figli partizionando lo spazio delle soluzioni in regioni più piccole che sono risolte in maniera ricorsiva (i.e. *branching*), men-

tre sono applicate delle regole per tagliare delle regioni dello spazio di ricerca che sono considerate sub-ottimali (i.e. *bounding*). Una volta che l'intero albero è stato esplorato viene tornata la soluzione migliore trovata durante il processo [35].

3.2 Problemi di Flusso

La categoria dei problemi di flusso fa riferimento a tutti quei problemi che si occupano di ottimizzare il flusso in una particolare struttura definita rete di flusso. Un grafo $G = (V, A)$, dove l'insieme V rappresenta l'insieme dei suoi vertici e l'insieme A rappresenta l'insieme degli archi tra i vertici, è definito rete di flusso (*flow network*) se ha una capacità positiva associata ad ogni arco e le due tipologie seguenti di vertici aggiuntivi [27]:

- Vertici sorgenti: i vertici sorgenti s sono il punto di origine dei flussi all'interno della rete.
- Vertici pozzo: i vertici pozzo t sono il punto di destinazione dei flussi all'interno della rete.

In figura 3.2 è presente un esempio di rete di flusso: il nodo S rappresenta la sorgente della rete, il nodo T la destinazione, i nodi intermedi N rappresentano i vertici della rete ed i numeri di fianco agli archi rappresentano la loro capacità.

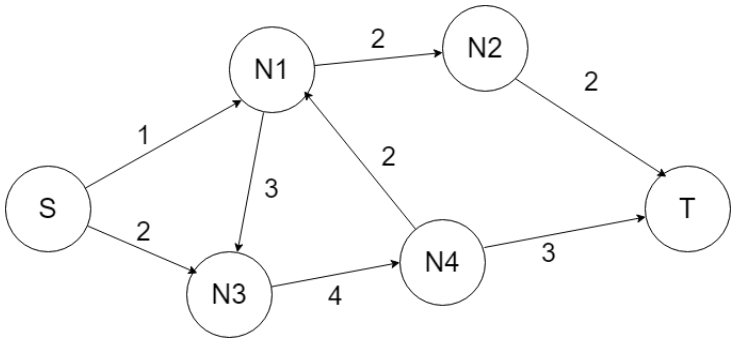

Figura 3.2: Grafo di una rete di flusso

Le reti di flusso possono essere utilizzate per modellare tramite programmazione lineare varie tipologie di problemi, come ad esempio problemi di flusso massimo (*MaxFlow*) in cui si vuole massimizzare il flusso entrante nelle destinazioni ed i problemi di costo minimo del flusso (*minimum-cost-flow*), in cui si vuole trovare il modo meno costoso per inviare il flusso dalle sorgenti alle destinazioni.

Un problema di programmazione lineare riferito ad una rete di flusso presenta generalmente i seguenti vincoli [27]:

$$f(v,w) \leq c(v,w) \qquad forall(v,w) \in VXV \tag{3.13}$$

$$f(v,w) \leq -f(w,v) \qquad forall(v,w) \in VXV \tag{3.14}$$

$$\sum_{u \in V} f(u,v) = 0 \qquad forallv \in V - \{s,t\} \tag{3.15}$$

Il vincolo 3.13 rappresenta il vincolo di capacità, che ci dice che il flusso passante per ogni arco (v,w) deve essere minore della capacità c di quell'arco. Il vincolo 3.14 modella il concetto che il flusso entrante in un arco deve essere uguale ma di segno opposto a quello uscente mentre il vincolo 3.15 indica che la somma totale dei flussi in ogni vertice $v \notin \{s,t\}$ deve essere 0. Come si può notare dalla presenza del vincolo 3.15 una rete di flusso modella un sistema in cui il flusso che parte dalle sorgenti ed arriva alle destinazioni deve essere conservato e non si può fermare nei vertici intermedi che compongono la rete.

I problemi di flusso possono essere applicati alla modellazione di diversi ed importanti problemi reali, come ad esempio:

- Reti idrauliche: un esempio di rete idraulica a cui applicare un problema di flusso è rappresentato dalla rete idraulica di una città: i nodi sorgente sono rappresentati dalle origini del flusso d'acqua (laghi, fiumi, serbatoi etc.), le destinazioni sono rappresentate ad esempio dalle abitazioni della città, i nodi intermedi sono rappresentati dalle stazioni di pompaggio interne alla rete e gli archi sono rappresentati dalle tubazioni che collegano i vari nodi.

- Reti di comunicazione: un esempio di rete di comunicazione è rappresentato dalle reti telefoniche: i nodi sorgente e destinazione sono i telefoni che effettuano e ricevono la chiamata, i nodi intermedi sono rappresentati dai nodi di rete o dai satelliti che amplificano e dirigono il segnale entrante e gli archi sono rappresentati dai cavi e dai vari *relays* che connettono i nodi.

- Reti di trasporto: un esempio di rete di trasporto è rappresentato dalla rete stradale: i nodi sorgente e destinazione sono rappresentati dai punti di interesse all'interno della rete(abitazioni, supermercati, luoghi di lavoro etc.), i nodi intermedi sono rappresentati dai vari incroci all'interno della rete e gli archi sono rappresentati dalle strade che collegano i vari nodi.

MultiCommodity Flow Una variazione ai classici problemi di flusso è rappresentata dai problemi in cui in una rete passano diversi tipi di flusso contemporaneamente (e.g. in una rete di trasporto possono passare veicoli diversi, in una rete idraulica diversi tipi di fluidi etc.). Un problema di questo tipo è definito *multicommodity flow problem* ed è caratterizzato dalla presenza di diversi tipi di flusso, detti *commodities*, dalle varie sorgenti verso le rispettive destinazioni, rispettando gli stessi vincoli di capacità definiti per i problemi di flusso con la differenza che ad ogni *commodity* è associato un peso che indica quanta capacità consuma una determinata *commodity* passando per un arco e ad ogni destinazione è associata anche una quantità richiesta di una *commodity* [30].

In figura 3.3 è presente un esempio di questo tipo di problema: in questo caso il nodo S1c1 è una sorgente e genera una *commodity* c1, il nodo S2c2 è una sorgente che genera una *commodity* c2 ed il nodo Tc1c2 è una destinazione che richiede entrambe le *commodities*.

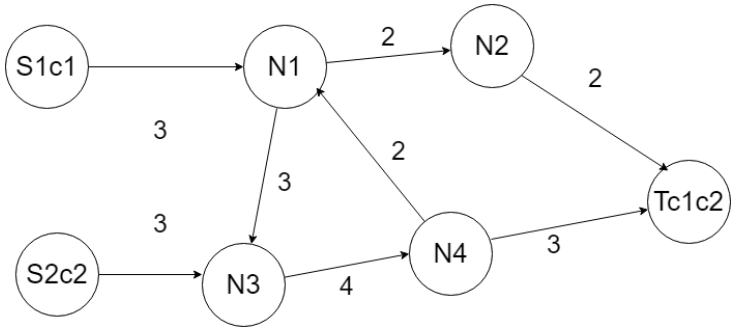

Figura 3.3: Grafo di una rete di flusso con multiple *commodities*

3.3 Formulazione matematica del problema

Dopo aver introdotto i concetti di programmazione lineare intera e di problema di flusso, è ora possibile proporre una formulazione matematica adatta a risolvere il problema presentato nella sezione 2.1 ed oggetto di questo lavoro.

3.3.1 Struttura iniziale del problema

Il problema da ottimizzare sarà un problema di tipo *Multicommodity flow* il cui grafo può essere descritto in questo modo:

- S è l'insieme dei nodi sorgente, che nel nostro problema rappresenta l'insieme dei *Client* che pubblicano dati relativi ad un *topic*.
- T è l'insieme dei nodi destinazione, che nel nostro problema rappresenta l'insieme dei *Client* che si sottoscrivono ad un determinato *topic*.
- B è l'insieme dei nodi intermedi, che nel nostro problema rappresenta l'insieme dei *broker* presenti nella rete.
- A è l'insieme degli archi presenti nel grafo, che nel nostro problema rappresentano le connessioni *Client-Broker* e *Broker-Broker*
- Φ è l'insieme delle *commodity* presenti nella rete, *commodities* che nel nostro problema modellano i *topic* presenti nel sistema.

Utilizzando il grafo ottenuto con gli elementi descritti sarebbe possibile impostare un problema di flusso ottimizzabile tramite programmazione lineare; va notato tuttavia che il problema di interesse presenta una criticità fondamentale che non viene ben modellata da una rete costruita utilizzando gli elementi descritti: le *commodities* generate dalle sorgenti potrebbero essere richieste da più destinazioni, fatto che rende necessario che i dati vengano generati una sola volta ma potenzialmente "duplicati" ed inviati verso più destinazioni da parte dei *broker*.

3.3.2 Modifica proposta alla struttura del problema

Per ovviare al problema esposto nella sezione precedente si propone la seguente modifica alla struttura del problema: Per ogni destinazione che richiede una determinata *Commodity* viene creata una nuova *Commodity* associata alla destinazione e generata dalla sorgente d'origine, sostituendo così l'insieme delle *Commodities* con un nuovo insieme composto da tutte le nuove *Commodities* ottenute. Le *Commodities* così ottenute possono essere descritte come sottoscrizioni da parte dei nodi destinazione ai vari *topic*. Viene poi inserito un nuovo insieme di destinazioni "*Dummy*" al problema, ottenuto aggiungendo al grafo una nuova destinazione associata alle vecchie destinazioni per ogni *commodity* richiesta dalla destinazione originale, mentre le vecchie destinazioni vengono collegate tramite archi con capacità infinita alle nuove destinazioni *Dummy* inserite. Per poter ottimizzare e modellare il problema viene infine aggiunto un nuovo arco che collega le sorgenti con le nuove destinazioni *Dummy*, arco che sarà parte fondamentale della formulazione proposta in quanto gli sarà associata una penalità nella funzione obiettivo. In figura 3.4 è presente un esempio della trasformazione del grafo effettuata eseguendo le operazioni appena descritte: partendo dal grafo presente in figura 3.3, le *Commodities* c1 e c2 vengono trasformate in DTc1 e DTc2, la vecchia destinazione Tc1c2 viene trasformata nel nodo T, vengono aggiunti due nodi *Dummy* DTS1c1 e DTS2c2 dato che la vecchia destinazione richie-

deva due *Commodities* ed infine vengono aggiunti due archi che collegano le sorgenti con i nuovi nodi *Dummy*.

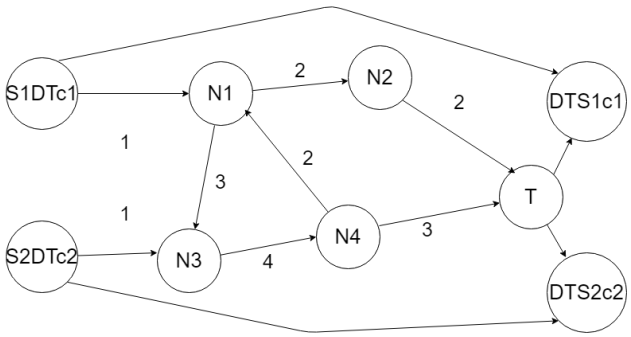

Figura 3.4: Aggiunta dei nodi destinazione "*Dummy*" alla rete

La nuova struttura del problema così ottenuto può essere dunque descritto allo stesso modo della struttura descritta nella 3.3.1 con l'aggiunta dei seguenti concetti:

- D è l'insieme dei nuovi nodi *Dummy* aggiunti al problema.
- K è l'insieme delle nuove *Commodities* generate dopo la modifica del problema.
- K_φ è l'insieme delle nuove *Commodities* che si riferiscono alla stessa *Commodity* $\phi \in \Phi$

3.3.3 Formulazione proposta

Utilizzando i concetti introdotti fin'ora viene in seguito esposta la formulazione matematica proposta per il problema analizzato: Definiamo x_{ij}^k come variabile binaria che assume valore 1 se la *commodity* $k \in K$ viene trasmessa dall'arco $\{i, j\}$. Definiamo $\xi_{i,j}^{\varphi}$ la variabile binaria che assume valore 1 se la *Commodity* $\varphi \in \Phi$ viene trasmessa dall'arco $\{i, j\}$. Definiamo a^{φ} la variabile che indica il peso di ogni *Commodity* $\varphi \in \Phi$. Definiamo $c_{i,j}$ la variabile che indica la capacità di ogni arco $\{i, j\}$. Definiamo b_s^k e b_t^k le variabili che assumono valore 1 se i nodi $s \in S, t \in D$ generano o accettano la *Commodity*

$k \in K$ e 0 altrimenti. Definiamo n_{max} la variabile che indica quante volte una commodity può essere duplicata all'interno della rete, e serve per fare in modo che le *commodity* generate $k \in K$ riferite ad una stessa *commodity* originaria $\phi \in \Phi$ siano trattate come un'unica *commodity*. Infine definiamo $p_{i,j}$ la variabile che indica la penalità che viene pagata se viene utilizzato l'arco $\{i, j\}$.

