POLITECNICO DI MILANO

Facoltà di Ingegneria del Sistemi
Dipartimento di Ingegneria Gestionale

Corso di Laurea in
Ingegneria Gestionale

Virtual Factory: Modellazione dei Sistemi di Produzione a Supporto della Valutazione delle Prestazioni

Relatore: Prof. Tullio A. M. TOLIO

Correlatore: Ing. Marcello URGO

Ing. Walter TERKAJ

Tesi di Laurea di:

Liliana Maria G. CAMINITI Matr. 770111

Anno Accademico 2012 – 2013

Alle donne della mia famiglia e a mio padre.

Ringraziamenti

Vorrei esprimere la mia gratitudine al Professore Tullio Tolio prima di tutto per la passione che trasmette a noi studenti in classe. I suoi insegnamenti mi hanno permesso di capire quale sia la strada che voglio intraprendere dalla fine del mio percorso di studi in avanti. E poi, vorrei ringraziarlo per aver creduto in me e per avermi dato la possibilità di lavorare ad una tesi che riguardasse tematiche così innovative e interessanti.

Desidero ringraziare sinceramente l'Ing. Marcello Urgo per il suo continuo supporto e, soprattutto, per la sua pazienza. Per aver creduto, fino in fondo, al valore di questo lavoro di tesi.

Vorrei, inoltre, ringraziare l'Ing. Walter Terkaj, soprattutto per il suo sostegno nella fase iniziale del lavoro.

Indice

Elenco delle figure

Elenco delle tabelle

Sommario

Le imprese a vocazione produttiva si trovano a dover aumentare il livello di flessibilità e rispondere rapidamente alle esigenze del mercato. Devono mantenere alto il livello di produttività e di qualità attraverso il miglioramento dei processi e l'integrazione delle diverse attività di fabbrica, ossia configurare in modo efficace ed efficiente il sistema di produzione nella sua interezza, in qualsiasi fase del ciclo di vita dell'impianto. Per progettare i sistemi produttivi in questo modo, bisogna tenere conto contemporaneamente di differenti aspetti che si influenzano tra loro. Tenere in considerazione tutti gli aspetti implica poter valutare le differenti alternative a disposizione e scegliere quelle migliori. A supporto di tali decisioni esiste il "Virtual Factory Framework", un progetto Europeo che concerne la definizione di modello di fabbrica virtuale. Si tratta di un ambiente virtuale integrato per consentire l'interoperabilità tra diversi strumenti software a supporto dei processi produttivi. Questo lavoro affronta il problema della valutazione delle prestazioni attraverso l'utilizzo del modello di dati VFF chiamato "Virtual Factory Data Model". Il VFDM può essere considerato come un meta-linguaggio condiviso che fornisce una definizione comune dei dati che sono condivisibili attraverso differenti software connessi al framework. In particolare, questa tesi si concentra su uno specifico dominio: il trasporto dei prodotti in lavorazione tra le diverse fasi del processo produttivo. E' stata, perciò, sviluppata un'estensione del modello VFDM in relazione a questo problema. Infine, è stato definito un livello ulteriore di formalizzazione, che serva come base per l'utilizzo di specifici metodi per la valutazione delle prestazioni. L'approccio proposto in questo lavoro di tesi è stato validato sulla base di due test case, uno realistico riferito a un impianto FMS e uno reale riferito ad un impianto per la rettifica di cilindri di laminazione, caso industriale fornito da Tenova Pomini, S.p.A.

Parole chiave: Virtual Factory Framework; Virtual Factory Data Model; Trasporti interni; Valutazione delle prestazioni; Sistemi Produttivi

Abstract

Companies are required to increase their level of flexibility and respond to market quickly, improving the level of productivity and quality through process improvement and integration into the various activities of the factory.This entail the need of software tools supporting product engineering and manufacturing during the various stages of product and factory lifecycles. The European project VFF "Virtual Factory Framework" supports the deployment of a virtual factory model, which is an integrated virtual environment used to enable the interoperability between software tools supporting the factory processes. This work addresses the performance evaluation problem through the application of the VFF data model for virtual factories, called "Virtual Factory Data Model". VFDM can be considered as the shared meta-language providing a common definition of the data that are shared among the software tools connected to the framework. In particular, this thesis focuses on a specific domain: the products transportation through the different steps of the manufacturing process. So, a new extension incident to this problem was developed for the VFDM. Finally a new level of formalisation was defined to use specific performance evaluation methods. The approach proposed in this thesis was validated through two "test case": the realistic one referring to a Flexible Manufacturing System and the specific industrial case proposed by Tenova Pomini S.p.a., concernig a grinding system for rolling mills.

Key words: Virtual Factory Framework; Virtual Factory Data Model; Material Handling; Performance Evaluation; Manufacturing Process .

Introduzione

Oggi le imprese a vocazione produttiva, devono aumentare il livello di flessibilità e rispondere rapidamente alle esigenze del mercato. Devono, quindi, mantenere alto il livello di produttività e di qualità attraverso il miglioramento dei processi e l'integrazione delle diverse attività di fabbrica, ossia configurare in modo efficace ed efficiente il sistema di produzione nella sua interezza, in qualsiasi fase del ciclo di vita dell'impianto.

Per progettare i sistemi produttivi in questo modo, bisogna tenere conto contemporaneamente di differenti aspetti che si influenzano tra loro, come il numero e il tipo di risorse di produzione necessarie, le modalità di trasporto che possano servire tali risorse, l'implementazione dei processi, il layout di sistema in generale, etc. Tenere in considerazione questi diversi aspetti implica poter valutare le differenti alternative a disposizione e scegliere quelle migliori rispetto alle esigenze di partenza.

A supporto di tali decisioni esistono differenti strumenti che implementano diverse tecniche, ognuna specifica per il proprio ambito di riferimento. Quindi bisogna utilizzarli congiuntamente per considerare tutti gli aspetti che entrano in gioco nella progettazione di un sistema produttivo.

Oggi molti sforzi di ricerca vengono orientati verso la *Digital Factory*, ossia un ambiente virtuale che rappresenta le molteplici caratteristiche di un sistema di produzione. Si tratta di una famiglia di strumenti che facilita la condivisione delle risorse, delle informazioni e della conoscenza inerente alla produzione e diventa un vero e proprio supporto alla collaborazione

tra i vari dipartimenti coinvolti nella progettazione, pianificazione, produzione e gestione di un impianto produttivo. Quindi, grazie agli strumenti *Digital Factory* è possibile usufruire della rappresentazione digitale di tutte le informazioni necessarie alla configurazione del sistema per poterne valutare le prestazioni.

Rappresentare la realtà in un contesto virtuale, infatti, consente di effettuare controlli o verifiche sui sistemi produttivi (virtuali) con l'obiettivo di trovare le migliori soluzioni (da applicare alla realtà) [1].

Una proposta concreta, strutturata ed estremamente innovativa di Digital Factory, è il Virtual Factory Framework (VFF), ossia un progetto di ricerca europeo (7FP), appena conclusosi, coordinato dall'ITIA-CNR in collaborazione con 27 partner, tra cui alcune tra le più importanti aziende e istituti nel panorama industriale europeo.

Il VFF può essere definito come:

> *"Un ambiente virtuale integrato e collaborativo volto a semplificare la condivisione di risorse di informazioni e di conoscenza in ambito produttivo, sostenendo, al contempo, la progettazione e la gestione di tutte le entità di fabbrica, da un unico prodotto alle reti di imprese, lungo tutte le fasi del ciclo di vita."* [2]

Si tratta di modello virtuale di fabbrica che si basa su una formalizzazione univoca della conoscenza, tale da essere condivisa da tipi di strumenti differenti per poter affrontare ogni tipo di analisi necessaria alla progettazione e configurazione dei sistemi produttivi. All'interno del progetto VFF è stato sviluppato un framework generale per la rappresentazione della conoscenza, il Virtual Factory Data Model (VFDM), che stabilisce un modello di dati coerente, standard, estensibile e comune per la rappresentazione degli oggetti di fabbrica relativi ai sistemi di produzione, risorse, processi e prodotti, e che si basa sull'utilizzo delle ontologie. Tale modello è condivisibile e standardizza-

to, perciò permette a più strumenti di utilizzare le informazioni così formalizzate. In questo modo, infatti, gli utenti o i metodi di configurazione, interfacciandosi direttamente con il VFF, potranno avere un quadro di riferimento più ampio. Infatti, potranno interrogare la conoscenza utilizzando più strumenti in grado di fornire le informazioni desiderate.

Con questa tesi si vuole dimostrare che l'utilizzo del VFF è una via concretamente percorribile e per fare ciò, si è voluto focalizzare l'attenzione su uno specifico sottoproblema, ossia il trasporto interno relazionato ai differenti *process step* del processo produttivo.

La modellazione dei trasporti interni in termini di formalizzazione della conoscenza, poi, è un problema noto che tipicamente viene semplificato riducendo le caratteristiche di interesse al solo tempo di trasporto. In realtà, la complessità di gestione dei trasporti interni è legata al numero elevato di vincoli e risorse che entrano in gioco. Inoltre, essi influenzano fortemente i costi di produzione, la sicurezza per gli operatori, la produttività, le scelte di layout del sistema produttivo e la scelta degli impianti. Inoltre, il VFF, e nello specifico il VFDM, non presenta una formalizzazione della conoscenza inerente ai trasporti interni completa di tutte le caratteristiche tipiche di questo tema. Per questi motivi, si è scelto di incentrare l'attenzione su questa tematica, e in particolare si sono definiti due step principali sui quali incentrare il lavoro di tesi:

I Formalizzazione della conoscenza relativa alle problematiche di trasporto all'interno di un impianto

II Utilizzo di tale conoscenza per poter rispondere a domande inerenti al trasporto interno che possono sorgere sia in fase di configurazione sia in fase di gestione di un sistema produttivo.

L'elaborato si sviluppa in cinque capitoli. Il primo ha l'obiettivo di fornire informazioni utili sugli argomenti trattati e sugli strumenti utilizzati, facendo riferimento alla letteratura relativa disponibile. In particolare si vogliono anticipare al lettore le terminologie e i concetti che sono utilizzati e richiamati lungo tutto il progetto di tesi.

Il Capitolo 2 descrive il modello per la formalizzazione dei trasporti interni, ottenuto facendo riferimento a modelli già esistenti nel progetto VFDM. Lo scopo finale è quello di definire e consolidare la conoscenza relativa ai trasporti interni all'impianto produttivo attraverso una formalizzazione che prende ispirazione da tale progetto.