La formulazione matematica è la seguente:

minimize

$$z = \sum_{k \in K} \sum_{i \in N} \sum_{j \in \Gamma_i^{-1}} p_{ji} x_{ji}^k \tag{3.16}$$

subject to

$$\sum_{j \in \Gamma_s} x_{sj}^k = b_s^k \qquad s \in S, \forall k \tag{3.17}$$

$$\sum_{j \in \Gamma_t^{-1}} x_{jt}^k = b_t^k \qquad t \in D, \forall k \tag{3.18}$$

$$\sum_{j \in \Gamma_i^{-1}} x_{ji}^k = \sum_{j \in \Gamma_i} x_{ij}^k \qquad i \in B \cup T, \forall k \tag{3.19}$$

$$\sum_{k \in K_\varphi} \sum_{j \in \Gamma_i} x_{ij}^k \leq n_{max} \xi_{ij}^\varphi \qquad \phi \in \Phi, i \in B \cup S \cup T \tag{3.20}$$

$$\sum_{\varphi \in \Phi} a^\varphi \xi_{ij}^\varphi \leq c_{ij} \qquad (i, j) \in A \tag{3.21}$$

$$x_{ij}^k, \xi_{ij}^\varphi \in \{0, 1\} \tag{3.22}$$

- L'equazione 3.16 rappresenta la funzione obiettivo del nostro problema. Per ottimizzare il nostro problema la variabile $p_{i,j}$ assumerà un valore positivo ≥ 0 per ogni arco $i \in S$, $j \in D$ e 0 altrimenti. In questo modo l'obiettivo sarà quello di minimizzare il costo del flusso interno alla rete penalizzando il flusso che passa direttamente tra nodi sorgenti e destinazioni *Dummy*.

- L'equazione 3.17 rappresenta la condizione che ogni sorgente della *commodity* k debba generare la *commodity*. Quando una sorgente non genera una determinata *commodity* il flusso uscente viene invece settato a 0, grazie alla variabile b_s^k

- L'equazione 3.18 rappresenta la condizione che ogni destinazione *Dummy* della *commodity* k debba ricevere la *commodity*. Quando una destinazione non richiede una determinata *commodity* il flusso entrante viene invece settato a 0, grazie alla variabile b_t^k

- L'equazione 3.19 rappresenta il vincolo che ogni *Commodity* che entra in un nodo i debba anche uscire, e modella il vincolo di continuità e conservazione del flusso all'interno della rete.

- L'equazione 3.20 rappresenta il vincolo che permette ad ogni *commodity* generata "k" riferita ad una *commodity* "φ" di essere trattata come una sola commodity, mettendo in relazione le variabili $x_{i,j}$ e $\xi_{i,j}$.

- L'equazione 3.21 rappresenta il vincolo di capacità degli archi. Da notare che questo vincolo considera il peso delle *commodity* φ originali.

- L'equazione 3.22 rappresenta i vincoli di interezza delle variabili decisionali ed in particolare ci dice che possono assumere solo valori 0 o 1.

3.3.4 Formulazione Lagrangiana

Come esposto nella sezione 3.1.1 è possibile definire un problema approssimato rilassando i vincoli considerati difficili ed inserendoli nella funzione obiettivo. Per risolvere il problema analizzato viene dunque proposta una formulazione ottenuta rilassando il vincolo 3.21 ed inserendolo nella funzione obiettivo. Questo rilassamento viene proposto oltre che per semplificare il problema rimuovendo un vincolo complesso, per rimuovere ogni vincolo locale ai nodi che compongono la rete, rendendo possibile la distribuzione

del calcolo in un'ottica di implementazione del sistema. Definendo λ_{ij} come la variabile di penalità da associare al vincolo rimosso dalla formulazione originale, la nuova formulazione proposta è la seguente:

minimize

$$z = \sum_{k \in K} \sum_{i \in N} \sum_{j \in \Gamma_i^{-1}} p_{ji} x_{ji}^k + \sum_{\phi \in \Phi} a^{\varphi} \sum_{ij \in A} \lambda_{ij} \xi_{ij}^{\varphi} - \sum_{ij} \lambda_{ij} c_{ij} \tag{3.23}$$

subject to

$$\sum_{j \in \Gamma_s} x_{sj}^k = b_s^k \qquad s \in S, \forall k \tag{3.24}$$

$$\sum_{j \in \Gamma_t^{-1}} x_{jt}^k = b_t^k \qquad t \in D, \forall k \tag{3.25}$$

$$\sum_{j \in \Gamma_i^{-1}} x_{ji}^k = \sum_{j \in \Gamma_i} x_{ij}^k \qquad i \in B \cup T, \forall k \tag{3.26}$$

$$\sum_{k \in K_{\varphi}} \sum_{j \in \Gamma_i} x_{ij}^k \leq n_{max} \xi_{ij}^{\varphi} \qquad \phi \in \Phi, i \in B \cup S \cup T \tag{3.27}$$

$$x_{ij}^k, \xi_{ij}^{\varphi} \in \{0, 1\} \tag{3.28}$$

$$\lambda_{ij} \geq 0 \tag{3.29}$$

3.4 Formulazione Alternativa del problema

In alternativa alla formulazione presentata nella sezione 3.3, che ha necessità di apportare varie modifiche al problema da ottimizzare, viene proposta in questa una seconda formulazione in grado di risolvere il problema senza aver necessità di modificare la rete di partenza.

3.4.1 Struttura del problema

Il problema da ottimizzare sarà un problema di tipo *Multicommodity flow* il cui grafo può essere descritto in questo modo:

- V è l'insieme dei nodi del problema, che vengono partizionati in nodi *client* V_c e *broker* V_b, in modo tale che $V = V_c \cup V_b$
- A è l'insieme degli archi presenti nel grafo, che nel nostro problema rappresentano le connessioni *Client-Broker* e *Broker-Broker*
- K è l'insieme delle *commodity* presenti nella rete, *commodities* che nel nostro problema modellano i *topic* presenti nel sistema.

A differenza della prima formulazione, per risolvere il problema è sufficiente aggiungere degli archi che colleghino le sorgenti con le destinazioni, dato che non è detto che esista un percorso a costo minimo interno alla rete analizzata e l'utilizzo di questi archi fittizi da parte della soluzione indicherà l'impossibilità di ottimizzare il problema con una determinata configurazione di rete.

In figura 3.5 è presente un esempio della trasformazione del grafo effettuata eseguendo le operazioni appena descritte: partendo dal grafo presente in figura 3.3, vengono aggiunti due archi che collegano le sorgenti con i nodi destinazione, lasciando invariato il resto della rete.

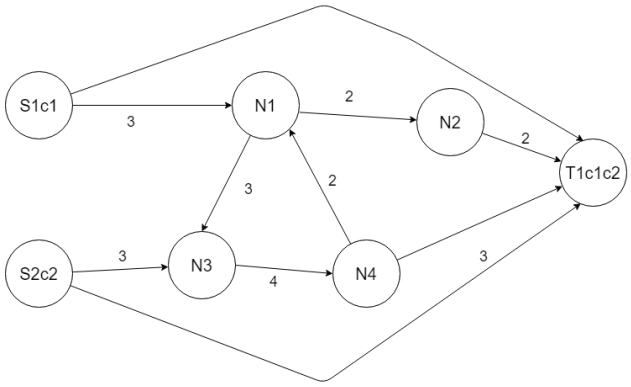

Figura 3.5: Aggiunta archi Sorgente-Destinazione seconda formulazione

3.4.2 Seconda Formulazione proposta

Utilizzando i concetti introdotti fin'ora viene in seguito esposta la formulazione matematica proposta per il problema analizzato: Definiamo x_{ij}^k come variabile binaria che assume valore 1 se la *commodity* $k \in K$ viene trasmessa dall'arco $\{i, j\}$. Definiamo $\xi_{i,j}^k$ la variabile positiva che assume valore pari al numero di *client* che ricevono la *commodity* $k \in K$ trasmessa dall'arco $\{i, j\}$. Definiamo a^k la variabile che indica il peso di ogni *Commodity* $k \in K$. Per ogni *commodity* definiamo s_k il nodo $s \in V_c$ che è sorgente della *commodity* k ed S_k l'insieme di nodi $s \in V_c$ in cui la *commodity* è richiesta. Per ogni arco $(i, j) \in A$ definiamo $u_{i,j}$ la capacità dell'arco. Per ogni nodo i e *commodity* k definiamo b_i^k la variabile che assume valore $b_i^k = |S_k|$ se $i = s_k$, $b_i^k = -1$ se $i \in S_k$ e $b_i^k = 0$ altrimenti. Infine definiamo $c_{i,j}$ la variabile che indica il costo che viene pagato per utilizzare l'arco $\{i, j\}$. La formulazione matematica è la seguente:

minimize

$$z = \sum_{k \in K} \sum_{(i,j) \in A} c_{ij} x_{ij}^k \tag{3.30}$$

subject to

$$\sum_{j \in \Gamma_i} \xi_{ij}^k - \sum_{j \in \Gamma_i^{-1}} \xi_{ji}^k = b_i^k \qquad k \in K, i \in V \tag{3.31}$$

$$\sum_{k \in K} a_k x_{ij}^k \leq u_{ij} \qquad (i, j) \in A \tag{3.32}$$

$$x_{ij}^k \leq \xi_{ij}^k \leq |S_k| x_{ij}^k \qquad k \in K, (i, j) \in A \tag{3.33}$$

$$\xi_{ij}^k \geq 0 \qquad k \in K, (i, j) \in A \tag{3.34}$$

$$x_{ij}^k \in \{0, 1\} \qquad k \in K, (i, j) \in A \tag{3.35}$$

- L'equazione 3.30 rappresenta la funzione obiettivo del nostro problema. Per ottimizzare il nostro problema la variabile $c_{i,j}$ assumerà un valore

$c_{i,j} = M_{i,j}$, con $M_{i,j}$ che assumerà un valore intero positivo adeguatamente grande per ogni arco tra sorgenti e destinazioni aggiunti al grafo e 1 altrimenti, per modellare il costo di utilizzare un arco che non è realmente presente nella rete. In questo modo l'obiettivo sarà quello di minimizzare il costo del flusso interno alla rete penalizzando il flusso che passa direttamente tra nodi sorgenti e destinazioni

- L'equazione 3.31 rappresenta la condizione di conservazione del flusso interna alla rete.
- L'equazione 3.32 rappresenta il vincolo di capacità degli archi.
- L'equazione 3.33 rappresenta i vincoli utilizzati per legare i due set di variabili decisionali x e ξ.
- Le equazioni 3.34 e 3.35 rappresentano infine il vincolo di positività per le variabili ξ^k ed il vincolo di assumere solo valori $\{0, 1\}$ per le variabili x^k

Capitolo 4

Sistema Implementato

Dopo aver introdotto i concetti tecnologici che caratterizzano il problema affrontato nei capitoli 1 e 2 ed i concetti matematici utilizzati per ottimizzarlo nel capitolo 3 è ora possibile descrivere la struttura e gli strumenti utilizzati per realizzare il sistema. Dato che l'obiettivo primario del sistema è l'ottimizzazione delle formulazioni matematiche presentate nelle sezioni 3.3.3 e 3.3.4 il componente principale del sistema è il *solver* utilizzato per risolvere i problemi di programmazione lineare. Questo capitolo mira ad illustrare le tecnologie utilizzate dal sistema, il *solver* adottato e la struttura del sistema.

4.1 Tecnologie adottate

Per lo sviluppo del sistema si sono utilizzate le seguenti tecnologie:

- C#
- C++
- Visual Studio
- git
- NuGet

L'adozione di queste tecnologie è giustificata dalla necessità di implementare un sistema portabile, scalabile e con alte prestazioni, oltre al fatto che sono tecnologie ampiamente utilizzate e dunque ben documentate e facili da manutenere anche da sviluppatori che non hanno lavorato al sistema durante l'implementazione.

4.1.1 C#

Per quanto riguarda la scelta di C# come linguaggio principale utilizzato nel progetto, ci sono varie motivazioni che hanno portato alla sua adozione. Nel corso degli anni il linguaggio si è costantemente evoluto ed aggiornato rendendo il suo utilizzo sempre più diffuso. I recenti sviluppi di C# ed in particolare il rilascio di .NET 5.0 [15], oltre all'introduzione di funzionalità che lo hanno avvicinato ad uno stile di programmazione più funzionale, hanno reso più semplice scrivere applicazioni in questo linguaggio che siano *cross-platform*. Altri fattori che hanno inciso sulla sua scelta sono il fatto che sia un linguaggio con buone prestazioni e scalabilità oltre al fatto di essere uno dei linguaggi più utilizzati nel campo della programmazione matematica e predittiva.

4.1.2 C++

L'uso di C++, sebbene in maniera non approfondita, è dovuto al fatto che la maggior parte dei solver di problemi di programmazione lineare sono scritti in questo linguaggio. Nonostante sia possibile utilizzare codice scritto in C++ in progetti scritti in C#, qualora sia necessario modificare il solver originale come descritto in maniera più approfondita nella sezione 4.2.7, l'utilizzo di C++ si rende obbligatorio.

4.1.3 Visual Studio

L'uso di Visual Studio è motivato dal fatto che è un IDE completo con supporto sia per C# che per C++. Sebbene ci siano altri IDE come Visual Code, Atom, Sublime text, ecc. Visual Studio è considerato l'IDE più potente per lo sviluppo e più completo per la quantità di caratteristiche che possiede. L'IDE permette anche di gestire in maniera più agile la compilazione ed il *deploy* delle applicazioni.