Nel Capitolo 3, viene presentato uno dei possibili strumenti a supporto della *manipolazione* di tale conoscenza, che in particolare, è in grado di rispondere a una selezione di domande specifiche relative ai trasporti interni.

Nel Capitolo 4 viene descritto il primo caso applicativo. Un caso realistico e relativo a un sistema manifatturiero FMS, con il quale si intende validare il tool scelto.

L'utlimo Capitolo, il 5, descrive il secondo caso applicativo: il caso reale dell'azienda **Tenova Pomini**, leader a livello mondiale nella progettazione e fornitura di rettifiche per cilindri di laminatoi. L'azienda, inoltre, offre ai clienti anche impianti con la formula "chiavi in mano" completi di tutta la parte dell'impiantistica che comprende anche i trasportatori.

Capitolo 1

Stato dell'arte

Come è stato detto nel capitolo introduttivo, per poter adempiere agli obiettivi che questa tesi si pone, è necessario utilizzare strumenti, progetti, applicativi e teorie specifiche.

Per questo motivo, con il seguente capitolo, si vogliono introdurre al lettore i concetti base cui fa riferimento il lavoro di tesi.

1.1 Il Virtual Factory Framework e la Struttura dei Conoscenza nel Virtual Factory Data Model

Una proposta concreta, strutturata ed estremamente innovativa di Digital Factory, è il Virtual Factory Framework (VFF), ossia un progetto di ricerca europeo (7FP), appena conclusosi, coordinato dall'ITIA-CNR in collaborazione con 27 partner (elencati in Figura 1.1) tra le più importanti aziende e istituti del panorama industriale europeo.

1.1.1 Virtual Factory Framework

Progettare il modello di fabbrica è un compito complesso che richiede strumenti di supporto per affrontare efficacemente tutte le fasi del ciclo di vita del sistema produttivo. Come il VFF, esistono altri strumenti comparabili che implementano la struttura per un ambiente virtuale collaborativo, rappresentando va-

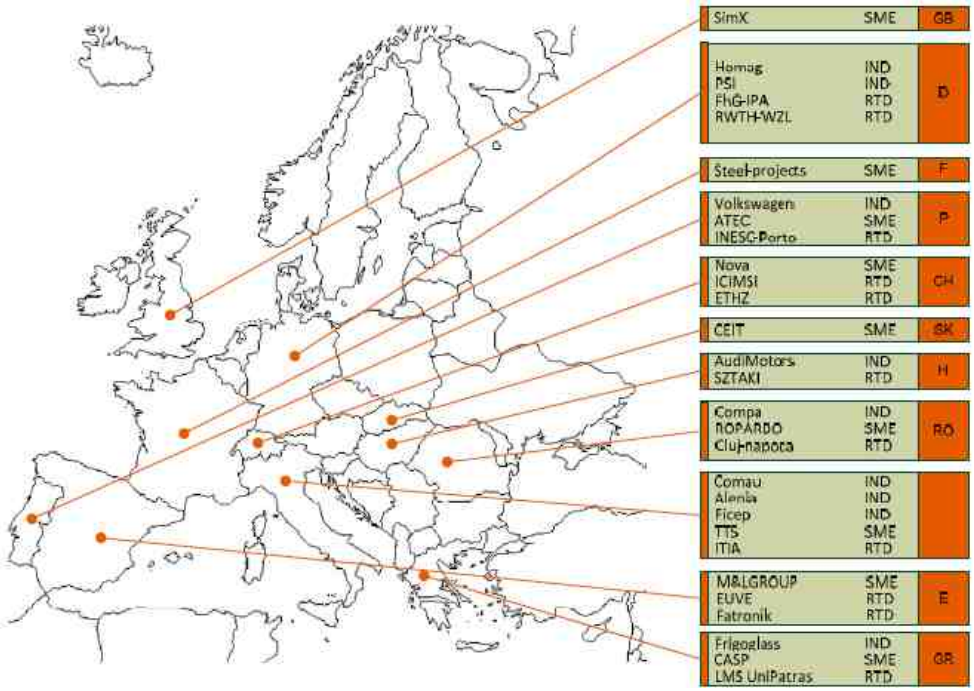


Figura 1.1: Istituti e Imprese coinvolte nel Progetto Europeo (7FP) Virtual Factory Framework. [3]

rie attività di fabbrica atte a facilitare la condivisione di risorse, di informazioni produttive e della conoscenza. Ad esempio lo standard tecnico ANSI/ISA-95 [8] che riguarda lo sviluppo di un interfaccia automatizzata tra impresa e sistemi di controllo. Usato per tutti i tipi di settore e di processo (continui o ripetitivi). B2MML, che sta per Business To Manufacturing Markup Language [9], è un'applicazione XML dello standard ANSI/ISA-95 e consiste in un insieme di schemi XML [10] che implementano i modelli di dati nello standard ISA-95.

Secondo lo standard ANSI/ISA-95, un processo di fabbricazione può essere modellato utilizzando una classe *ProcessSegment*, e questa può rappresentare una singola fase di un processo di fabbricazione, oppure l'intero processo attraverso la composizione. La classe è legata ad altre classi che servono a caratterizzare il processo (ad esempio l'attrezzatura necessaria, gli operatori, il materiale. Inoltre, i rapporti di precedenza tra le

diverse fasi di processo possono essere definiti grazie alla classe *ProcessSegmentDependency*. La classe *EquipementSegmentSpecification* consente all'utente di specificare i componenti di attrezzaggio necessari per l'esecuzione di una fase di processo e come utilizzarla. Lo standard in analisi consente all'utente di definire liberamente le proprietà personalizzate che possono essere collegati alla maggior parte delle classi che rappresentano processi e risorse di produzione.

Tuttavia, tale flessibilità, può presentare un inconveniente dal punto di vista dell'interoperabilità, in quanto se due utenti usano lo stesso standard, per potersi scambiare i dati caratterizzati da semantica adeguata, dovranno mettersi d'accordo sulla definizione delle proprietà degli oggetti. Un ulteriore limitazione riguarda il fatto che l'ANSI/ISA-95 non fornisce un supporto completo per la modellazione dei dati fisici quali il collocamento e la rappresentazione della forma di oggetti appartenenti al sistema produttivo (ad esempio una macchina utensile).

Esistono poi, strumenti ICT rivolti alla definizione del modello di fabbrica e che si basano sull'utilizzo di suite Product Lifecycle Management (PLM) onnicomprensive e sempre più sofisticate. In particolare è possibile fare riferimento a PROMISE, un progetto a livello europeo dedicato all'*IMS* (Intelligence Manufacturing Systems), che si basa, appunto, sull'utilizzo di suite PLM. Questo progetto risponde all'esigenza di monitorare lo stato e controllare le condizioni dei prodotti (finiti o componentistica) e del loro ambiente durante il ciclo di vita. Per poter fare ciò, anche PROMISE, come VFF mette l'accento sulle informazioni e sulla loro gestione, al fine di progettare meglio il prodotto, oppure per fare manutenzione predittiva. Gli step di PROMISE sono: (1) RACCOLTA DATI (parametri tecnici adottati, eventi significativi, condizioni ambientali, storia stati prodotto) I valori numerici e categorici non vengono analizzati

subito, per evitare che ci siano errori in questa fase, è necessario costruire strumenti che identifichino gli "outlier", i dati non corretti etc., per avere affidabilità e consistenza dei dati; (2) ANALISI E TRASFORMAZIONE (know how sul prodotto per segmentare la fascia mercato, profili particolari, causa di guasto, predire vita utile residua) (3) DECISION PROCESSES (prendere decisioni sulla base della conoscenza creata)

Il progetto è diviso in tre fasi:

- BOL (Design-Produce-Supply)
- MOL (Service-Manutenzione)
- EOL (modelli di recupero materiale)

E' possibile asserire che PROMISE (attraverso una gestione delle informazioni con approccio PLM) fornisce benefici relativamente a differenti aspetti come[4]:

1 Performance finanziarie (Aumento fatturato, Aumento mercato entrandoci prima, Aumento ciclo vita prodotti e Diminuzione costi di richiamata)

2 Riduzione dei tempi (Tempo progetti, Tempo modifica e TTM)

3 Qualità (Migliore qualità lavorazioni, Riduzione scarti)

4 Business (Aumento riutilizzo parti e aumento tasso introduzione nuovi prodotti, migliore tracciabilità prodotti è fead importante es. temperatura)

La limitazione di questo progetto, però risiede nei costi di acquisto, di implementazione e manutenzione dei più avanzati software PLM. Infatti, questi costi non sono sostenibili per le piccole e medie imprese. Inoltre, la maggior parte di questi strumenti, seppur fortemente orientati al Collaborative Product

Development and Management (CPDM), non sono ancora completamente in grado di fornire tutte le funzionalità richieste soprattutto a causa della mancanza di interoperabilità tra alcuni importanti software che li compongono [1]. Molte soluzioni di questo genere, infatti, sono fortemente focalizzate sui *dettagli* meccanici trascurando il punto di vista del sistema di produzione.

L'approccio VFF, dal canto suo, ha l'intento di apportare importanti progressi agli strumenti ad esso precedenti.

Tra queste migliorie si possono citare la definizione di un quadro di riferimento per le attività di pianificazione dei processi produttivi; lo sviluppo di un modello di dati di fabbrica comune ed estensibile che riguarda prodotti, processi e risorse per sopperire alla limitata interoperabilità tra le differenti piattaforme software che usano i *formati proprietari*. Il Progetto VFF, inoltre, dovrebbe consentire un maggiore risparmio sui tempi e costi, accrescendo, contemporaneamente, le performance nella progettazione, nella gestione, nella valutazione e riconfigurazione di attrezzature nuove o esistenti. Inoltre migliora in maniera tangibile la capacità di simulare i comportamenti dinamici e complessi lungo il ciclo di vita della fabbrica[2].

E' per via di queste migliorie che si è scelto tale progetto, ma anche perché essendosi concluso pochi mesi fa risulta interessante valutarne le potenzialità.