4.1.4 NuGet

NuGet [16] è il gestore di pacchetti per .NET e fornisce gli strumenti per produrre e pubblicare pacchetti. Inoltre la NuGet Gallery è il repository centrale di pacchetti .NET e il suo utilizzo facilita la pubblicazione dei lavori sviluppati utilizzando la piattaforma .NET. Il suo utilizzo è stato fondamentale per gestire l'integrazione del solver utilizzato dal sistema all'interno del progetto.

4.1.5 Git

Git è il *distributed version control system*(DVCS) open source più utilizzato ed integrato con la maggior parte dei progetti software moderni. L'utilizzo di git è necessario ed incoraggiato sia per tenere traccia del lavoro svolto che per favorire la condivisione ed il lavoro di diverse persone sullo stesso progetto.

4.2 Solver utilizzato

Dato che già esistono vari solver per la programmazione lineare, sia commerciali come ad esempio CPLEX di IBM [7] e GUROBI [9] sia non commerciali come GNU Linear Programming Kit (GLPK) [8] e COIN-OR Branch-and-Cut (CBC) [1], per lo sviluppo del sistema si è scelto di utilizzare un solver

già esistente. Il solver scelto è stato CoinMP [6], un solver *open source* e dunque di libero utilizzo.

Il solver CoinMP, componente del progetto Coin-OR [3, 4], è mirato a fornire una Windows dynamic linked library (DLL) che fornisca in modo semplice ed in un' unico progetto la maggior parte delle funzionalità dei progetti COIN-OR LP(CLP)[1], COIN-OR Branch-and-Cut(CBC)[2] and Cut Generation Library(CGL)[3] che sono pubblicamente accessibili su github.

Oltre a fornire una libreria in formato dll è possibile compilare il progetto CoinMP in modo da essere utilizzato anche su sistemi operativi Unix-like, fattore molto importante al momento della scelta di questo progetto come solver dato che la portabilità è un tema sempre sensibile ed importante in un progetto informatico.

4.2.1 Progetto COIN-OR

Il progetto COIN-OR è stato annunciato per la prima volta nel 2000 durante il diciassettesimo simposio sulla programmazione matematica di Atlanta ed è gestito dalla COIN-OR Foundation. La scopo della fondazione è di creare e diffondere conoscenza riguardo tutti gli aspetti della computazione relativi alla ricerca operativa. Il progetto COIN-OR è dunque composto da vari sotto progetti e moduli ognuno relativo a determinati problemi di ricerca operativa. Per quanto concerne il sistema implementato i moduli più importanti sono rappresentati da COIN-OR Linear Programming(CLP) e COIN-OR Branch and Cur (CBC).

[1]Il progetto COIN-OR Linear Programming (LP) è un solver di problemi di programmazione lineare open-source, che fornisce implementazioni sia di un solver per il simplesso che di uno per il simplesso duale

[2]Il progetto COIN-OR branch and cut fornisce un'implementazione open-source di un solver per problemi di programmazione lineare mista intera

[3]Il progetto COIN-OR cut generation library fornisce varie implementazioni di generatori di tagli utilizzati dagli algoritmi di ottimizzazione, ed è usato sia da CBC che da CLP

4.2.2 CoinMP

Il progetto CoinMP è nato per fornire una semplice interfaccia di utilizzo per i progetti CLP, CBC e CGL citati nella sezione 4.2, per fare ciò mette a disposizione una Windows dynamic linked library (DLL) da includere nei propri progetti per ottenere le funzionalità di ottimizzazione richieste. Il codice è scritto utilizzando il linguaggio C++ e presenta dipendenze da molti altri progetti COIN-OR. Oltre ai già citati CLP e CBC e CGL le altre dipendenze notevoli sono:

- **CoinUtils**: un insieme di classi e funzioni che sono generalmente utili in molti progetti COIN-OR. Alcune di queste funzionalità notevoli sono classi per manipolare e memorizzare matrici sparse e vettori, fattorizzare le matrici, costruire rappresentazioni di programmi matematici e comparare i numeri float con una certa tolleranza.
- **COIN-OR Open Solver Interface (OSI)**: fornisce una classe astratta per rappresentare un solver generico per la programmazione lineare (LP). I programmi scritti seguendo lo standard OSI possono essere collegati a qualunque solver con un'interfaccia OSI e dovrebbero produrre risultati corretti. OSI fornisce supporto alle seguenti operazioni:
  - Creazione della formulazione LP.
  - Modifica della formulazione aggiungendo direttamente righe/colonne.
  - Modifica della formulazione aggiungendo tagli forniti da CGL.
  - Risoluzione della formulazione.
  - Estrazione delle informazioni della soluzione.
  - Invocazione del componente Branch and Bound del solver che la implementa.

  Oltre che da progetti appartententi a COIN-OR questa interfaccia è utilizzata in molti altri solver sia commerciali come CPLEX e GUROBI che open source come GLPK.

4.2.3 Compilare un progetto ed ottenere le librerie

La maggior parte dei progetti che fanno parte di COIN-OR sono scritti in linguaggio C++ e sono liberamente accessibili su github. I progetti sono impostati in modo da essere utilizzabili da Visual Studio, dato che ognuno ha la sua soluzione visual e le dipendenze necessarie a funzionare. È inoltre possibile utilizzare il progetto COIN-OR coinbrew [5] per scaricare e compilare il componente che si desidera utilizzare: coinbrew fornisce gli strumenti per scaricare progetti, le loro dipendenze e compilarli.

4.2.4 Problemi e soluzioni del progetto CoinMP

Dal momento che il progetto CoinMP fornisce un'interfaccia per l'utilizzo di altri componenti COIN-OR, non è aggiornato e manutenuto tanto spesso quanto gli altri componenti. Per questo motivo l'ultima versione disponibile è stata rilasciata nel 2019 ed è stata compilata utilizzando versioni precedenti rispetto alle attuali release di CLP e CBC. Inoltre i progetti Visual Studio inclusi nel *repository* non sono aggiornati e per scaricare il codice e compilare il progetto utilizzando versioni recenti delle dipendenze necessarie, occorre ottenere manualmente i vari progetti da cui dipende ed aggiornare le dipendenze in Visual Studio. Dopo gli aggiornamenti è possibile compilare il codice utilizzando l'IDE per ottenere una DLL utilizzabile su windows, oppure utilizzare coinbrew per compilare il progetto ed ottenere il file in formato Shared Object (SO), equivalente delle DLL per sistemi Linux.

4.2.5 Studio di metodologie di interoperabilità tra codice C++ e C#

La principale difficoltà nell'utilizzare la libreria CoinMP all'interno di un progetto scritto in C# e dunque codice managed è dovuta al fatto che è una libreria scritta in C++ e quindi codice unmanaged. Per ovviare al problema è stata utilizzata la tecnologia Platform Invoke [18] che permette al codi-

ce gestito di accedere a funzioni, strutture e callback di librerie scritte in linguaggi non gestiti.

Codice managed e unmanaged

Quando si parla di codice managed o gestito si fa riferimento a codice che non viene compilato in codice macchina ma viene eseguito da un Runtime [2], che nel caso di .NET è denominato Common Language Runtime o CLR.

La caratteristica principale del codice gestito è che vengono forniti diversi servizi importanti come ad esempio la gestione automatica della memoria, dei limiti di sicurezza, dell'indipendenza dai tipi e così via.

Con codice unmanaged o non gestito si fa riferimento al codice che viene compilato in linguaggio macchina e viene eseguito direttamente dal sistema operativo. Il modo in cui viene eseguito un programma C/C++ è dunque opposto a quanto avviene per il codice gestito, fornendo più possibilità al programmatore che può ad esempio lavorare direttamente con la memoria utilizzando i puntatori, ma anche più responsabilità, dato che non è presente nessun servizio che si occupi della gestione della memoria, della sicurezza e delle altre problematiche risolte dal codice gestito.

Platform Invoke

La tecnologia Platform Invoke (P/Invoke) permette di accedere a strutture, funzioni e callback di librerie non gestite da codice gestito C#. La maggior parte delle funzionalità dell'API P/Invoke è contenuta in due namespaces: System and System.Runtime.InteropServices. Utilizzare questi namespaces permette di descrivere come comunicare ed interagire con il componente nativo non gestito.

Nel seguente pezzo di codice 4.1 è presente un esempio di utilizzo della tecnologia P/Invoke: Utilizzando l'attributo **DllImport("NomeLibreria")** viene comunicato al runtime che deve caricare la libreria non gestita. La riga che segue l'attributo **DllImport** è necessaria per la definizione di un metodo gestito, metodo che deve avere la stessa signature di quello non gestito. La

```
using System;
using System.Runtime.InteropServices;

public class Program
{
        [DllImport("CoinMP")]
        static extern IntPtr CoinCreateProblem(string problemName);
}
```

Figura 4.1: Codice esempio di utilizzo P/Invoke

parola chiave extern è utilizzata per dire al runtime che il metodo definito è esterno e che quando invocato il runtime dovrebbe trovarlo nella libreria specificata nell'attributo **DllImport**. Il tipo di ritorno *IntPtr* della funzione è un tipo di dato utilizzato per rappresentare un puntatore generico e memorizzare un possibile tipo di dato che viene restituito da metodi unmanaged che non hanno un tipo equivalente in codice managed.

4.2.6 Implementazione di un Wrapper per la libreria CoinMP

Dato che per utilizzare il solver CoinMP all'interno di progetti C# è necessario aggiungere del codice aggiuntivo che renda possibile la comunicazione tra codice managed ed unmanaged, si è deciso di sviluppare e pubblicare come sottoprogetto una libreria Wrapper che permettesse di utilizzare il solver in maniera semplice e trasparente in qualsiasi nuovo progetto sviluppato in C#.

Struttura del sotto-progetto Wrapper

Dopo aver terminato l'analisi di CoinMP e lo studio della tecnologia P/Invoke, il passo successivo nello sviluppo della libreria è stato la definizione del tipo di progetto da implementare e la sua effettiva implementazione.

Considerando la portabilità come requisito fondamentale della libreria da sviluppare, la prima scelta effettuata è stata quella di utilizzare il framework .NET 5.0, che permette di utilizzare il software prodotto sia in ambiente Windows che Unix-like. Il secondo fattore preso in considerazione riguarda la tipologia di progetto da utilizzare: in questo caso si è scelto di creare un progetto di libreria utilizzando Visual Studio, in modo da avere come risultato del progetto una libreria managed utilizzabile da qualsiasi altro progetto sviluppato usando il framework .NET 5.0.

Un altro vantaggio di utilizzare questo tipo di progetto è il supporto fornito dall'IDE alla creazione di un paccheto NuGet che rende la pubblicazione della libreria più semplice e ne permette l'utilizzo da parte di altri sviluppatori in maniera più agevole, dato che è sufficiente usare il manager di pacchetti NuGet per utilizzare la libreria così prodotta.

Implementazione wrapper e problemi CoinMP

-Implementazione wrapper

Utilizzando gli strumenti definiti nelle sezioni precedenti si è proceduto all'implementazione della libreria. Il progetto è stato suddiviso in 3 sotto progetti:

- **WrapperCoinMP**: il progetto principale che utilizza la tecnologia P/Invoke per wrappare la libreria CoinMP e fornisce le strutture ed i metodi per interagire con essa tramite codice managed utilizzando il linguaggio C#.
- WrapperTest: progetto utilizzato per effettuare Unit Test del progetto principale.
- TestMain: progetto che fornisce un semplice programma che mostra il possibile utilizzo della libreria sviluppata.

-Problemi CoinMP

Oltre ai problemi di aggiornamento del progetto e della difficoltà di ot-

tenere e compilare una versione aggiornata della libreria citati nella sezione 4.2.4, CoinMP presenta alcuni problemi che sono emersi durante lo sviluppo del wrapper. Il problema principale che affligge CoinMP è che pur utilizzando l'interfaccia OSI citata nella sezione 4.2.2, non mette a disposizione metodi per aggiungere righe e colonne ad un problema caricato, rendendo difficile e laborioso lavorare su un problema di ottimizzazione che può cambiare dinamicamente durante l'esecuzione. Dato che queste funzionalità sono cruciali per l'utilizzo di un solver, la soluzione adottata in questo progetto è stata quella di modificare direttamente la libreria CoinMP per aggiungere le funzioni richieste, come spiegato nella sezione 4.2.7.

Altri problemi minori e che quindi non sono stati affrontati, riguardano la mancata implementazione di alcune funzioni che sono però presenti nell'interfaccia della libreria CoinMP, come ad esempio la mancata implementazione della funzione utilizzata per leggere un problema da file[4].

4.2.7 Modifica della libreria CoinMP

Come spiegato nella sezione 4.2.6 la libreria CoinMP non mette a disposizione dei metodi per aggiungere righe e colonne ai problemi caricati. Per risolvere questo problema si è deciso di aggiungere i metodi necessari direttamente nella libreria CoinMP, essendo CoinMP un progetto open source liberamente modificabile.

Modifiche al codice

Per modificare la libreria, come primo approccio si è scaricata una copia del codice direttamente dal repository pubblico del progetto e sono stati sistemati i problemi di dipendenze citati nella sezione 4.2.4. Dato che il progetto è scritto in linguaggio C++ le modifiche apportate sono state fatte utilizzando questo linguaggio. Le funzioni aggiunte alla libreria CoinMP sono le seguenti:

[4]La funzione per scrivere un problema su file è invece presente.

- CoinAddRow: funzione che permette l'aggiunta di una riga ad un problema caricato in memoria, modificando in maniera esplicita i vari puntatori alle strutture utilizzate per memorizzare il problema.
- CoinAddColumn: funzione che permette l'aggiunta di una colonna ad un problema caricato in memoria, modificando in maniera esplicita i vari puntatori alle strutture utilizzate per memorizzare il problema.
- CoinNullifyRow: funzione che permette di annullare i coefficienti di una specifica riga in un problema caricato in memoria: questa funzione è stata aggiunta perché CoinMP non mette a disposizione dei metodi per modificare le righe di un problema, e per ciò non offre la possibilità di eliminare dei vincoli durante il suo utilizzo.

Rilascio della libreria modificata

Dopo aver completato e testato le modifiche effettuate alla libreria sulla versione privata del progetto, si sono presi contatti con l'associazione COIN-OR per sapere se la modifica effettuata potesse essere utile al progetto: è stata dunque suggerita la possibilità di rilasciare le modifiche effettuate al progetto in una fork del codice di base e di effettuare una pull request per eventualmente aggiungere le modifiche al progetto.

4.2.8 Completamento e rilascio di WrapperCoinMP

Dopo aver modificato la libreria CoinMP come spiegato nella sezione precedente si è proceduto a completare il wrapper con i metodi aggiunti a CoinMP ed effettuare la compilazione e rilascio del progetto.

Come specificato nella sezione 4.2.6 oltre ad effettuare un classica release del progetto è stato anche creato un pacchetto NuGet della libreria. Utilizzando il manager di pacchetti è possibile aggiungere la libreria WrapperCoinMP a qualunque progetto che utilizzi il framework .NET 5.0.

4.3 Sistema finale

Dopo aver implementato il *wrapper* per poter utilizzare il solver CoinMP presentato nella sezione precedente si è proceduto all'implementazione di un sistema in grado di leggere dati relativi ad una rete da ottimizzare, modellarne il grafo, applicare le formulazioni matematiche presentate nella sezione 3.3, creare un problema corretto ed ottimizzabile da CoinMP ed infine presentare i risultati dell'ottimizzazione.

4.3.1 Lettura e modellazione del grafo di rete

Per modellare il grafo sono stati creati vari *record*, ognuno con lo scopo di modellare un determinato concetto (e.g. nodi del grafo, archi, *commodities* etc.). È possibile dunque costruire o modificare un grafo tramite l'invocazione di diversi metodi forniti dal sistema, per permettere si l'inizializzazione di un problema a *runtime* che la modifica di problemi già caricati per testare diverse configurazioni di rete possibili. È inoltre possibile ed incoraggiato l'utilizzo del formato JSON per la modellazione ed il caricamento di un problema, per permettere di ottimizzare reti predefinite e salvate in un formato comune ed *Human readable*.

Formato JSON del grafo

Per poter salvare e leggere delle rappresentazioni di reti in formato JSON, è stato definito un formato in cui salvare i campi che rappresentano la struttura del grafo. Utilizzando come grafo d'esempio il grafo presente in figura 4.2 la sua rappresentazione JSON è mostrata in figura 4.3:

All'interno dell' *array* "CommoditySources" è presente il campo "Weight", che indica il peso della commodity generato da una sorgente e dunque la quantità di banda richiesta alla sua trasmissione[5]. Per quanto riguarda l'*array* che

[5]Lo stesso campo è presente nell'*array* "CommoditySinks" per favorire il riutilizzo ed un eventuale futura evoluzione in cui una sorgente richiede diverse copie della stessa *commodity*. Attualmente il campo viene ignorato

rappresenta gli archi del grafo, viene utilizzato il campo "Capacity" per indicare la capacità dell'arco; in questo caso oltre a fissare un valore specifico per un arco è possibile assegnargli capacità infinita settando il valore del campo a -1.

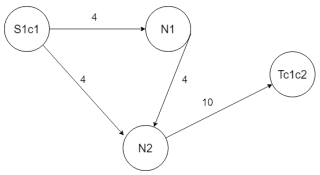

Figura 4.2: Grafo d'esempio per formato JSON

```
{
  "Nodes": [ "N1","N2"],
  "Commodities": [ "c1", "c2" ],
  "CommoditiesSources": [
   {
      "Name": "S",
      "Commodity": "c1",
      "Weight": 4
    },
    {
      "Name": "S",
      "Commodity": "c2",
      "Weight": 4
    }
  ],
  "CommoditiesSinks": [
    {
      "Name": "T",
      "Commodity": "c1",
      "Weight": -1
    },
    {
      "Name": "T",
      "Commodity": "c2",
      "Weight": -1
    }
  ],
  ...
```

```
...
{
  "Edges": [
    {
      "Source": "S",
      "Destination": "N1",
      "Capacity": 4
    },
    {
      "Source": "S",
      "Destination": "N2",
      "Capacity": 4
    }
    {
      "Source": "N1",
      "Destination": "N2",
      "Capacity": 4
    }
    {
      "Source": "N2",
      "Destination": "T",
      "Capacity": 8
    }
  ]
}
```

Figura 4.3: Esempio di grafo in formato JSON

4.3.2 Modifica del grafo e formulazione matematica

Dopo aver inizializzato un grafo da ottimizzare, sia se costruito manualmente che se letto da JSON, il sistema si occupa di applicare le trasformazioni descritte nella sezione 3.3.2 in caso si applichi la prima formulazione proposta

nella sezione 3.3.3, oppure le modifiche descritte nella sezione 3.5 in caso si applichi la seconda formulazione proposta nella sezione 3.4.2. A questo punto è possibile inizializzare il problema di programmazione lineare creando le strutture necessarie al *solver* CoinMP per ottimizzare il problema. Durante questa fase è possibile salvare il problema di programmazione lineare ottenuto sia in formato CSV che MPS[6] oppure di farlo stampare a schermo.

4.3.3 Ottimizzazione e salvataggio del risultato

Dopo aver inizializzato il *solver* con la formulazione matematica generata, è a questo punto possibile ottimizzare il problema. Il *solver* procederà ad applicare gli algoritmi necessari alla risoluzione del problema e fornirà la soluzione cercata, soluzione che è a questo punto visualizzabile graficamente a schermo e salvabile in formato JSON per fornire un quadro più completo della rete e dei flussi ottimali risultato dell'ottimizzazione.

[6]Il formato MPS è il formato standard utilizzato per salvare modelli di problemi lineari

Capitolo 5

Casi d'uso

Dopo aver introdotto il paradigma *Publish/Subscribe* nel capitolo 1, la descrizione del problema da risolvere nel capitolo 2, le formulazioni matematiche proposte nel capitolo 3 e l'implementazione del sistema nel capitolo 4, in questo capitolo si procederà ad illustrare alcuni casi d'uso del sistema implementato ed i rispettivi risultati ottenuti applicando le tre formulazioni matematiche proposte. Per il primo caso d'uso sono presentati a titolo d'esempio anche le rappresentazioni matriciali del problema in formato CSV ed i risultati delle ottimizzazioni in formato JSON, rappresentazioni che sono state omesse per i successivi casi d'uso data l'eccessiva dimensione delle stesse che le avrebbe dunque rese illeggibili.

5.1 Caso d'uso 1

Per il primo caso d'uso preso in considerazione è stata utilizzata la rete rappresentata dal grafo presente in figura 4.3. La rete è caratterizzata dalla presente di una sorgente che genera due *commodity* di peso 4, richieste da un *client* che le consuma entrambe e con due nodi *broker* interni.

5.1.1 Prima Formulazione

Formulazione matematica

Dopo aver caricato il problema dalla sua rappresentazione in formato JSON, il sistema si occupa di applicare al grafo le trasformazioni presentate in sezione 3.3.2 e di generare poi le righe che rappresentano i vincoli utilizzati per inizializzare il *solver* prima di ottimizzare il problema. In tabella 5.1 è presente la rappresentazione dei vincoli del problema in forma matriciale, ottenuta facendo salvare al sistema i vincoli applicati al problema in formato csv. In particolare la prima riga della tabella rappresenta la funzione obiettivo, con la variabile $p_{i,j} = 1000$ quando associata all'arco $\{i, j\}$ che collega le sorgenti con le destinazioni *Dummy* e 0 altrimenti. Le altre righe della tabella rappresentano i rimanenti vincoli applicati al problema.

STc2_S→N1	STc2_S→N2	STc2_N1→N2	STc2_T→Tc1	STc2_S→Tc1	STc2_N2→T	STc2_T→Tc2	STc2_S→Tc2	STc1_S→N1	STc1_S→N2	STc1_N1→N2	STc1_T→Tc1	STc1_S→Tc1	STc1_N2→T	STc1_T→Tc2	STc1_S→Tc2	Xic1S→N1	Xic1S→N2	Xic1N1→N2	Xic1T→Tc1	Xic1S→Tc1	Xic1N2→T	Xic1T→Tc2	Xic1S→Tc2	Xic2S→N1	Xic2S→N2	Xic2N1→N2	Xic2T→Tc1	Xic2S→Tc1	Xic2N2→T	Xic2T→Tc2	Xic2S→Tc2		
0	0	0	0	1000	0	0	1000	0	0	0	0	1000	0	0	1000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	MIN	
0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	E
1	1	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	0	0	0	0	0	1	E
0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	E
0	0	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	0	0	0	0	0	E
0	0	0	0	0	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	0	1	E
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	E
1	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	E
0	0	0	0	0	0	0	0	1	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	E
0	1	1	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	E
0	0	0	0	0	0	0	0	0	1	1	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	E
0	0	0	-1	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	0	0	0	E
0	0	0	0	0	0	0	0	0	0	0	-1	0	1	-1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	E
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	-100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	L
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	-100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	L
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	-100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	L
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	-100	0	0	0	0	0	0	0	0	0	0	0	L
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-100	0	0	0	0	0	0	0	0	L
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-100	0	0	0	0	0	0	0	L
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-100	0	0	0	0	0	0	L
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-100	0	0	0	L
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	4	L
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	4	0	0	0	0	0	0	4	L
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	4	0	0	0	0	0	4	L
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	4	0	0	0	0	1.7E+308	L
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	4	0	0	0	1.7E+308	L
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	4	0	0	10	L
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	4	0	1.7E+308	L
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	4	1.7E+308	L

Tabella 5.1: Rappresentazione matriciale dei vincoli del problema in prima formulazione

Risultato dell'ottimizzazione

Dopo l'inizializzazione, il solver si occupa di ottimizzare il problema e di presentare la soluzione trovata. In figura 5.1 è presente una parte della struttura utilizzata per salvare lo stesso risultato in formato JSON, da cui sono stati gli archi con valore 0 e dunque non utilizzati. Da notare il valore pari a 0 del risultato della funzione obiettivo, che indica il fatto che il solver è riuscito ad ottimizzare il problema senza dover utilizzare gli archi fittizi presenti tra sorgenti e destinazioni *dummy*.