Come è possibile vedere in Figura 1.2, l'architettura VFF si basa su tre pilastri principali [1]:

i **Virtura Factory Data Model semantico:** stabilisce un modello di dati coerente, standard, estensibile e comune per la rappresentazione degli oggetti di fabbrica relativi ai sistemi di produzione, risorse, processi e prodotti. Il modello comune di dati può essere considerato come un *meta-linguaggio*

condiviso che fornisce una definizione comune dei dati e della conoscenza memorizzata in un *repository*.

ii **Virtual Factory Manager semantico:** governa il *repository* in cui sono memorizzati dati e conoscenza [6] [5]. Il VFM, quindi, governa la conoscenza condivisa ed è la parte *core* del VFF e gestisce i moduli disaccoppiati della Virtual Factory fornendo un accesso controllato ai diversi progetti di VF.

iii **Moduli Virtual Factory separati:** strumenti software che implementano i vari metodi e servizi a supporto delle attività relative alla progettazione di fabbrica, alla valutazione delle prestazioni, alla gestione, al controllo della produzione, etc. Si tratta di applicativi commerciali o non commerciali inseriti in una workstation remota o sul server in cui risiede il Virtual Factory Manager. Integrando i moduli VF, dotati di diverse funzionalità e livelli di dettaglio, e mantenendo la stessa rappresentazione di fabbrica, consentirebbe di avere la possibilità di raggiungere una vasta gamma di utenti.

I pilastri del VFF sono stati concepiti per ottenere uno strumento aperto, scalabile e semplice grazie all'inserimento dei moduli VF disaccoppiati, riducendo, così, i *costi di investimento* rispetto a software *"all-in-one"*.

1.1.2 Virtual Factory Data Model

Il lavoro di tesi si sviluppa a partire dalla formalizzazione della conoscenza basata sul Virtual Factory Data Model, per questo motivo è bene spiegare in modo più approfondito come si sviluppa questo *pilastro* del VFF.

Il VFDM mira a formalizzare e integrare i concetti di "building", prodotto, processo e risorse produttive gestite da strumenti digitali che supportano le fasi del ciclo di vita della fab-

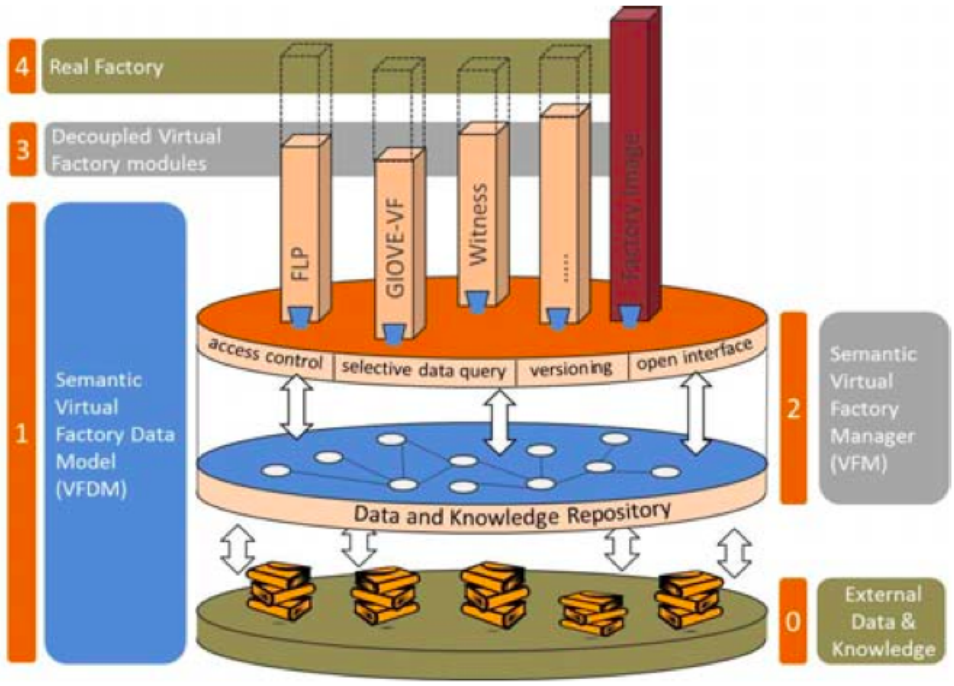


Figura 1.2: Architettura del Virtual Factory Framework [1].

brica. Questi strumenti, di solito, hanno a che fare con dati input/output che riguardano:

- *Building:* riguarda la struttura fisica della fabbrica (muri, colonne, etc.).
- *Prodotto:* vale a dire il prodotto visto come l'output di produzione dalla fabbrica.
- *Processo:* ossia i processi che vengono eseguiti da un sistema per trasformare direttamente o indirettamente un prodotto.
- *Risorse Produttive:* si tratta di risorse che sono usate dal sistema con l'obiettivo finale di trasformare il prodotto(o un semilavorato). Queste risorse possono essere operatori, macchine, mezzi di trasporto etc.

- *Sistemi Produttivi:* ossia i sistemi di trasformazione (sistemi di produzione, assemblaggio o disassemblaggio) che modificano il prodotto attraverso l'utilizzo di risorse fisiche e/o risorse umane in un processo.

- *Fabbrica:* vale a dire la fabbrica vista nel suo insieme durante il suo ciclo di vita.

Il VFDM è un modello ontologico che sfrutta gli standard tecnici per normare l'assegnamento dei nomi agli attributi. Sono numerosi i contributi scientifici e le proposte di standard che affrontano il tema della formalizzazione della conoscenza inerente ai processi produttivi, è possibile citare tra questi lo standard Industry Foundation classes (IFC) della buildingSMART [11] che è parzialmente basato sullo standard STEP [12], oppure ANSI/ISA-95 (cfr. paragrafo 1.1.1), oppure ancora, il Process Specification Language (PSL).

In particolare si tratta di strumenti che pongono l'accento sul problema dello sviluppo di un modello di dati olistico e completo per la rappresentazione dei sistemi di produzione, considerando sia gli aspetti tangibili, ossia le macchine utensili, i tipi di parti da produrre, etc., sia quelli intangibili, come i piani di processo e le logiche produttive [7].

La particolarità del VFDM è il fatto che questo sfrutti diversi standard già esistenti, come quelli elencati precedentemente, cercando di favorire l'interoperabilità tra strumenti software.

Il VFDM adotta in particolare le tecnologie del Semantic Web per lo sviluppo del VFDM come ontologia [5] che offrono la possibilità al VFF di rappresentare una semantica formale, un modello efficiente, una gestione distribuita dei dati, una interoperabilità delle diverse applicazioni. In questo modo permettono di sfruttare strumenti generici che possano estrapolare informazioni dall'ontologia ed effettuare ragionamenti su di essa, fornendo

così un supporto generico che non è personalizzato sul dominio specifico.

Confrontando tecnologie simili, il Semantic Web è, ad esempio, preferito alla tecnologia XSD perché questa non consente di effettuare una caratterizzazione esplicita dei dati con i loro rapporti a livello semantico [14]. I riferimenti, poi, non sono modellati opportunamente in modo tale da avere una coerenza referenziale. Pertanto i dati distribuiti possono essere difficilmente gestiti e l'integrazione dei diversi domini di conoscenza può risultare complessa [15].

Le caratteristiche principali del VFDM basato sulle ontologie possono essere schematizzate come segue [7]:

- Il vocabolario OWL 2.0 [15] del *Web Ontology Language* [16] è stato adottato per definire tutte le classi, le proprietà e le restrizioni che possono essere utilizzate per creare gli *individuals* da memorizzare nel repository del VFDM.

- Dopo l'analisi dello stato dell'arte, sono state selezionate le classi IFC Industry Foundation Classes [11] e le STEP-NC [12] per essere integrate nel VFDM al fine di modellare la fabbrica. Rispettivamente per le classi più astratte degli oggetti le prime e per modellare il processo produttivo le seconde.

- Gli standard selezionati sono stati parzialmente tradotti in ontologie. In effetti, l'implementazione ufficiale di IFC e STEP-NC come ontologie non esiste ancora.

- Gli standard tecnici sono stati estesi in modo tale da giungere alla creazione di nuove strutture dati.

1.1.3 L'Architettura del Virtual Factory Data Model

I differenti domini di conoscenza del VFDM sono organizzati in macro-aree costituite da una o più ontologie, creando così una struttura gerarchica di ontologie per scomporre il problema riducendone la complessità. Il VFDM definisce solo i cosiddetti meta-dati, ossia classi, proprietà e restrizioni, mentre le istanze reali (gli *individuals* del OWL) sono memorizzate nel repository di dati. La Tabella 1.1 elenca le ontologie contenute in ogni macro area del VFDM, la cui architettura è rappresentata in Figura 1.3. Il VFDM è composta da 593 classi, 508 proprietà degli oggetti e una proprietà dei dati. Le regole principali seguite in sede di creazione del VFDM sono:

- Le ontologie che contengono solo le definizioni di nuove classi hanno "VFF" come prefisso del nome (per esempio *VffCommons*), mentre le ontologie create dall'importazione/trasformazione di ontologie di terze parti o di standard tecnici hanno il prefisso uguale all'acronimo della sorgente (ad esempio "Ifc per lo standard IFC, "StepNc" per lo standard STEP-NC).

- Le nuove classi sono denominate con il prefisso "VFF" mentre le classi che sono importate da uno standard tecnico hanno come prefisso lo standard stesso.

Infine, dato il gran numero di ontologie importate, analisi specifiche sono state rivolte a tale problematica al fine di evitare la generazione di ridondanze e incongruenze tra le classi derivate da standard diversi. Se la ridondanza non può essere evitata, allora sono state definite le classi equivalenti oppure sono state stabilite relazioni gerarchiche.

VFDM- Macro Area	Ontologie
Commons	VffCommons; IfcActorResource, IfcCostResource, IfcDateTimeResource, IfcUtilityResource, IfcRepresentationResource, IfcExternalReferenceResource, IfcTopologyResource, IfcGeometricConstraintResource, IfcGeometricModelResource, IfcGeometryResource, IfcMaterialResource, IfcMeasureResource, IfcPropertyResource, IfcQuantityResource; IfcKernel, IfcProductExtension, IfcProcessExtension, IfcControlExtension; VffStochasticResource, VffFailureResource
Building	IfcSharedBldgElements
Prodotto	StepNcAP10, VffProduct
Processo	VffProcess
Risorse produttive	VffProductionResource
Sistema Produttivo	VffSystem
Fabbrica	VffFactory

Tabella 1.1: Le Macro Aree nel Virtual Factory Data Model [1]

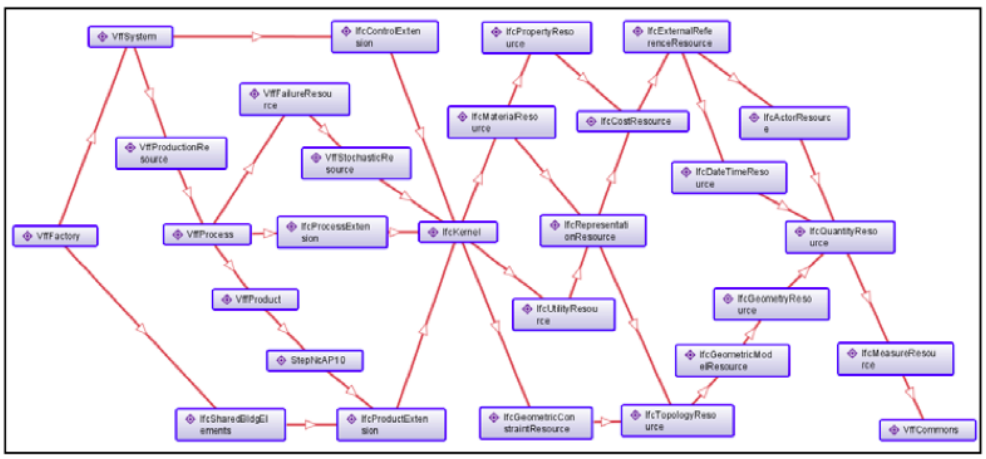


Figura 1.3: Architettura del Virtual Factory Data Model.

1.1.4 VFDM: Modellazione del sistema produttivo

Come è stato precedentemente scritto, il VFDM si basa principalmente sullo standard IFC, che è stato tradotto in una serie di ontologie adottando l'approccio Semantic Web. Le *Entità* dello

standard IFC vengono mappate nelle Classi OWL nel VFDM. La maggior parte delle classi derivate da IFC sono specializzazioni di due classi fondamentali denominate *IfcTypeObject* e *IfcObject*. La prima è la generalizzazione di qualsiasi cosa o processo visti come *tipo*, nel caso della seconda, invece, sono visti come *occurrence*. Gli individui OWL della classe *IfcObject* possono essere connessi con un individuo di classe *IfcTypeObject*.

Le sottoclassi della classe *IfcTypeObject* sono *IfcTypeProduct*, *IfcTypeProcess* e *IfcTypeResource*. La sottoclasse *IfcTypeProduct* rappresenta un tipo oggetto generico che può essere correlato a un contesto geometrico o spaziale (ad esempio, i prodotti, le macchine utensili, i sistemi di trasporto, etc.); *IfcTypeProcess* definisce un tipo generico di processo che può essere utilizzato per trasformare un input in output (ad esempio attività di assemblaggio, lavorazione, etc.); *IfcTypeResource* rappresenta le informazioni relative ai tipi di risorse necessarie per eseguire un processo. Una risorsa rappresenta l' "uso delle cose". *IfcObject* ha le tre sottoclassi principali: *IfcProduct*, *IfcProcess*, *IfcResource* che rappresentano un evento del tipo corrispondente modellato dalle sottoclassi *IfcTypeObject*.

Le classi generiche appena descritte possono essere sfruttate per progettare una vasta gamma di sistemi di produzione, tenendo in considerazione gli aspetti sia fisici sia logici. Le sottoclassi di *IfcTypeObject* possono essere utilizzate per specificare le caratteristiche di progetto di un sistema produttivo, si pensi, ad esempio, ai part-type da produrre (come individui di *IfcTypeProduct*), alle pianificazioni dei processi (come individui di *IfcTypeProcess*), al tipo di risorse produttive richieste (come individui di *IfcTypeResource*).

D'altra parte, le sottoclassi di *IfcObject* possono essere utilizzate per rappresentare la fase di esecuzione di un sistema produttivo definendo i pezzi in lavorazione (come individui di *IfcProduct*), le operazioni effettivamente eseguite (come individui

di IfcProcess), e l'uso di risorse produttive (come individui di *IfcResource*).
Durante le fasi di pianificazione e progettazione del sistema produttivo, i tipi-risorsa che necessitano di un tipo-processo possono essere specificati per mezzo della classe *IfcRelAssignsToProcess*.

Durante la fase di esecuzione del sistema produttivo (sia reale sia simulato), le occorrenze del processo e le risorse possono essere create facendo riferimento a specifici tipi definiti durante la fase di progettazione grazie alla classe *IfcRelDefinesByType*.

Come dettagliatamente descritto da Terkay et al. [1], il VFDM specializza alcune classi dello standard IFC per l'ambito produttivo, prestando attenzione in particolare a classi di tipo *IfcTypeProduct*,*IfcTypeProcess*, *IfcTypeResource* e le corrispondenti classi *occurrence*: *IfcProduct*, *IfcProcess*, *IfcResource*.

VffProcessType e *VffProcess*, poi, sono definite come sottoclassi di *IfcTypeProcess* e *IfcProcess* rispettivamente, e sono usate per modellare processi di trasformazione generici che, fornito un dato input, restituiscono un certo output in accordo con specifiche regole e usando uno determinato set di risorse, in altre parole utilizzano una "ricetta".

Un processo può essere descritto nella sua interezza oppure essere scomposto in sottoclassi grazie alla classe *IfcRelNests*. Inoltre, vincoli di precedenza tra i processi possono essere definiti mediante la classe *IfcRelSequence*, mentre le entità in input e output di un processo possono essere collegate mediante le classi *IfcRelAssignsToProcess* e *IfcRelAssignsToProduct*, rispettivamente.

La *VffProductionResourceType* e la *VffProductionResource* sono rispettivamente sottoclassi di *IfcTypeResource* e *IfcResource* e modellano una generica risorsa utilizzata in fabbrica (insieme ai suoi sistemi di produzione). Queste classi sono ulteriormente specializzate per rappresentare risorse riferite all'attrezzatura, ai

materiali e alle risorse umane. Nel VFDM le classi *VffMachineryElementType* e *VffMachineryElementType* sono state definite come sottoclassi di *IfcTypeProduct* e *IfcProduct*, rispettivamente, e sono atte a rappresentare le generiche parti che compongono l'attrezzatura della macchina. Infine, sono state create classi con specifiche proprietà (ad es. *VffProcessProperties* e *VffMachineryElementProperties*) per caratterizzare adeguatamente i processi, le risorse e gli elementi di macchine.

In seno al lavoro di tesi, è interessante riprendere e sottolineare che le sottoclassi *VffProcessType* e *VffProcess* sono specializzate nella rappresentazione della produzione, dell'assemblaggio, della manutenzione e del processo di movimentazione dei materiali.

Si vuole mettere l'accento in particolar modo sull'ultima di queste rappresentazioni: il **trasporto interno** identificato con la parola *Handling*, in quanto l'elaborato di tesi parte proprio da tale rappresentazione.

La sottoclassi appartenenti alla classe *VffProcessType* sono: *VffManufacturingProcessType*, *VffAssemblyProcessType*, *VffMaintenanceProcessType* e *VffHandlingProcessType*. La sottoclasse *VffManufacturingProcessType* è ulteriormente specializzata per *VffMachiningProcessType* che a sua volta è sottoclasse di *StepNcMachiningWorkingstep*.

VffProcessProperties è la classe che rappresenta un insieme di proprietà predefinite di un processo di trasformazione generica. Gli individui di questa classe possono essere associati a individui appartenenti sia a *VffProcess* sia *VffProcessType* [7].

1.2 Processo di Produzione e Trasporto

Prima di entrare nel merito del problema, si vogliono introdurre anche i concetti chiave della movimentazione dei materiali per

definire al lettore il dominio di riferimento in maniera puntuale. Questo per definire le caratteristiche che entrano in gioco in fase di formalizzazione della conoscenza.

La movimentazione dei materiali (o material handling (MH)) rappresenta l'insieme delle decisioni riguardanti lo spostamento fisico dei beni all'interno della fabbrica. La movimentazione dei materiali comprende tutte le attività connesse con i flussi di materiali negli impianti di produzione e di stoccaggio ed è lo strumento di integrazione tra le diverse aree operative.

La movimentazione dei materiali, in senso generale, ha lo scopo di:

- rendere disponibile, attraverso l'impiego di opportuni metodi e strumenti, la giusta quantità del giusto materiale nel posto giusto
- rispettare i tempi, le sequenze e le condizioni richieste
- effettuare tutte le attività minimizzando il costo che ne deriva.

Oltre agli obiettivi di costo, attraverso il miglioramento delle attività di gestione del MH, è possibile, infatti, perseguire l'obiettivo di miglioramento delle condizioni di lavoro in termini di tutela delle condizioni di sicurezza e minimizzazione dello sforzo per l'operatore [17].

In generale tale movimentazione si può "declinare" in (1) layout generale del sistema, (2) magazzino, (3) imballaggio e (4) sistemi di trasporto interno. In questa tesi ci si concentrerà sull'ultimo concetto, quindi il termine movimentazione dei materiali è inteso come sinonimo dei sistemi di trasporto interno.

Le attività operative che compongono il trasporto interno sono:

- trasporto

- stoccaggio
- prelievo frazionato *(picking)*
- smistamento *(sorting)*
- raggruppamento *(merging)*
- indirizzamento *(dispatching)*
- alimentazione
- posizionamento
- orientamento.

I trasporti interni possono essere, in prima istanza, classificati:

- per tipo di materiale
- per tipo di funzionamento
- per tipo di energia motrice
- per tipo di movimento
- per tipo di comando

In questo elaborato si focalizza l'attenzione sulla classificazione dei trasporti per tipo di movimento[17] presentata in Figura 1.4.

Si riportano di seguito alcuni esempi per le categorie principali di mezzi di trasporto interni in ambito industriale, molto diversi tra loro. Questo per sottolineare al lettore quanto sia necessario poter effettuare delle scelte strutturate sia in fase di progettazione del layout di sistema produttivo sia in fase di gestione dei trasporti interni. Avere differenti possibilità di trasporto significa avere differenti caratteristiche in gioco e quindi *vincoli* da considerare in sede di decision making [18].

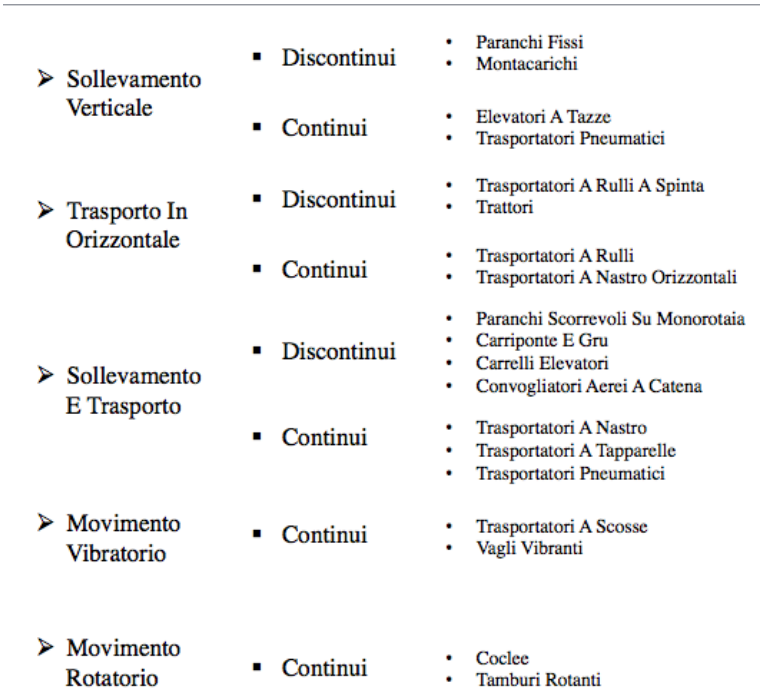


Figura 1.4: Classificazione dei trasporti per tipo di movimento [17].