In figura 5.2 è presente l'output testuale del risultato dell'ottimizzazione in forma completa ed infine in figura 5.3 è visibile la struttura della rete finale da cui sono stati rimossi gli archi inutilizzati: come è possibile notare la rete ottima ha necessità di utilizzare entrambi i broker per poter consegnare le *commodities* richieste al nodo destinazione, con il nodo N2 che deve aprire due connessioni per consegnare le *commodities* diverse al nodo T.

```
{
  "EdgesResult": [
    {
      "Source": "S",
      "Destination": "N1",
      "Commodity": "STc2",
      "Value": 1.0
    },
    {
      "Source": "N1",
      "Destination": "N2",
      "Commodity": "STc2",
      "Value": 1.0
    },
    {
      "Source": "N2",
      "Destination": "T",
      "Commodity": "STc2",
      "Value": 1.0
    },
    {
      "Source": "T",
      "Destination": "Tc2",
      "Commodity": "STc2",
      "Value": 1.0
    },
    {
      "Source": "S",
      "Destination": "N2",
      "Commodity": "STc1",
      "Value": 1.0
    },
    {
      "Source": "T",
      "Destination": "Tc1",
      "Commodity": "STc1",
      "Value": 1.0
    },
    {
      "Source": "N2",
      "Destination": "T",
      "Commodity": "STc1",
      "Value": 1.0
    },
  ],
  "Objective": 0.0
```


Figura 5.1: Esempio risultato in formato JSON

```
The optimized total flow value is 0, and the edges optimized values are
 The source node is S, the destination node is N1, the commodity is STc2, the optimized value for the edge is 1
The source node is S, the destination node is N2, the commodity is STc2, the optimized value for the edge is 0
The source node is N1, the destination node is N2, the commodity is STc2, the optimized value for the edge is 1
The source node is T, the destination node is Tc1, the commodity is STc2, the optimized value for the edge is 0
The source node is S, the destination node is Tc1, the commodity is STc2, the optimized value for the edge is 0
The source node is N2, the destination node is T, the commodity is STc2, the optimized value for the edge is 1
The source node is T, the destination node is Tc2, the commodity is STc2, the optimized value for the edge is 1
The source node is S, the destination node is Tc2, the commodity is STc2, the optimized value for the edge is 0
The source node is S, the destination node is N1, the commodity is STc1, the optimized value for the edge is 0
The source node is S, the destination node is N2, the commodity is STc1, the optimized value for the edge is 1
The source node is N1, the destination node is N2, the commodity is STc1, the optimized value for the edge is 0
The source node is T, the destination node is Tc1, the commodity is STc1, the optimized value for the edge is 1
The source node is S, the destination node is Tc1, the commodity is STc1, the optimized value for the edge is 0
The source node is N2, the destination node is T, the commodity is STc1, the optimized value for the edge is 1
The source node is T, the destination node is Tc2, the commodity is STc1, the optimized value for the edge is 0
The source node is S, the destination node is Tc2, the commodity is STc1, the optimized value for the edge is 0
```

Figura 5.2: Risultato testuale dell'ottimizzazione del caso d'uso 1 utilizzando la prima formulazione matematica

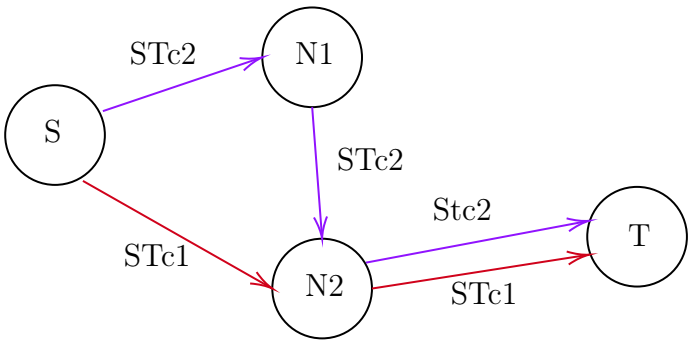

Figura 5.3: Grafo risultato dell'ottimizzazione del caso d'uso 1 utilizzando la prima formulazione matematica

5.1.2 Prima Formulazione - Rilassamento Lagrangiano

Formulazione matematica

Per applicare la formulazione lagrangiana esposta in sezione 3.1.1 il sistema effettua le stesse operazioni effettuate per applicare la prima formulazione, con la differenza che in questo caso rimuove alcuni vincoli e li aggiunge con penalità alla funzione obiettivo. In tabella 5.2 è presente la rappresentazione dei vincoli del problema in forma matriciale: in questo caso sono presenti meno vincoli, mentre alla funzione obiettivo è ottenuta utilizzando un $\lambda_{i,j}$ pari al valore costante 20.

	$x_{1,2,4}$	$x_{1,3,4}$	$x_{2,1,3}$	$x_{2,1,4}$	$x_{2,3,1}$	$x_{2,3,4}$	$x_{2,4,1}$	$x_{2,4,3}$	$x_{3,1,2}$	$x_{3,1,4}$	$x_{3,2,1}$	$x_{3,2,4}$	$x_{3,4,1}$	$x_{3,4,2}$	$x_{4,1,2}$	$x_{4,1,3}$	$x_{4,2,1}$	$x_{4,2,3}$	$x_{4,3,1}$	$x_{4,3,2}$	$x_{1,2,3}$	$x_{1,3,2}$	$x_{1,4,2}$	$x_{1,4,3}$	$x_{2,1,4}$	$x_{2,3,4}$	$x_{2,4,3}$	$x_{3,1,4}$	$x_{3,2,4}$	$x_{3,4,2}$	$x_{4,1,3}$	$x_{4,2,3}$	$x_{4,3,2}$	$x_{1,2,3}$	$x_{1,3,2}$	$x_{1,4,2}$	$x_{1,4,3}$	$x_{2,1,3}$	$x_{2,3,1}$	$x_{2,4,1}$	$x_{2,4,3}$	$x_{3,1,2}$	$x_{3,2,1}$	$x_{3,4,1}$	$x_{3,4,2}$	$x_{4,1,2}$	$x_{4,2,1}$	$x_{4,3,1}$	$x_{4,3,2}$	
$x_{1,2,3}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	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	0	0	0
$x_{1,3,2}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	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	0	0	0
$x_{1,4,2}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	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	0	0	0
$x_{1,4,3}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	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	0	0	0
$x_{2,1,3}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$x_{2,1,4}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$x_{2,3,1}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$x_{2,3,4}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$x_{2,4,1}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$x_{2,4,3}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$x_{3,1,2}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$x_{3,1,4}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$x_{3,2,1}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0
$x_{3,2,4}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0
$x_{3,4,1}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0
$x_{3,4,2}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0
$x_{4,1,2}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0
$x_{4,1,3}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0
$x_{4,2,1}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
$x_{4,2,3}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
$x_{4,3,1}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
$x_{4,3,2}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
$\lambda_{1,2,3}$	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\lambda_{1,3,2}$	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\lambda_{1,4,2}$	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\lambda_{1,4,3}$	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\lambda_{2,1,3}$	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\lambda_{2,1,4}$	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\lambda_{2,3,1}$	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\lambda_{2,3,4}$	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\lambda_{2,4,1}$	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\lambda_{2,4,3}$	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\lambda_{3,1,2}$	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\lambda_{3,1,4}$	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\lambda_{3,2,1}$	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\lambda_{3,2,4}$	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\lambda_{3,4,1}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\lambda_{3,4,2}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\lambda_{4,1,2}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\lambda_{4,1,3}$	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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\lambda_{4,2,1}$	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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\lambda_{4,2,3}$	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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\lambda_{4,3,1}$	0	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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\lambda_{4,3,2}$	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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Tabella 5.2: Rappresentazione matriciale dei vincoli del problema lagrangiano

Risultato dell'ottimizzazione

In figura 5.4 è presente una parte della struttura utilizzata del risultato in formato JSON, da cui sono stati gli archi con valore 0 e dunque non utilizzati. In questo caso il valore della funzione obiettivo non potrà più essere 0, in quanto per utilizzare ogni arco bisognerà pagare un penalità. Il risultato di 320 ci indica comunque che il solver non ha dovuto ricorrere agli archi fittizi, dato che in caso li avesse utilizzati avrebbe aggiunto un valore pari a 1000 alla funzione obiettivo.

In figura 5.5 è presente l'output testuale del risultato dell'ottimizzazione in forma completa ed infine in figura 5.6 è visibile la struttura della rete finale da cui sono stati rimossi gli archi inutilizzati: utilizzando la formulazione lagrangiana, il sistema trova in questo caso una soluzione alternativa rispetto a quella trovata dalla prima formulazione, invertendo le destinazioni iniziali delle *commodities* a partire dalle sorgenti.

```
{
  "EdgesResult": [
    {
      "Source": "S",
      "Destination": "N2",
      "Commodity": "STc2",
      "Value": 1.0
    },
    {
      "Source": "N2",
      "Destination": "T",
      "Commodity": "STc2",
      "Value": 1.0
    },
    {
      "Source": "T",
      "Destination": "Tc2",
      "Commodity": "STc2",
      "Value": 1.0
    },
    {
      "Source": "S",
      "Destination": "N2",
      "Commodity": "STc1",
      "Value": 1.0
    },
    {
      "Source": "T",
      "Destination": "Tc1",
      "Commodity": "STc1",
      "Value": 1.0
    },
    {
      "Source": "N2",
      "Destination": "T",
      "Commodity": "STc1",
      "Value": 1.0
    },
  ],
  "Objective": 320.0
}
```


Figura 5.4: Esempio risultato lagrangiano in formato JSON

```
The optimized total flow value is 320, and the edges optimized values are
 The source node is S, the destination node is N1, the commodity is STc2, the optimized value for the edge is 0
The source node is S, the destination node is N2, the commodity is STc2, the optimized value for the edge is 1
The source node is N1, the destination node is N2, the commodity is STc2, the optimized value for the edge is 0
The source node is T, the destination node is Tc1, the commodity is STc2, the optimized value for the edge is 0
The source node is S, the destination node is Tc1, the commodity is STc2, the optimized value for the edge is 0
The source node is N2, the destination node is T, the commodity is STc2, the optimized value for the edge is 1
The source node is T, the destination node is Tc2, the commodity is STc2, the optimized value for the edge is 1
The source node is S, the destination node is Tc2, the commodity is STc2, the optimized value for the edge is 0
The source node is S, the destination node is N1, the commodity is STc1, the optimized value for the edge is 0
The source node is S, the destination node is N2, the commodity is STc1, the optimized value for the edge is 1
The source node is N1, the destination node is N2, the commodity is STc1, the optimized value for the edge is 0
The source node is T, the destination node is Tc1, the commodity is STc1, the optimized value for the edge is 1
The source node is S, the destination node is Tc1, the commodity is STc1, the optimized value for the edge is 0
The source node is N2, the destination node is T, the commodity is STc1, the optimized value for the edge is 1
The source node is T, the destination node is Tc2, the commodity is STc1, the optimized value for the edge is 0
The source node is S, the destination node is Tc2, the commodity is STc1, the optimized value for the edge is 0
```

Figura 5.5: Risultato testuale dell'ottimizzazione del caso d'uso 1 utilizzando la formulazione lagrangiana

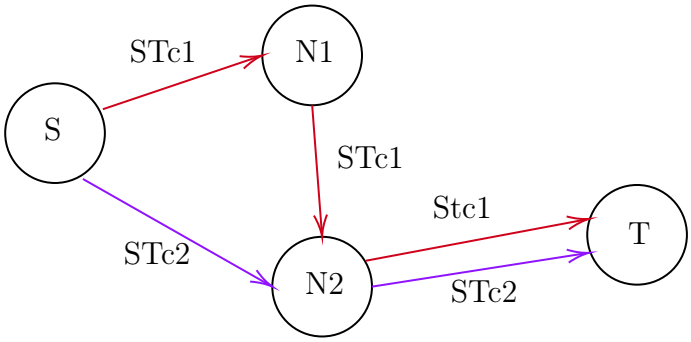

Figura 5.6: Grafo risultato dell'ottimizzazione del caso d'uso 1 utilizzando la formulazione lagrangiana

5.1.3 Seconda Formulazione

Formulazione matematica

Anche per applicare la seconda formulazione il sistema esegue lo stesso procedimento esposto in precedenza. In tabella 5.3 è presente la rappresentazione dei vincoli del problema in forma matriciale: a differenza delle due formulazioni precedenti, in questo caso la funzione obiettivo ha un costo associato ad ogni arco vero pari ad 1, mentre per gli archi fittizi tra sorgenti e destinazioni il costo è pari a 1000.

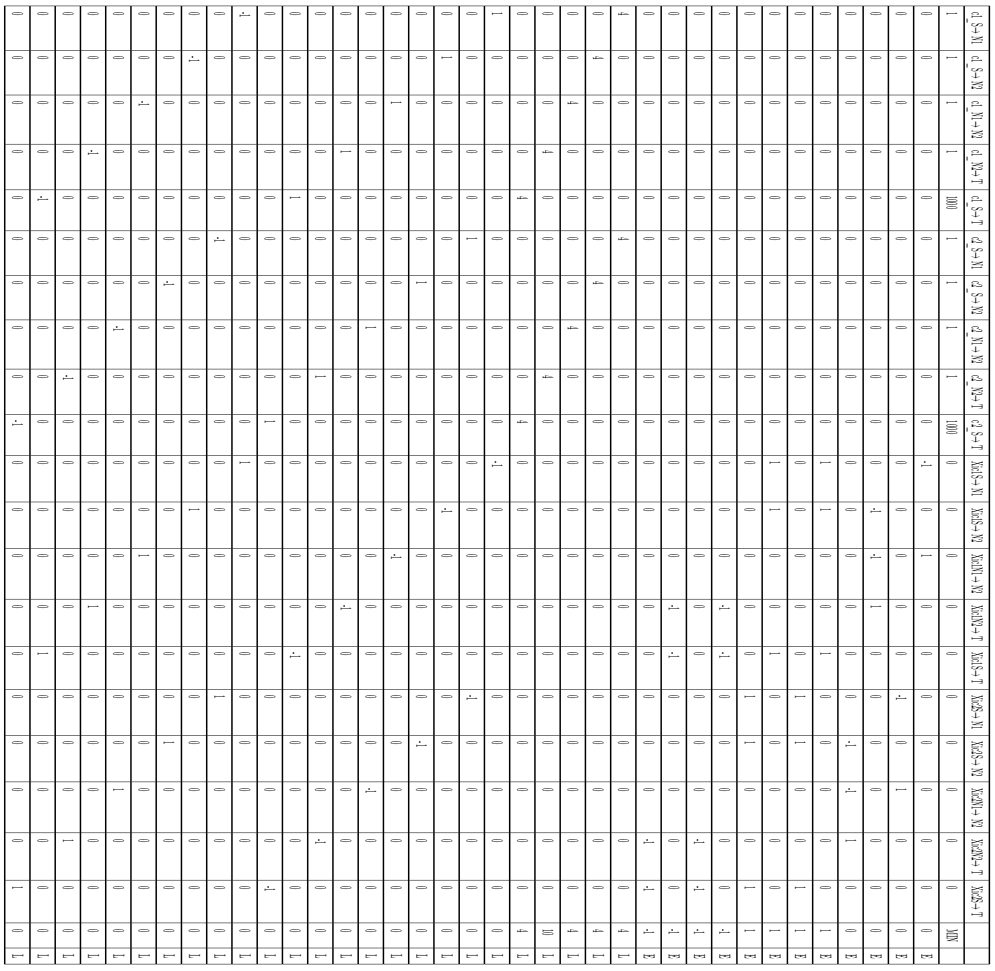

Tabella 5.3: Rappresentazione matriciale dei vincoli del problema in seconda formulazione

Risultato dell'ottimizzazione

In figura 5.7 è presente una parte della struttura utilizzata del risultato in formato JSON, da cui sono stati rimossi gli archi con valore 0 e dunque non utilizzati. In questo caso il valore della funzione obiettivo indica il numero di connessioni necessarie a consegnare le *commodity* richieste, risultato che verrebbe fortemente penalizzato in caso venisse utilizzata una connessione inizialmente non presente nel grafo.