Tra i mezzi di trasporto per la movimentazione interna alla fabbrica disponibili, i più comuni sono:

1. *Carrello industriale*: mezzo di trasporto di tipo discontinuo senza vincoli di mobilità. Consente la movimentazione orizzontale e verticale dei carichi. I carrelli industriali si dividono in: ELEVATORI, COMMISSIONATORI E TRASPORTATORI. Il tempo ciclo per i carrelli industriali è calcolato come segue.

$$TempoCiclo = Tempi fissi + Tempi variabili$$

I *tempi fissi* sono la somma dei tempi standard e non dipendono dalla localizzazione delle unità di carico (u.d.c.) (SEGNALAZIONE, POSIZIONAMENTO, CICLO FORCHE, CURVE). *Tempi variabili*, invece, sono la somma dei tempi di translazione (orizzontale e verticale) che dipendo-

no dalle distanze e dalle prestazioni cinematiche dei carrelli.

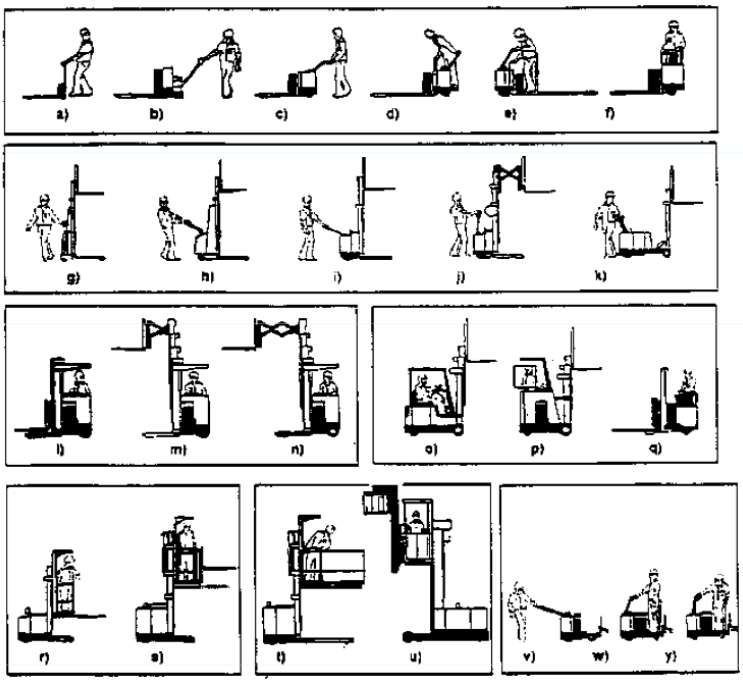


Figura 1.5: Tipologia carrelli secondo la posizione del conducente [18].

2. *Gru a Ponte o Carroponte*: consiste in un ponte costruito con uno o più travi che supportano un paranco mobile. Guide motorizzate di estremità consentono il movimento del ponte lungo le rotaie. I movimenti tipici sono quello longitudinale del ponte, appunto, quello trasversale del carrello, il sollevamento e l'abbassamento del carico effettuato per tramite dell'argano/paranco. Le velocità da considerare sono la velocità del carrello (movimentazione), velocità gancio (presa), velocità ponte/trave (sostegno). All'argano, installato su un carrello o un paranco, sono applicate una o più funi le quali, con un sistema di carrucole, rinvii e ganci o altri dispositivi di sollevamento consentono il sollevamento dei pesi. Il carroponte è un'apparecchiatura di sollevamento soggetta a specifiche normative sia costruttive

che di verifica periodica. Esistono carriponte per portate che variano da centinaia di chili a centinaia di tonnellate.

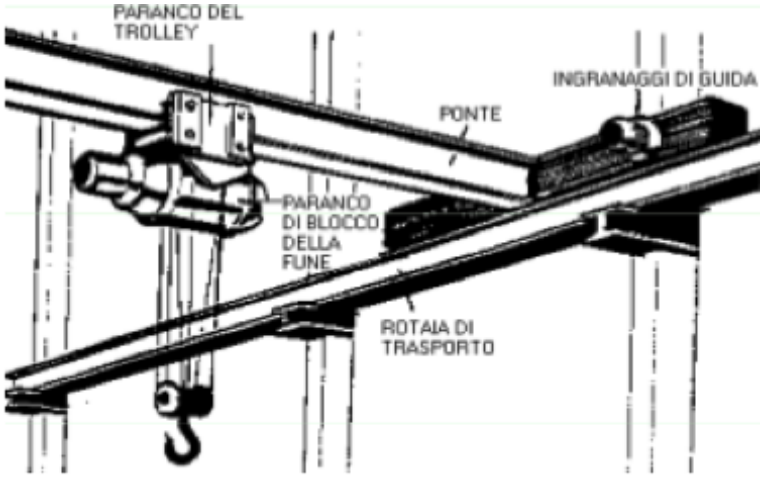


Figura 1.6: Esempi di gru a ponte [18].

3. *Convogliatori* (trasportatori di tipo continuo): Mezzi di trasporto che necessitano di installazioni fisse a terra o aeree. I sistemi di movimentazione appartenenti a questa categoria possono essere:

   - Convogliatori aerei: per la movimentazione verticale ad esempio dei capi appesi nel settore abbigliamento.
   - Carrelli automotori: composti da carrelli (motore a corrente alternata o continua simile ad AGV), da vie di corsa aeree (leghe leggere o di acciaio modulari scambi, spostamenti verticali), il sistema di gestione e controllo e i dispositivi di carico e scarico. Le prestazioni tipiche riguardano la Velocità massima tra 0.3-1.3 m/s; la Portata pari a 500Kg per i carrelli singoli e fino a 2500 Kg per i carrelli doppi.

4. *AGV: Automated Guide Vehicles Systems*: sistemi di movimentazione basati su carrelli automatici e costituiti da 4 componenti:

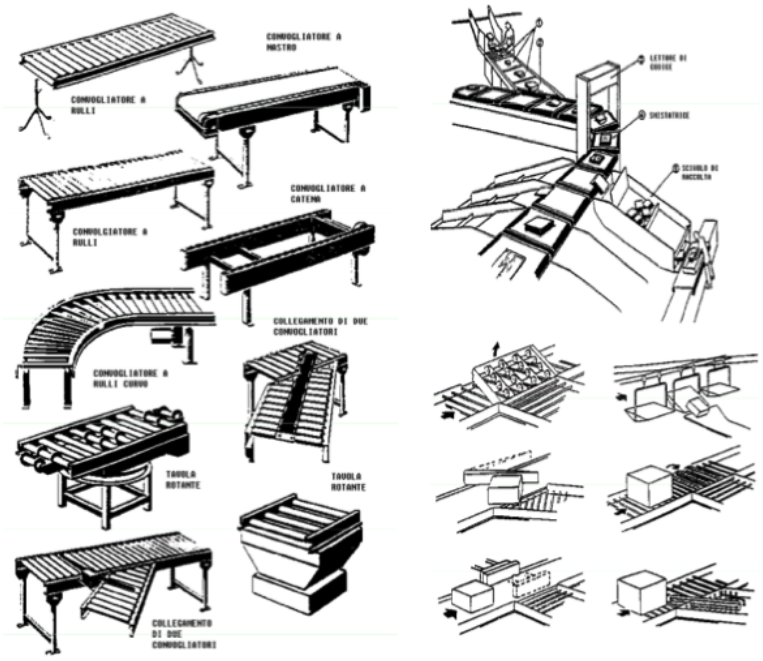


Figura 1.7: Esempi di convogliatori e sistemi di smistamento [18].

- Veicoli a guida automatica
- Impianto di guida
- Impianto di trasmissione
- Sistema di gestione (calcolatore dedicato)

L'impianto di guida può essere a *guida libera* ove la navigazione e stimata tramite programmazione software, la guida e inerziale con sistema laser, oppure a *guida fissa* di tipo magnetico (frequenza 1-100 kHz), monofrequenza per tutti i tratti e corrente solamente nel percorso da effettuare o intensità diversa nei vari tratti, multifrequenza (fino a 5) non serve trasmissione in corrispondenza ad incroci, ottica: luce riflessa/telecamera. Le prestazioni tipiche sono:

i *Velocità massima*: 1m/s (antinfortunistica), oltre 2m/s.

ii *Accelerazione/decelerazione*: 0.5 m/s;

iii *Tempo carico/scarico*: oltre 20s compreso posiz.

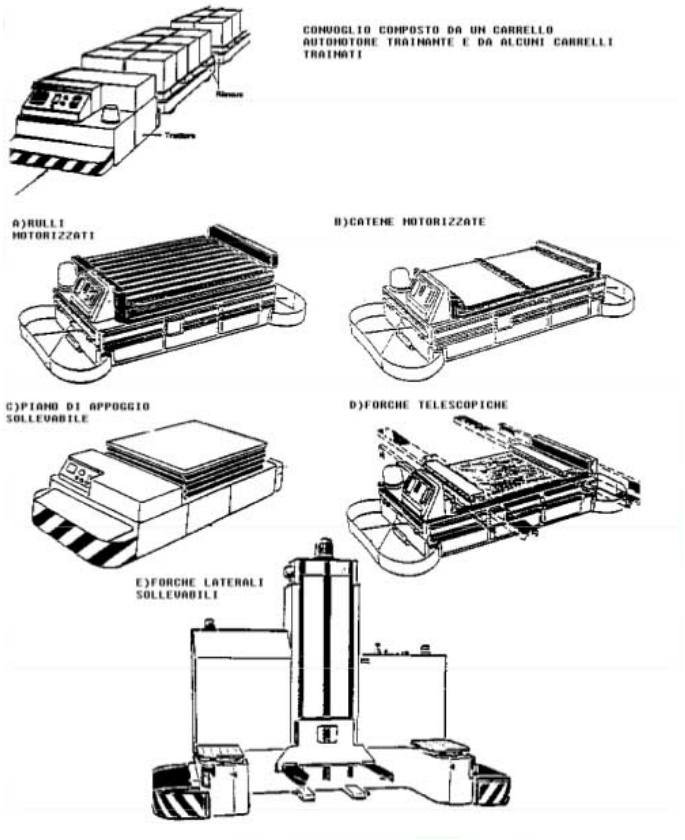


Figura 1.8: Esempi di AGV [18].

iv *Tempo di comunicazione e organizzazione missione*: 30 s

v *Raggio minimo di curvatura*: 1.5 m

vi *Batterie di accumulo.*

vii *Disponibilità*: prodotto delle disponibilità dei sistemi di controllo, di guida dei veicoli, veicoli, interfacce. [18]

1.3 Teoria dei Grafi

Lo scopo di questo ultimo paragrafo, infine, è quello di effettuare una sorta di richiamo alla *Teoria dei Grafi* che introduca i concetti fondamentali ad essa legati che sono stati utilizzati

nella fase conclusiva di questo lavoro di Tesi.