In figura 5.8 è presente l'output testuale del risultato dell'ottimizzazione in forma completa ed infine in figura 5.9 è visibile la struttura della rete finale da cui sono stati rimossi gli archi inutilizzati: utilizzando la seconda formulazione per questo problema si ottiene la stessa soluzione trovata applicando la prima formulazione.


```
{
  "EdgesResult": [
    {
      "Source": "S",
      "Destination": "N2",
      "Commodity": "c1",
      "Value": 1.0
    },
    {
      "Source": "N2",
      "Destination": "T",
      "Commodity": "c1",
      "Value": 1.0
    },
    {
      "Source": "S",
      "Destination": "N1",
      "Commodity": "c2",
      "Value": 1.0
    },
    {
      "Source": "N1",
      "Destination": "N2",
      "Commodity": "c2",
      "Value": 1.0
    },
    {
      "Source": "N2",
      "Destination": "T",
      "Commodity": "c2",
      "Value": 1.0
    }
  ],
  "Objective": 5.0
}
```


Figura 5.7: Esempio risultato seconda formulazione in formato JSON

```
The optimized total flow value is 5, and the edges optimized values are
 The source node is S, the destination node is N1, the commodity is c1, the optimized value for the edge is 0
The source node is S, the destination node is N2, the commodity is c1, the optimized value for the edge is 1
The source node is N1, the destination node is N2, the commodity is c1, the optimized value for the edge is 0
The source node is N2, the destination node is T, the commodity is c1, the optimized value for the edge is 1
The source node is S, the destination node is T, the commodity is c1, the optimized value for the edge is 0
The source node is S, the destination node is N1, the commodity is c2, the optimized value for the edge is 1
The source node is S, the destination node is N2, the commodity is c2, the optimized value for the edge is 0
The source node is N1, the destination node is N2, the commodity is c2, the optimized value for the edge is 1
The source node is N2, the destination node is T, the commodity is c2, the optimized value for the edge is 1
The source node is S, the destination node is T, the commodity is c2, the optimized value for the edge is 0
```

Figura 5.8: Risultato testuale dell'ottimizzazione del caso d'uso 1 utilizzando la seconda formulazione matematica

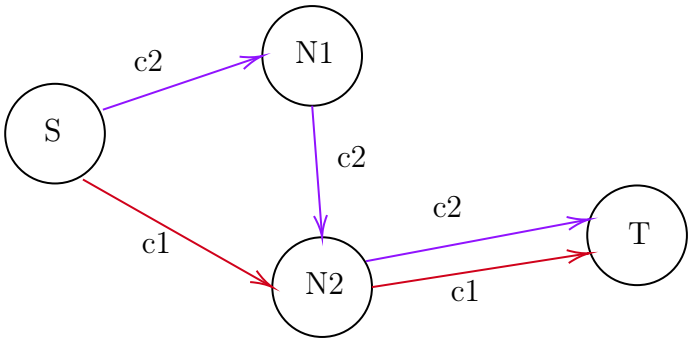

Figura 5.9: Grafo risultato dell'ottimizzazione del caso d'uso 1 utilizzando la seconda formulazione matematica

5.2 Caso d'uso 2

Come anticipato nella sezione 5 la presentazione del risultato di ottimizzazione per problemi anche solo leggermente più grandi di quello presentato nella sezione 5.1 risulta difficile per via della dimensione assunta dallo stesso. Pertanto i risultati dei prossimi casi d'uso verranno presentati tramite l'output testuale ottenuto dal sistema, da cui saranno rimossi tutti gli archi non utilizzati, e tramite la rappresentazione del grafo ottimo ottenuto grazie all'ottimizzazione.

5.2.1 Rete analizzata

Per il secondo caso d'uso è stata presa in considerazione una rete più complessa rispetto a quella utilizzata nel primo caso. In figura 5.10 è presente una rappresentazione della rete, rete che può essere così descritta:

- *Commodity*: Sono presenti tre diverse *commodity*:
  - k1 con peso 45
  - k2 con peso 35
  - k3 con peso 55
- Sorgenti: sono presenti due diverse sorgenti:
  - S1, che produce le *commodity* k1 e k2
  - S2, che produce la *commodity* k3
- Destinazioni: sono presenti due destinazioni:
  - T1, che consuma le *commodity* k1 e k2
  - T2, che consuma le *commodity* k1 e k3
- Sono presenti quattro broker N1, N2, N3 ed N4

Tutti gli archi presenti hanno la stessa capacità pari a 70.

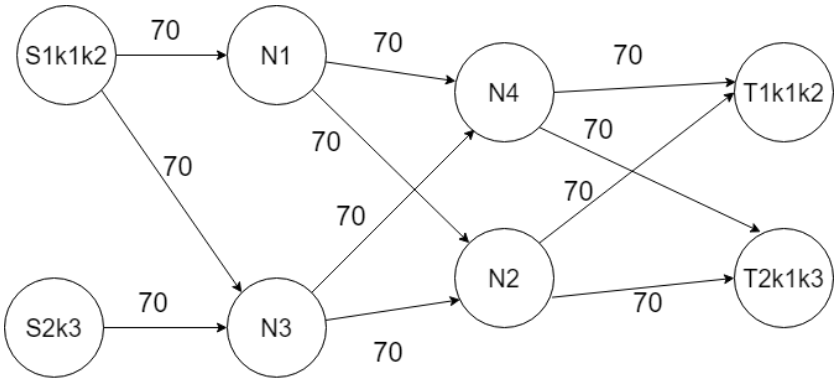

Figura 5.10: Grafo del secondo caso

5.2.2 Risultati Prima formulazione

In figura 5.11 è visibile il risultato dell'ottimizzazione ottenuta utilizzando la prima formulazione, mentre in figura 5.12 è presente la struttura della rete finale a cui sono stati rimossi gli archi inutilizzati. Come si può notare in questo caso per distribuire la *commodity* k1 alle due destinazioni che la richiedevano, la rete risultante sfrutta il percorso $S1 \rightarrow N3 \rightarrow N4$, per poi utilizzare il *broker* N4 per consegnare la *commodity* alle due destinazioni richiedenti.

```
The optimized total flow value is 0, and the edges optimized values are
 The source node is S1, the destination node is N1, the commodity is S1T1k2, the optimized value for the edge is 1
The source node is N1, the destination node is N2, the commodity is S1T1k2, the optimized value for the edge is 1
The source node is N2, the destination node is T1, the commodity is S1T1k2, the optimized value for the edge is 1
The source node is T1, the destination node is T1k2, the commodity is S1T1k2, the optimized value for the edge is 1
The source node is S2, the destination node is N3, the commodity is S2T2k3, the optimized value for the edge is 1
The source node is N3, the destination node is N2, the commodity is S2T2k3, the optimized value for the edge is 1
The source node is N2, the destination node is T2, the commodity is S2T2k3, the optimized value for the edge is 1
The source node is T2, the destination node is T2k3, the commodity is S2T2k3, the optimized value for the edge is 1
The source node is S1, the destination node is N3, the commodity is S1T1k1, the optimized value for the edge is 1
The source node is N3, the destination node is N4, the commodity is S1T1k1, the optimized value for the edge is 1
The source node is T1, the destination node is T1k1, the commodity is S1T1k1, the optimized value for the edge is 1
The source node is N4, the destination node is T1, the commodity is S1T1k1, the optimized value for the edge is 1
The source node is S1, the destination node is N3, the commodity is S1T2k1, the optimized value for the edge is 1
The source node is N3, the destination node is N4, the commodity is S1T2k1, the optimized value for the edge is 1
The source node is T2, the destination node is T2k1, the commodity is S1T2k1, the optimized value for the edge is 1
The source node is N4, the destination node is T2, the commodity is S1T2k1, the optimized value for the edge is 1
```

Figura 5.11: Risultato testuale dell'ottimizzazione del caso d'uso 2 utilizzando la prima formulazione matematica

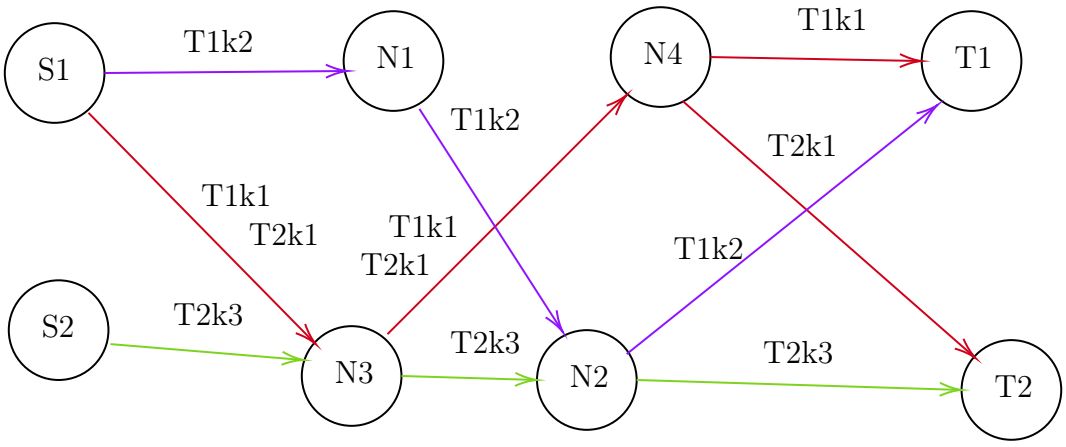

Figura 5.12: Grafo risultato dell'ottimizzazione del caso d'uso 2 utilizzando la prima formulazione matematica

5.2.3 Risultati formulazione lagrangiana

Applicando la formulazione lagrangiana con parametro $\lambda_{i,j} = 5$ a questo problema, il risultato ottenuto è identico a quello ottenuto applicando la prima formulazione: anche in questo caso la formulazione lagrangiana si è dimostrata una buona approssimazione del problema originale.

5.2.4 Risultati Seconda formulazione

In figura 5.13 è visibile il risultato dell'ottimizzazione ottenuta utilizzando la seconda formulazione, mentre in figura 5.14 è presente la struttura della rete finale a cui sono stati rimossi gli archi inutilizzati. Utilizzando questa formulazione la rete ottenuta è anche in questo caso uguale a quella ottenuta con le due precedenti formulazioni.

```
The optimized total flow value is 10, and the edges optimized values are
 The source node is S1, the destination node is N3, the commodity is k1, the optimized value for the edge is 1
The source node is N3, the destination node is N4, the commodity is k1, the optimized value for the edge is 1
The source node is N4, the destination node is T1, the commodity is k1, the optimized value for the edge is 1
The source node is N4, the destination node is T2, the commodity is k1, the optimized value for the edge is 1
The source node is S1, the destination node is N1, the commodity is k2, the optimized value for the edge is 1
The source node is N1, the destination node is N2, the commodity is k2, the optimized value for the edge is 1
The source node is N2, the destination node is T1, the commodity is k2, the optimized value for the edge is 1
The source node is S2, the destination node is N3, the commodity is k3, the optimized value for the edge is 1
The source node is N3, the destination node is N2, the commodity is k3, the optimized value for the edge is 1
The source node is N2, the destination node is T2, the commodity is k3, the optimized value for the edge is 1
```

Figura 5.13: Risultato testuale dell'ottimizzazione del caso d'uso 2 utilizzando la seconda formulazione matematica

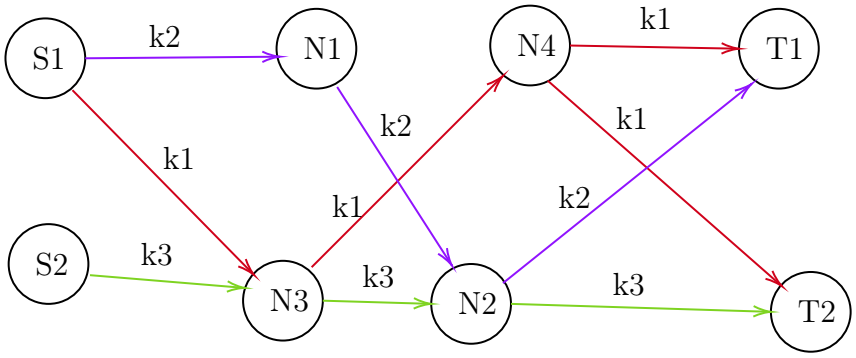

Figura 5.14: Grafo risultato dell'ottimizzazione del caso d'uso 2 utilizzando la seconda formulazione matematica

5.3 Caso d'uso 3

5.3.1 Rete analizzata

Per il terzo caso d'uso è stata presa in considerazione una rete molto più complessa rispetto alle due precedenti. In figura 5.15 è presente una rappresentazione della rete, rete che può essere così descritta:

- *Commodity*: Sono presenti sei diverse *commodity*:
  - k1 con peso 20
  - k2 con peso 20
  - k3 con peso 25
  - k4 con peso 25
  - k5 con peso 15
  - k6 con peso 20
- Sorgenti: sono presenti tre diverse sorgenti:
  - S1, che produce le *commodity* k1 e k2
  - S2, che produce le *commodity* k3 e k4
  - S3, che produce le *commodity* k5 e k6
- Destinazioni: sono presenti tre destinazioni:
  - T1, che consuma le *commodity* k1, k4 e k6
  - T2, che consuma le *commodity* k2, k3 e k4
  - T3, che consuma le *commodity* k4, k5 e k6
- Sono presenti otto broker N1, N2, N3, N4, N5, N6, N7 ed N8

Tutti gli archi presenti hanno la stessa capacità pari a 40.