Un grafo $G = (V, E)$ è una struttura composta da V, set finito di elementi chiamati *vertici* e da E, un set non ordinato di coppie di nodi chiamate *archi* [20] . Un *arco diretto* o *digrafo* si definisce allo stesso modo, eccetto per il fatto che ogni arco è una coppia ordinata, dotata di direzione da un nodo "partenza" al nodo di "arrivo".

La letteratura riguardante la teoria dei grafi presenta una distinzione tra gli elementi della struttura del grafo. In particolare:

i Gli oggetti semplici sono detti sia *vertici* sia *nodi*.

ii I collegamenti tra i vertici:

- Se **orientati**, sono detti *archi* e il grafo è detto *orientato*
- Se **non orientati**, sono detti *spigoli* e il grafo è detto *non orientato*

Sia per quanto riguarda i grafi non orientati sia per quelli orientati, il generico arco dal nodo i al nodo j è formalmente indicato come (i, j), sebbene risulti maggiormente appropriato indicare con $\{i, j\}$ l'arco generico di un grafo non orientato. Un arco (i, i) è detto *loop*.

Solitamente risulta utile disegnare il grafo. Ovviamente il computer che dovrà analizzare le sue caratteristiche non troverebbe lo stesso giovamento dell'individuo. Quindi per rappresentare i grafi in modo appropriato per il calcolatore sono necessarie: una *lista degli archi*, una *matrice di incidenza* e una *matrice di adiacenza*.

La *lista degli archi* contiene semplicemente un ingresso per ogni arco (i, j). Nel caso dell'esempio riportato in Figura 1.9, tale elenco conterrà $(1, 2)$, $(1, 3)$, $(1, 4)$, $(2, 3)$, $(2, 5)$, $(3, 4)$, $(3, 5)$,

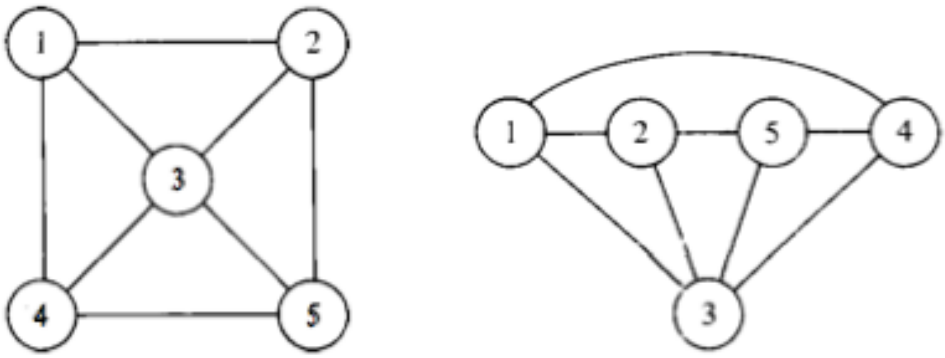


Figura 1.9: Due rappresentazioni grafiche dello stesso grafo.

$(4, 5)$. Ogni lista può essere raggruppata, ordinata e manipolata in differenti modi.

Nel grafo non diretto si dice che lo spigolo (i, j) è incidente su i e j. Nel grafo diretto si dice che l'arco diretto (i, j) è incidente da i a j, ossia l'arco esce da i ed entra in j. La *matrice di incidenza nodo-arco* ha tante righe quanti sono i nodi e tante colonne quanti sono gli archi del grafo in analisi. Nella colonna relativa ad un arco (i, j) si avranno tutte le componenti pari a 0 tranne quelle che si trovano nella riga i e nella riga j. Se il grafo è *non orientato* queste due componenti saranno entrambe pari a $+1$, mentre se il grafo *orientato*, la componente relativa al nodo predecessore i sarà pari a $+1$ e quella relativa al nodo successore sarà -1.

Considerando un generico grafo orientato come quello in Figura 1.10, se ogni arco è numerato con indice k, allora la *matrice di incidenza* $B = (b_{i,k})$ è definita come segue:

- $b_{i,k} = +1$ se l'arco k è incidente verso il nodo i.
- $b_{i,k} = +1$ se l'arco k è incidente dal nodo i.
- $b_{i,k} = 0$ altrimenti.

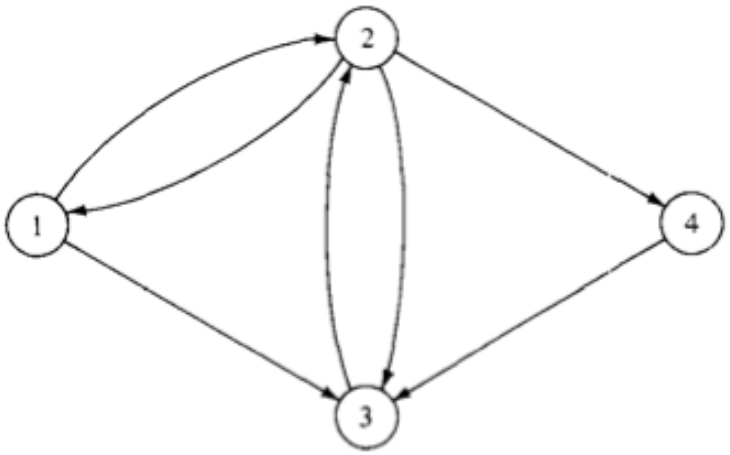


Figura 1.10: Esempio di grafo orientato *(digrafo)*.

$$B = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{ccccccc} (1,2) & (1,3) & (2,1) & (2,3) & (2,4) & (3,2) & (4,3) \end{array} \\ \begin{pmatrix} -1 & -1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & -1 & -1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & -1 \end{pmatrix} \end{array}$$

Ogni colonna contiene esattamente un $+1$ e un -1. Se fossimo nel caso del grafo non orientato, si noterebbero esattamente due 1 per ogni colonna.

Se esiste un arco (i, j) si dice che i nodi i e j sono adiacenti. Per definizione, nessun nodo è adiacente verso se stesso. Un grafo $G = (V', E')$ è detto *sottografo* del grafo $G = (V, E)$, se $V' \subseteq V$ e $A' \subseteq A$.

Due nodi i e j si dicono connessi se esiste un cammino $\{i, j\}$. Un grafo G si dice connesso se tutte le coppie di nodi sono connesse. Un componente di un grafo G è connesso al massimo nel sottografo. Per esempio, dato un nodo di partenza, da questo è possibile raggiungere un dato insieme di nodi. E, partendo poi da uno qualunque di questi è possibile raggiungere uno qualunque dei nodi appartenenti allo stesso insieme.

I valori numerici posso essere assegnati ai nodi o agli archi di un grafo per rappresentare i costi, la capacità minima dei flussi, le probabilità di guasto, e così via. In generale nel caso di assegnazione di strutture addizionali è più corretto parlare di *network* piuttosto che di grafo [20].

1.3.1 Il Problema del Cammino Minimo: il Metodo Bellman-Ford

Si supponga che a ogni arco (i, j) di un digrafo venga assegnata una "lunghezza" a_{ij}. Un naturale e intuitivamente interessante problema è quello di trovare il cammino diretto più breve da una specifica origine a una specifica destinazione, senza mai passare due volte dallo stesso nodo. E' possibile formulare e risolvere come problema del cammino minimo (o *shortest-path problem* una grande varietà di problemi di ottimizzazione.

Formalmente, il cammino (o path) è la sequenza di nodi $< v_0, v_1, ..., v_k >$ in un grafo $G = (V, E)$ tale che ogni nodo è connesso con il vertice successivo nella sequenza (gli archi (v_i, v_{i+1}) per $i = 0, 1, ..., k - 1$ sono gli archi del set E).

Nel problema del cammino minimo si assegna ad ogni arco un valore reale che si definisce *peso*, il problema del cammino minimo tra due vertici è la ricerca del percorso che minimizza la somma dei costi associati all'attraversamento di ciascun arco [20].

Una soluzione al problema dello shortest-path è quello che viene chiamato un "algoritmo di tracciamento (di rotta)" (*pathing algorithm).* I più importanti algoritmi di questa categoria sono:

- Algoritmo di Dijkstra: risolve problemi con una sola sorgente se tutti i pesi degli archi sono maggiori o uguali a zero. Senza richiedere un elevato tempo d'esecuzione, questo algoritmo può infatti calcolare la strada più breve da un determinato nodo di partenza p a tutti gli altri nodi del grafo.