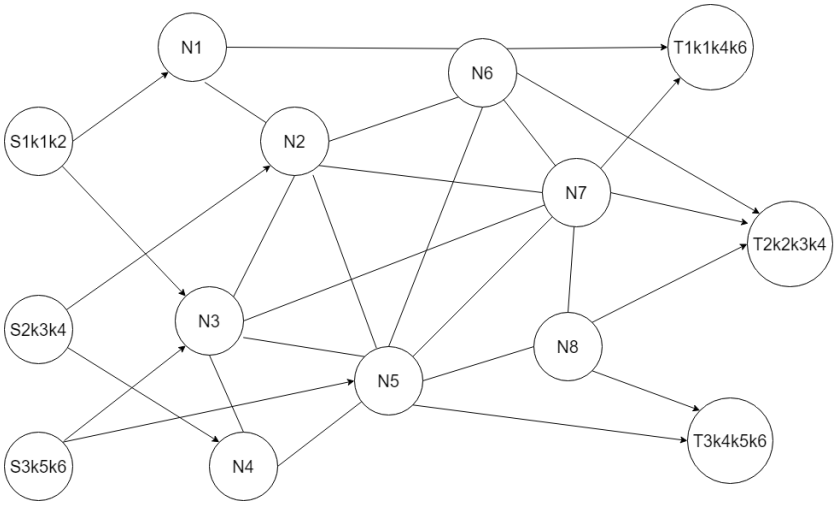

Figura 5.15: Grafo del terzo caso d'uso

5.3.2 Risultati Prima formulazione

In figura 5.16 è visibile il risultato dell'ottimizzazione ottenuta utilizzando la prima formulazione, mentre in figura 5.17 è presente la struttura della rete finale a cui sono stati rimossi gli archi inutilizzati. Anche in questa soluzione la rete ha sfruttato la stessa connessione per distribuire *commodities* uguali, utilizzando i broker laddove necessario per ridirigere le *commodities* verso le rispettive destinazioni

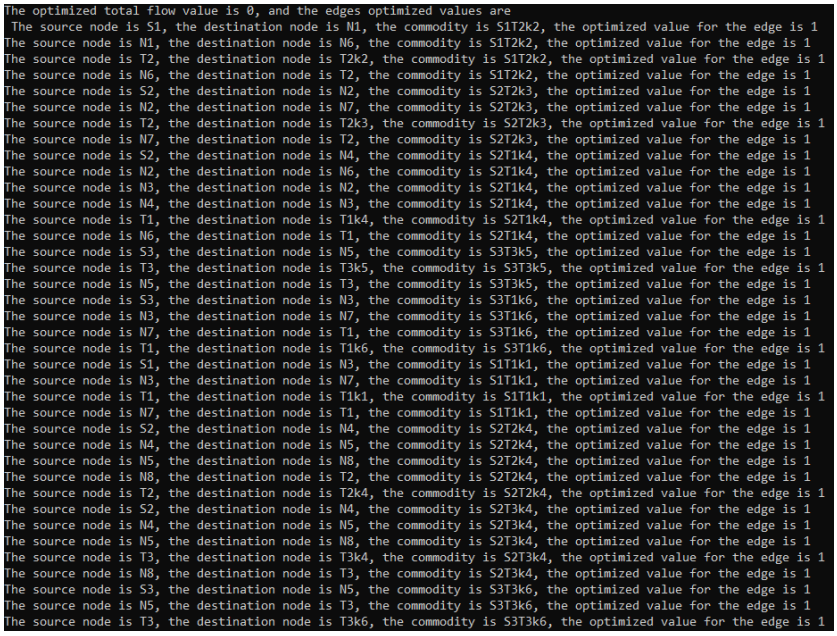

Figura 5.16: Risultato dell'ottimizzazione del caso d'uso 3 utilizzando la prima formulazione matematica

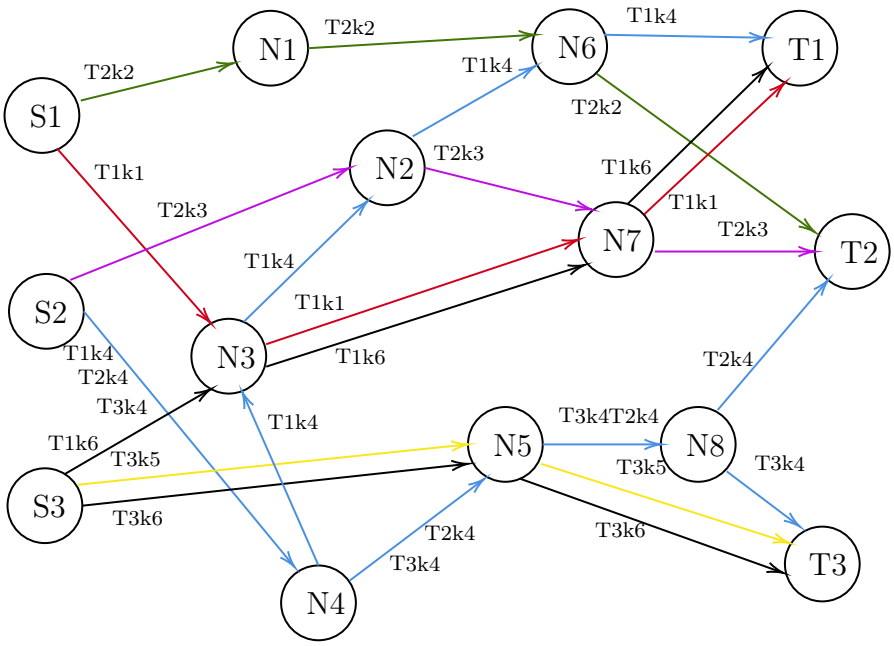

Figura 5.17: Grafo risultato dell'ottimizzazione del caso d'uso 3 utilizzando la prima formulazione matematica

5.3.3 Risultati formulazione lagrangiana

In figura 5.18 è visibile il risultato dell'ottimizzazione ottenuta utilizzando la formulazione lagrangiana, mentre in figura 5.19 è presente la struttura della rete finale a cui sono stati rimossi gli archi inutilizzati. In questo caso applicando un $\lambda_{i,j} = 3$ alla funzione obiettivo, la rete risultante varia in modo notevole rispetto al risultato della prima ottimizzazione, lasciando i nodi $N4$ ed $N8$ inutilizzati. Il risultato è una rete che pur ottimizzando il percorso in questo caso viola i vincoli di capacità di alcuni archi, come ad esempio il collegamento $S2 \rightarrow N2$ o quello $N5 \rightarrow T3$, rimanendo comunque simile alla rete risultante nel primo e caso e dunque un'approssimazione accettabile.

```
The optimized total flow value is 1365, and the edges optimized values are
The source node is S1, the destination node is N1, the commodity is S1T2k2, the optimized value for the edge is 1
The source node is N1, the destination node is N6, the commodity is S1T2k2, the optimized value for the edge is 1
The source node is T2, the destination node is T2k2, the commodity is S1T2k2, the optimized value for the edge is 1
The source node is N6, the destination node is T2, the commodity is S1T2k2, the optimized value for the edge is 1
The source node is S2, the destination node is N2, the commodity is S2T2k3, the optimized value for the edge is 1
The source node is N2, the destination node is N7, the commodity is S2T2k3, the optimized value for the edge is 1
The source node is T2, the destination node is T2k3, the commodity is S2T2k3, the optimized value for the edge is 1
The source node is N7, the destination node is T2, the commodity is S2T2k3, the optimized value for the edge is 1
The source node is S2, the destination node is N2, the commodity is S2T1k4, the optimized value for the edge is 1
The source node is N2, the destination node is N6, the commodity is S2T1k4, the optimized value for the edge is 1
The source node is T1, the destination node is T1k4, the commodity is S2T1k4, the optimized value for the edge is 1
The source node is N6, the destination node is T1, the commodity is S2T1k4, the optimized value for the edge is 1
The source node is S3, the destination node is N5, the commodity is S3T3k5, the optimized value for the edge is 1
The source node is T3, the destination node is T3k5, the commodity is S3T3k5, the optimized value for the edge is 1
The source node is N5, the destination node is T3, the commodity is S3T3k5, the optimized value for the edge is 1
The source node is S3, the destination node is N5, the commodity is S3T1k6, the optimized value for the edge is 1
The source node is N5, the destination node is N7, the commodity is S3T1k6, the optimized value for the edge is 1
The source node is N7, the destination node is T1, the commodity is S3T1k6, the optimized value for the edge is 1
The source node is T1, the destination node is T1k6, the commodity is S3T1k6, the optimized value for the edge is 1
The source node is S1, the destination node is N3, the commodity is S1T1k1, the optimized value for the edge is 1
The source node is N3, the destination node is N7, the commodity is S1T1k1, the optimized value for the edge is 1
The source node is T1, the destination node is T1k1, the commodity is S1T1k1, the optimized value for the edge is 1
The source node is N7, the destination node is T1, the commodity is S1T1k1, the optimized value for the edge is 1
The source node is S2, the destination node is N2, the commodity is S2T2k4, the optimized value for the edge is 1
The source node is N2, the destination node is N6, the commodity is S2T2k4, the optimized value for the edge is 1
The source node is N6, the destination node is T2, the commodity is S2T2k4, the optimized value for the edge is 1
The source node is T2, the destination node is T2k4, the commodity is S2T2k4, the optimized value for the edge is 1
The source node is S2, the destination node is N2, the commodity is S2T3k4, the optimized value for the edge is 1
The source node is N2, the destination node is N6, the commodity is S2T3k4, the optimized value for the edge is 1
The source node is N6, the destination node is N5, the commodity is S2T3k4, the optimized value for the edge is 1
The source node is T3, the destination node is T3k4, the commodity is S2T3k4, the optimized value for the edge is 1
The source node is N5, the destination node is T3, the commodity is S2T3k4, the optimized value for the edge is 1
The source node is S3, the destination node is N5, the commodity is S3T3k6, the optimized value for the edge is 1
The source node is N5, the destination node is T3, the commodity is S3T3k6, the optimized value for the edge is 1
The source node is T3, the destination node is T3k6, the commodity is S3T3k6, the optimized value for the edge is 1
```

Figura 5.18: Risultato dell'ottimizzazione del caso d'uso 3 utilizzando la formulazione lagrangiana

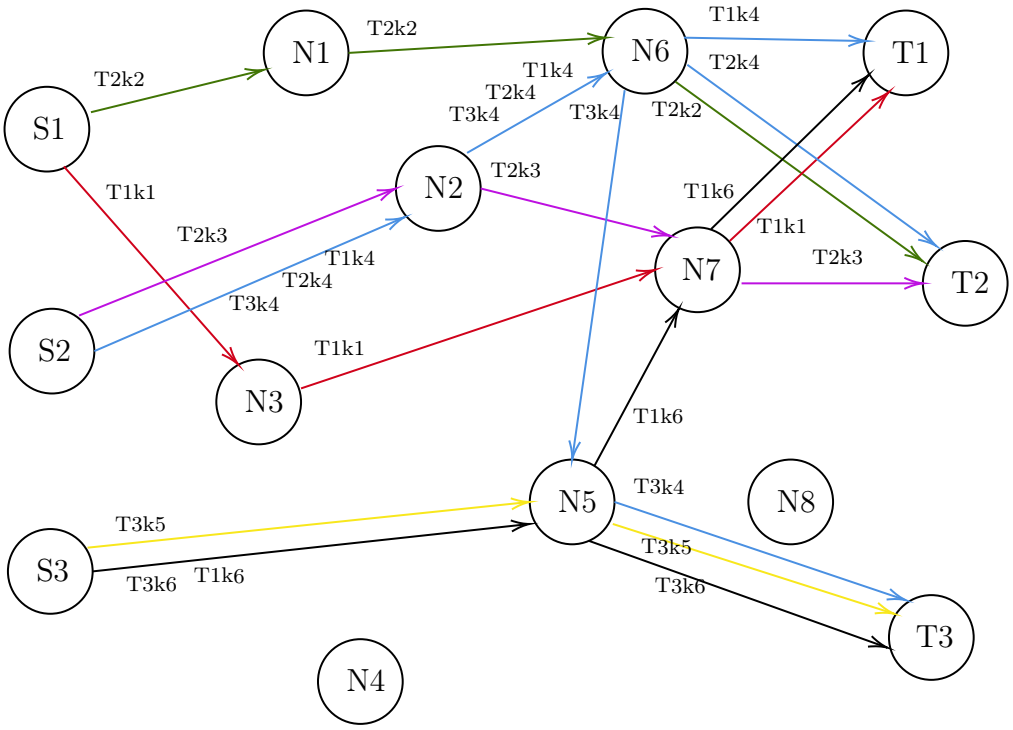

Figura 5.19: Grafo risultato dell'ottimizzazione del caso d'uso 3 utilizzando la formulazione lagrangiana