- Algoritmo di Bellman-Ford: risolve problemi con una sola sorgente, anche se i pesi degli archi sono negativi
- Algoritmo di Floyd-Warshall: risolve tutte le possibili coppie
- Algoritmo di Johnson:risolve tutte le coppie, può essere più veloce dell'algoritmo di Floyd-Warshall su grafi sparsi

L'algoritmo che si è scelto di utilizzare in questa tesi è quello di **Bellman-Ford** che calcola i cammini minimi di un unica sorgente su un grafo diretto pesato (dove alcuni pesi degli archi possono essere negativi). L'algoritmo è nella sua struttura base molto simile a quello di Dijkstra, ma invece di selezionare il nodo di peso minimo, tra quelli non ancora processati, semplicemente processa tutti gli archi e lo fa $|V| - 1$ volte, dove $|V|$ è il numero di archi nel grafo. Le ripetizioni permettono alle distanze minime di propagarsi accuratamente attraverso il grafo poiché, in assenza di cicli negativi il cammino minimo può solo visitare ciascun nodo al più una volta. Bellman-Ford ha una complessità temporale pseudo-polinomiale, vale a dire $O(|V||E|)$ ove $|V|$ ed $|E|$ sono rispettivamente il numero di vertici e di archi del grafo. Ciò è dovuto al fatto che non possono essere fatte assunzioni sull'ordine nel quale gli archi vengono controllati e le relative etichette aggiornate. In effetti, però, dando sistematicità ai controlli e agli aggiornamenti è possibile tenere sotto controllo il numero di iterazioni. La procedura che segue l'algoritmo di Bellman-Ford è schematizzabile come segue (questa implementazione prende in ingresso un grafo, rappresentato come liste di vertici ed archi, e modifica i vertici in modo tale che i loro attributi distanza e predecessore memorizzino i cammini minimi) :

- **Passo 1**: Inizializzazione del grafo.
  Per ogni nodo (vertice) v:

se v è il nodo sorgente (di partenza) allora distanza(v) = 0
il predecessore di v è se stesso;
altrimenti distanza(v) = infinito

- **Passo 2**: Processamento degli archi reiterato.
  Per $i = 1, ..., k - 1$ ove k=numero totale di nodi
  Per ogni arco (u, v):
  u = sorgente (nodo di partenza);
  v = destinazione;
  se $v > u + peso(u, v)$:
  allora:
  $v = u + peso(u, v)$
  $predecessore(v) = u$.

Capitolo 2

Modello Ontologico per la Formalizzazione della Conoscenza Inerente al Sistema Produttivo

Il primo problema affrontato in questa tesi è la formalizzazione della conoscenza riferita ai trasporti tra le diverse fasi di un processo produttivo. L'obiettivo è quello di renderla strutturata, estensibile e, soprattutto, condivisa.

Il seguente Capitolo descrive il **modello ontologico** che ha lo scopo di consolidare la conoscenza relativa ai trasporti interni all'impianto produttivo e metterla in relazione con quella riferita agli step di processo produttivo già ampiamente rappresentata nel VFDM.

2.1 Strumenti Utilizzati per la Formalizzazione della Conoscenza

Come si è detto, il problema dello sviluppo di un modello di dati "generale" per il dominio di produzione viene affrontato attraverso l'utilizzo delle ontologie basate su ben precisi standard tecnici.

Attraverso l'utilizzo degli standard, gli attributi riguardanti il dominio che si vuole descrivere e rappresentare formalmente vengono "normati" e quindi slegati da colui che li definisce.

Nonostante la numerosità di standard tecnici oggi disponibili, si deve evidenziare il fatto che questi forniscano un set di caratteristiche molto specifici in relazione al settore a cui si indirizzano. L'estensibilità degli standard, per permettere una agevole applicazione in settori diversi, è garantita attraverso l'utilizzo delle "wildcard" per aggiungere ulteriori informazioni più complesse, a seconda della necessità del progettista. Quindi lo standard permette di modellare eventuali caratteristiche aggiuntive, perdendo, però, il requisito di "condivisibilità" dei dati.

Le ontologie, dal canto loro, sono una "rappresentazione formale, condivisa ed esplicita di una concettualizzazione di un dominio di interesse".Creano, cioè, il presupposto per la condivisione della conoscenza tra differenti applicativi, permettendo un flusso fluido dei dati tra entità differenti.

L'ontologia, quindi, risolve il problema della generalizzazione della struttura grazie a una sorta di *grammatica* che permette di specificare delle relazioni tra oggetti, così chiunque conosca quella grammatica (o consulti il vocabolario OWL 2.0, per esempio) può comprendere il significato di ogni relazione.

Alla luce di ciò, risulta molto interessante l'idea del VFDM di usare questa grammatica per definire le relazioni e sfruttare standard già esistenti, come quelli elencati nel paragrafo 1.1.2, per cercare di favorire l'interoperabilità tra strumenti software. Questo approccio richiede sia la traduzione delle norme esistenti in un linguaggio comune, sia lo sviluppo delle estensioni necessarie alla rappresentazione di concetti nel campo di applicazione del VFF [1]. Questa formula di integrazione e aggregazione di standard differenti rappresenta sicuramente un grande punto di forza del VFDM, in quanto, così facendo, si ottengono due vantaggi contemporaneamente:

i utilizzo integrato di diversi standard.

ii possibilità di usare le ontologie che permettono di sopperire alla limitazione derivante dalla necessità di un modello di

dati centralizzato, permettendo invece uno schema a dati distribuiti.

2.2 Approccio al Problema in Termini di Virtual Factory Data Model

Come anticipato nei capitoli precedenti, all'interno del VFDM la formalizzazione della conoscenza relativa alla movimentazione dei materiali è fornita dalla classe *HandlingProcess* (sottoclasse della classe *Process*, paragrafo 1.1.4) a cui è legata la caratteristica: tempo di trasporto.

Considerando il fatto che, come accennato precedentemente, le caratteristiche in gioco non riguardano solo il tempo di trasporto, questa parte della tesi è volta a formalizzare il trasporto interno, in modo generico, condivisibile ed estensibile.

2.2.1 Il Virtual Factory Data Model per il Processo

Tra gli aspetti interessanti del VFDM spicca la questione della definizione del *Processo*.

I rapporti tra i processi e le risorse possono essere formalizzati come mostrato in Figura 2.1, dove i box rappresentano le classi e gli archi rappresentano le restrizioni di proprietà che collegano le classi secondo la Manchester OWL Syntax [7] [15]. Inoltre, tale figura mostra come i dati di progetto del sistema (parte superiore della figura) possano essere collegati con i dati di esecuzione del sistema (parte inferiore della figura). Durante la fase di progettazione/pianificazione dei sistemi di produzione, i tipi risorse necessarie a un tipo processo possono essere specificate per mezzo della classe di relazione oggettivata chiamata *IfcRelAsignToProcess*, mentre, i "fornitori" delle risorse (come gli individui di classe *IfcObjectDefinition*) possono essere collegati ai tipi risorse dalla classe *IfcRelAssignsToResource* [7].

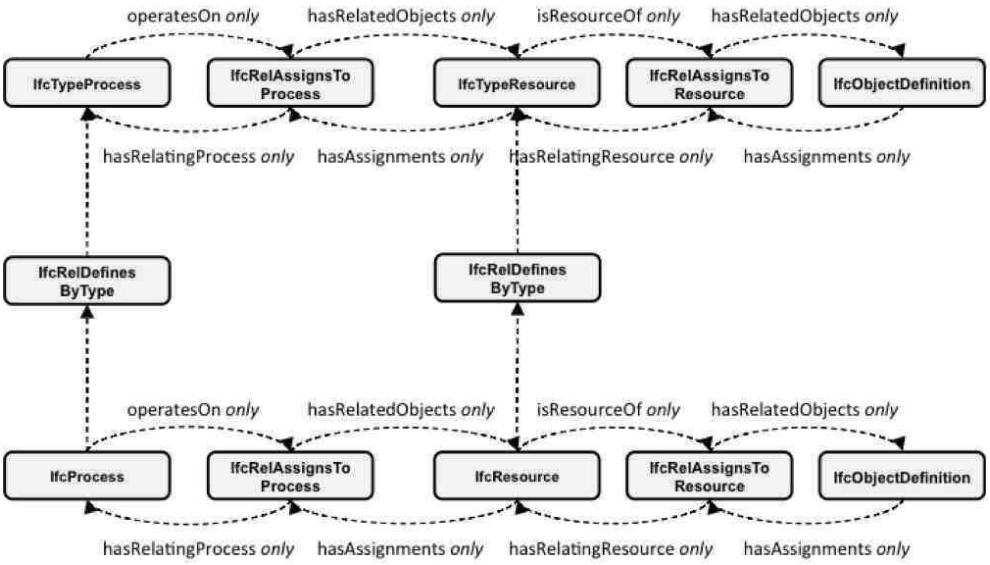


Figura 2.1: La formalizzazione del *processo* e la relazione con le sue *risorse*. [7]

Focalizzando l'attenzione sulla parte superiore rappresentata in Figura 2.1, è possibile portare un esempio applicativo per meglio comprendere la forza di tale modello, per semplicità si fa riferimento al Test Case usato da W. Terkaj e M. Urgo nel loro articolo [7].

Individui di VffManufacturingProcessType	**Description**	**Tipo risorse richieste come individui di VffProductionResourceType**
processingPlan01	Process plan	N/A
DR01	Drilling operation	drillingRes01
ML01	Milling operation	millingRes01
ML02	Milling operation	millingRes02
QC01	Quality control	qualityControlRes01
GR01	Grinding operation	grindingRes01

Tabella 2.1: Pianificazione del processo

Come schematizzato in Tabella 2.1, il caso utilizzato rappresenta una linea di produzione composta da cinque segmenti di processo produttivo ognuno dei quali richiede specifici "tipo risorsa" di produzione.

Gli stessi "tipo risorsa" sono forniti da specifici "tipo macchina" o macchine. Nella fattispecie, tenendo conto delle Tabelle 2.1 e 2.2 e volendo considerare lo step di processo ML02, per esempio, si vede che questo richiede risorse del tipo *millingRes02*, il che comporta che i "tipo macchina" o macchine che possono servire questo processo saranno rispettivamente MtC (macchine MS02 o MS03) e la MS04. In Figura 2.2 è possibile vedere come queste informazioni vengono formalizzate.

Individui di VffMachineryElement	**Relativi individui di VffMachineryElementType**	**Descrizione**	**Tipi risorse forniti come individui del VffProductionResourceType**
DS01	MtA	Drilling machine	drillingRes01
MS01	MtB	Milling machine	millingRes01
MS02	MtC	Milling machine	millingRes02
MS03	MtC	Milling machine	millingRes02
MS04	MtB	Milling machine	millingRes02
CS01	MtD	Quality control machine	qualityControlRes01
GS01	MtE	Grinding machine	grindingRes01

Tabella 2.2: Relazioni tra il tipo processo, il tipo risorsa e i *machinery element*

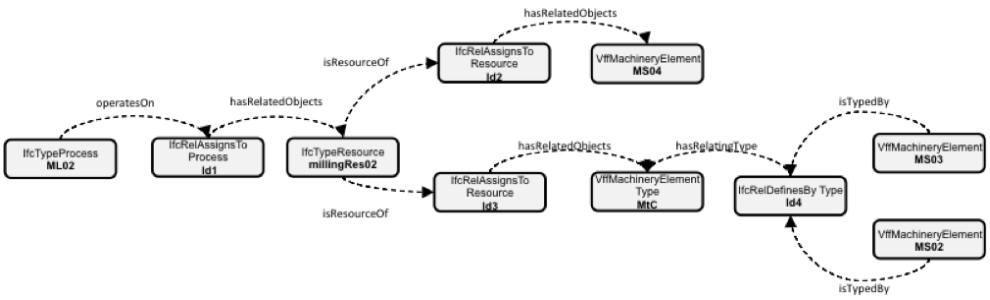


Figura 2.2: Relazioni tra il tipo processo, il tipo risorsa e i machinery element. [7]

Quindi, per definire lo step di processo esiste una "ricetta" uguale per tutti i segmenti coinvolti nella pianificazione alla quale associare le capability di processo, in questo caso le macchine e i "tipo macchina" che sono in grado di effettuare un dato step di processo.