5.3.4 Risultati Seconda formulazione

In figura 5.20 è visibile il risultato dell'ottimizzazione ottenuta utilizzando la seconda formulazione, mentre in figura 5.21 è presente la struttura della rete finale a cui sono stati rimossi gli archi inutilizzati. Utilizzando questa formulazione la rete ottenuta è molto diversa rispetto alla rete ottenuta applicando la prima formulazione, mantenendo comunque le medesime caratteristiche di distribuzione e riutilizzo di connessioni per dirigere le *commodities* verso le rispettive destinazioni.

```
The optimized total flow value is 22, and the edges optimized values are
 The source node is S1, the destination node is N1, the commodity is k1, the optimized value for the edge is 1
The source node is N1, the destination node is N6, the commodity is k1, the optimized value for the edge is 1
The source node is N6, the destination node is T1, the commodity is k1, the optimized value for the edge is 1
The source node is S1, the destination node is N3, the commodity is k2, the optimized value for the edge is 1
The source node is N3, the destination node is N7, the commodity is k2, the optimized value for the edge is 1
The source node is N7, the destination node is T2, the commodity is k2, the optimized value for the edge is 1
The source node is S2, the destination node is N2, the commodity is k3, the optimized value for the edge is 1
The source node is N2, the destination node is N6, the commodity is k3, the optimized value for the edge is 1
The source node is N6, the destination node is T2, the commodity is k3, the optimized value for the edge is 1
The source node is S2, the destination node is N4, the commodity is k4, the optimized value for the edge is 1
The source node is N4, the destination node is N5, the commodity is k4, the optimized value for the edge is 1
The source node is N5, the destination node is N8, the commodity is k4, the optimized value for the edge is 1
The source node is N8, the destination node is N7, the commodity is k4, the optimized value for the edge is 1
The source node is N7, the destination node is T1, the commodity is k4, the optimized value for the edge is 1
The source node is N8, the destination node is T2, the commodity is k4, the optimized value for the edge is 1
The source node is N8, the destination node is T3, the commodity is k4, the optimized value for the edge is 1
The source node is S3, the destination node is N5, the commodity is k5, the optimized value for the edge is 1
The source node is N5, the destination node is T3, the commodity is k5, the optimized value for the edge is 1
The source node is S3, the destination node is N5, the commodity is k6, the optimized value for the edge is 1
The source node is N5, the destination node is N6, the commodity is k6, the optimized value for the edge is 1
The source node is N5, the destination node is T3, the commodity is k6, the optimized value for the edge is 1
The source node is N6, the destination node is T1, the commodity is k6, the optimized value for the edge is 1
```

Figura 5.20: Risultato dell'ottimizzazione del caso d'uso 3 utilizzando la seconda formulazione matematica

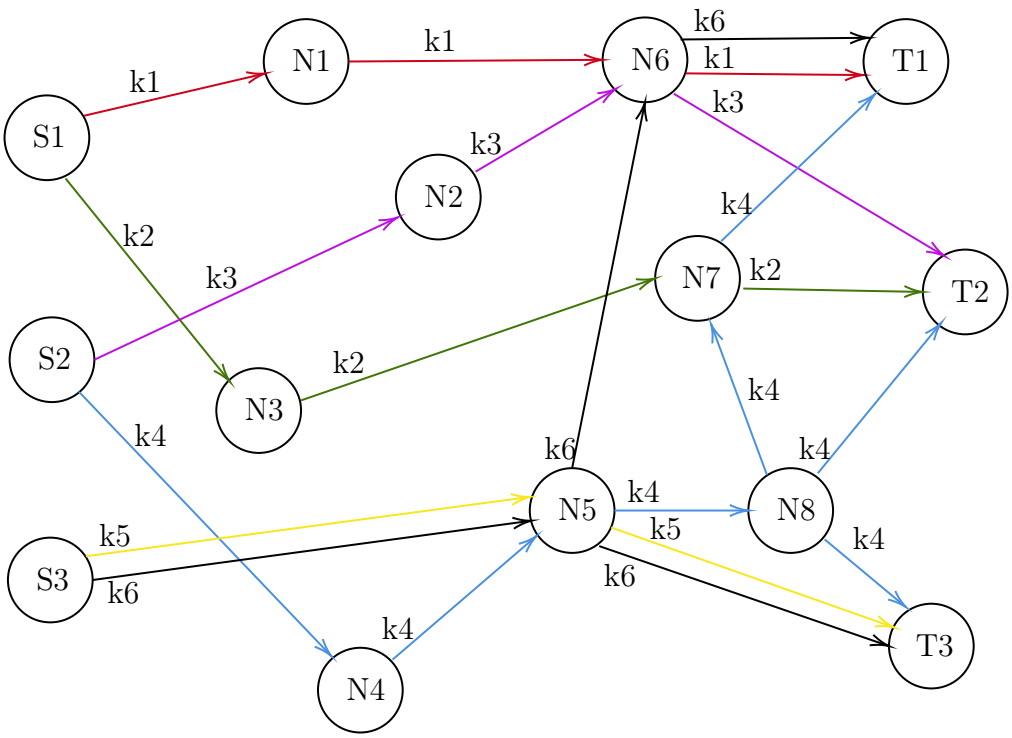

Figura 5.21: Grafo risultato dell'ottimizzazione del caso d'uso 3 utilizzando la seconda formulazione matematica

Conclusioni

Lo sviluppo di questo lavoro ha portato all'applicazione pratica di tecniche di programmazione lineare all'ottimizzazione di reti in ambienti reali e distribuiti. In un contesto in cui l' IoT sta diventando sempre più presente ed utilizzato, avere un modo per ottimizzare le reti di comunicazione utilizzate da questo tipo di sistemi ricopre un ruolo cruciale nella progettazione degli stessi. Inoltre lo studio e l'applicazione di formule di programmazione lineare ha permesso di acquisire una visione diversa sull'utilizzo dell'informatica e della programmazione rispetto a problemi reali, mettendo in evidenza come argomenti e tecniche apparentemente diverse siano molto più legate le une alle altre rispetto a quello che si può pensare.

Un'altra parte fondamentale del lavoro, oltre al sistema implementato, è stata ricoperta dallo sviluppo del sottoprogetto Wrapper per il solver CoinMP, sottoprogetto che è stato molto utile sia per lo studio di tecniche utilizzate per far comunicare stili di programmazione diversi che per acquisire esperienza nello sviluppo di librerie da rilasciare pubblicamente, utilizzando e partecipando a progetti *open-source* come quelli che compongono le librerie COIN-OR.

Lavori Futuri

I possibili sviluppi futuri del lavoro sono orientati al miglioramento del funzionamento del sistema e delle formulazioni matematiche proposte. I miglioramenti più interessanti che da apportare al sistema sono quelli che prevedono una ricerca automatica dei $\lambda_{i,j}$ da assegnare nella formulazione lagrangiana e metodi automatizzati per verificare il risultato ottenuto dall'ottimizzazione.

Gli sviluppi futuri proposti dunque sono:

- Verifica automatizzata del risultato di un'ottimizzazione.
- Ricerca automatica dei valori λ per la formulazione lagrangiana.
- Analisi di sensitività dei valori da assegnare alle variabili costanti nelle varie formulazioni.

Oltre agli sviluppi già citati, un importante sviluppo futuro riguarda l'aggiunta di una formulazione lagrangiana della seconda formulazione proposta, in modo da poterla utilizzare in maniera equivalente alla prima formulazione.

Bibliografia

[1] CBC. https://github.com/coin-or/Cbc.

[2] Codice Gestito. https://docs.microsoft.com/it-it/dotnet/standard/managed-code.

[3] COIN-OR: Computational Infrastructure for Operations Research, 2004. https://https://www.coin-or.org.

[4] COIN-OR repositories. https://github.com/coin-or.

[5] coinbrew. https://github.com/coin-or/coinbrew.

[6] CoinMP. https://github.com/coin-or/CoinMP.

[7] Cplex. https://www.ibm.com/analytics/cplex-optimizer.

[8] GLPK. https://www.gnu.org/software/glpk.

[9] Gurobi. https://www.gurobi.com.

[10] LoRa Alliance. https://lora-alliance.org/.

[11] LoRa and LoRaWAN doc. https://lora-developers.semtech.com/documentation/tech-papers-and-guides/lora-and-lorawan.

[12] LoRa History. https://blog.semtech.com/a-brief-history-of-lora-three-inventors-share-their-personal-story-at-the-things-conference.

[13] MQTT. https://mqtt.org/.

[14] MQTT Version 5.0. Edited by Andrew Banks, Ed Briggs, Ken Borgendale, and Rahul Gupta. 07 March 2019. OASIS Standard. https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html. Latest version: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html.

[15] .NET 5.0. https://docs.microsoft.com/en-us/dotnet/core/dotnet-five.

[16] NuGet. https://www.nuget.org/.

[17] OR-Notes J.E. Beasley. http://people.brunel.ac.uk/~mastjjb/jeb/or/lprelax.html.

[18] Platform Invoke. https://docs.microsoft.com/en-us/dotnet/standard/native-interop/pinvoke.

[19] Specifiche LoRaWAN. https://lora-alliance.org/wp-content/uploads/2020/11/what-is-lorawan.pdf.

[20] R. Baldoni, M. Contenti, and A. Virgillito. *The Evolution of Publish/Subscribe Communication Systems*, pages 137–141. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[21] K. Birman and T. Joseph. Exploiting virtual synchrony in distributed systems. *SIGOPS Oper. Syst. Rev.*, 21(5):123–138, Nov. 1987.

[22] J. Clausen. Branch and bound algorithms-principles and examples. 2003.

[23] G. B. Dantzig and M. N. Thapa. *Linear programming 1: introduction*. Springer Science & Business Media, 2006.

[24] P. Eugster. Type-based publish/subscribe: Concepts and experiences. *ACM Trans. Program. Lang. Syst.*, 29(1):6–es, Jan. 2007.

[25] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, June 2003.

[26] T. S. Ferguson. Linear programming: A concise introduction. *Website. Available at https://www.math.ucla.edu/~tom/LP.pdf*, 2000.

[27] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum-flow problem. *J. ACM*, 35(4):921–940, Oct. 1988.

[28] R. E. Gomory. An algorithm for integer solutions to linear programs. *Recent advances in mathematical programming*, 64(260-302):14, 1963.

[29] M. A. Jaeger, H. Parzyjegla, G. Mühl, and K. Herrmann. Self-organizing broker topologies for publish/subscribe systems. In *Proceedings of the 2007 ACM Symposium on Applied Computing*, SAC '07, page 543–550, New York, NY, USA, 2007. Association for Computing Machinery.

[30] T. Leighton, F. Makedon, S. Plotkin, C. Stein, E. Tardos, and S. Tragoudas. Fast approximation algorithms for multicommodity flow problems. *Journal of Computer and System Sciences*, 50(2):228–243, 1995.

[31] C. Lemaréchal. Lagrangian relaxation. In *Computational combinatorial optimization*, pages 112–156. Springer, 2001.

[32] Y. Liu and B. Plale. Survey of publish subscribe event systems. 06 2003.

[33] B. Mishra and A. Kertesz. The use of mqtt in m2m and iot systems: A survey. *IEEE Access*, 8:201071–201086, 2020.

[34] J. E. Mitchell. Branch-and-cut algorithms for combinatorial optimization problems. *Handbook of applied optimization*, 1:65–77, 2002.

[35] D. R. Morrison, S. H. Jacobson, J. J. Sauppe, and E. C. Sewell. Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, 19:79–102, 2016.

[36] D. Soni and A. Makwana. A survey on mqtt: A protocol of internet of things(iot). 04 2017.

[37] X. Sun and N. Ansari. Traffic load balancing among brokers at the iot application layer. *IEEE Transactions on Network and Service Management*, 15(1):489–502, 2018.

[38] Y. Wang and X. Ma. A general scalable and elastic content-based publish/subscribe service. *IEEE Transactions on Parallel and Distributed Systems*, 26(8):2100–2113, Aug 2015.

[39] M. B. Yassein, M. Q. Shatnawi, S. Aljwarneh, and R. Al-Hatmi. Internet of things: Survey and open issues of mqtt protocol. In *2017 International Conference on Engineering MIS (ICEMIS)*, pages 1–6, 2017.