2.2.2 Formalizzazione della Conoscenza per i Trasporti Interni

Per i trasporti interni ci si è ispirati alla "ricetta" sopra definita, ove le capability per il trasporto rappresentano l'insieme dei mezzi di trasporto disponibili per effettuare la movimentazione. In particolare, è importante sottolineare che le risorse assegnate al processo di trasporto sono tutti i mezzi in grado di effettuare **almeno** un trasporto richiesto partendo da una delle macchine a monte verso una delle macchine a valle. Per spiegare concretamente cosa si è fatto, si prendano in considerazione due *ManufacturingTypeProcess* del Test Case citato nel paragrafo 2.2.2 ai quali si vuole collegare il processo di trasporto "capace" di effettuare un dato trasporto.

In Figura 2.3 è possibile vedere l'esempio pratico di proposta di formalizzazione riferita al caso specifico di trasporto da una delle stazioni dello step di processo ML02 a una delle stazioni di QS01 (in questo caso vi è una sola stazione di controllo qualità). I set di processo sono riferiti a quelli del processPlan definito nelle Tabelle 2.1 e 2.2.

Si noti che in tale rappresentazioni non si usano gli acronimi riferiti agli standard, perché questa formalizzazione è una proposta che non è stata concretamente inserita nel VFF.

Per tale formalizzazione, si è cercato comunque di rimanere aderenti allo standard ISA-95, perciò l'*IfcTypeProcess* del processo diventa *TypeHandlingProcess* [1] per il trasporto, in egual modo *IfcTypeResource* diventa *TypeHandlingResource*, mentre i *VffMachineryElement* diventano *TransportElement*.

Quindi, la principale differenza tra la formalizzazione della classe *TypeProcess* e quella della classe *TypeHandling* risiede nel fatto che nel primo caso si associano al tipo risorsa tutti i *MachineryElement* o *MachineryElementType* indipendentemente dalla

[1] Non si inserisce l'acronimo degli standard utilizzati dal VFDM perché la formalizzazione non è stata inserita nel progetto VFF.

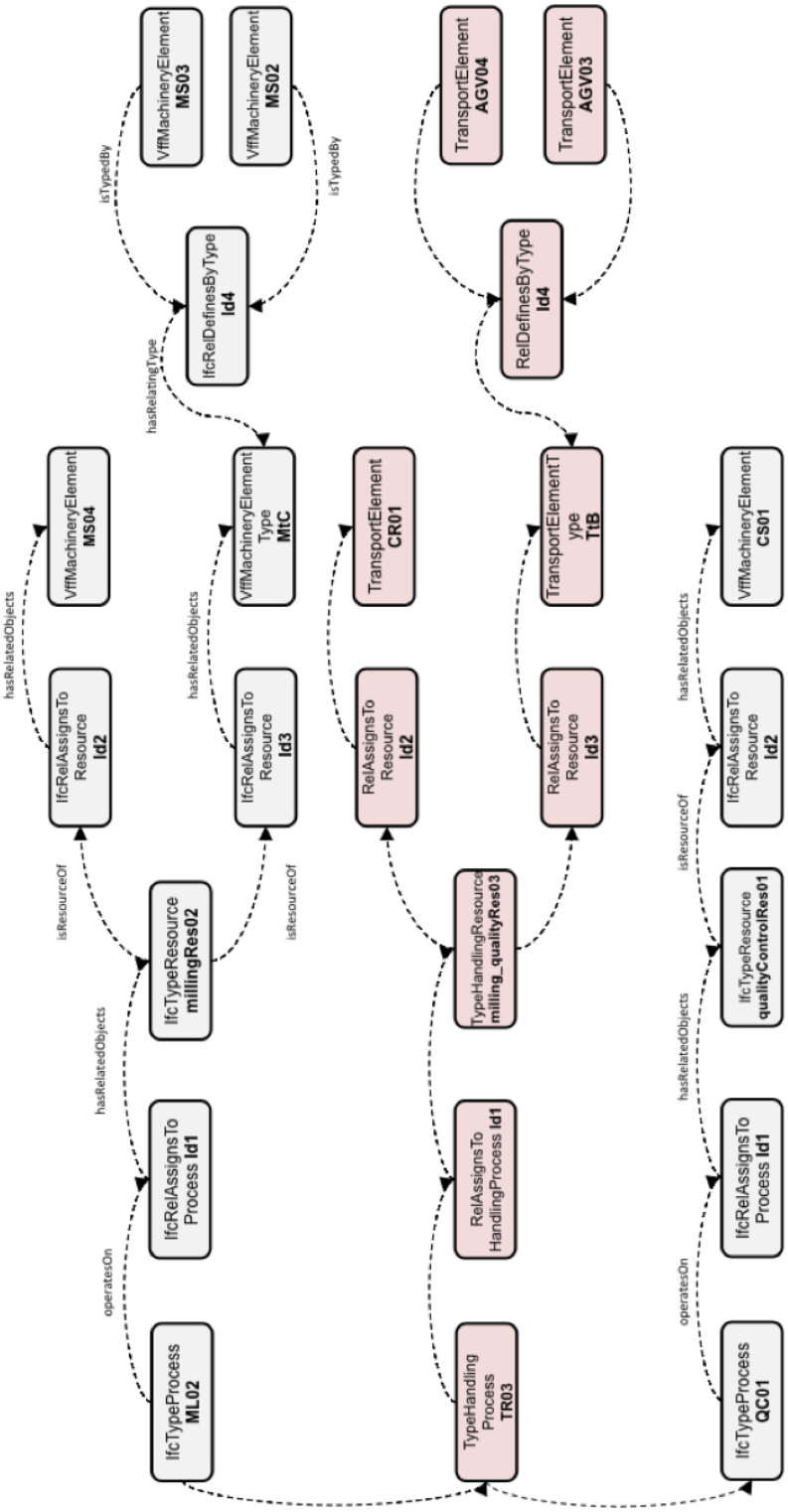


Figura 2.3: Formalizzazione del processo di trasporto in relazione a due step di processo.

scelta che si effettuerà in fase di scheduling o gestione; nel caso della classe di trasporto, invece, si aggregano tutte le risorse fisiche, ossia i mezzi di trasporto (o tipo mezzo di trasporto) che sono in grado di effettuare quel tipo di trasporto, ovvero che colleghino almeno un *MachineryElement* del set di processo a monte e uno di quello a valle.

Ciò implica che non tutti i trasportatori siano in grado soddisfare tutte le operazioni di trasporto possibili, nasce perciò il problema di **fattibilità di utilizzo del trasportatore assegnato**.

Per questo motivo serve definire a priori e in maniera formale tale *fattibilità*, e per questo si è creato un *Dominio Geometrico* del singolo mezzo di trasporto che definisse il **luogo dei punti che può essere servito dal trasportatore**. In Figura 2.4 si riporta il Dominio Geometrico riferito all'esempio trattato precedentemente, in particolare si tratta di una schematizzazione dell'impianto ove i box possono essere generiche stazioni di lavoro, buffer, magazzini, specifici scaffali nel magazzino o generici punti di interesse.

In questo caso si tratta del Dominio dei trasportatori AGV03 e AGV04 che, tra l'altro, condividono alcuni punti di interesse, ossia la macchina MS03 e la stazione CS01.

Una volta formalizzata la conoscenza, un'ulteriore problematica che deve essere valutata è la questione della relazione tra i trasporti e gli step di processo. Siccome la scelta delle macchine viene definita in *runtime*, la verifica di fattibilità deve essere necessariamente effettuata anch'essa in *runtime*.

Per verificare tale fattibilità, ossia conoscere le effettive modalità di trasporto disponibili e i trasportatori associati, è necessario ricorrere all'utilizzo di un *ragionatore*.

I programmi informatici possono, usare l'ontologia per una varietà di scopi, tra cui il ragionamento induttivo, la classificazione, e svariate tecniche per la risoluzione di problemi. Questa

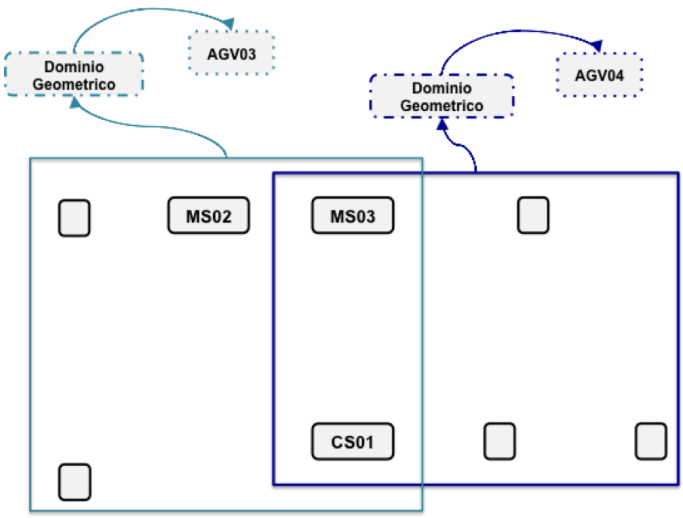


Figura 2.4: Esempio di Dominio Geometrico per due mezzi di trasporto.

particolare questione, però, verrà trattata nel capitolo successivo.

Capitolo 3

Modello Funzionale per Interrogare la Conoscenza

L'ultima questione trattata in questa tesi riguarda la connessione tra modello ontologico e un qualsiasi tool che possa elaborare la conoscenza e fornire opportunamente le informazioni richieste dall'utente o da metodi analitici, ad esempio, per la valutazione delle prestazioni.

Prima di procedere, è necessario ipotizzare di poter evitare il passaggio dalla strutturazione alla concreta formalizzazione nel VFF della conoscenza inerente i trasporti interni.

In altri termini, per poter procedere con la definizione del modello funzionale, si terrà conto della sola struttura che i dati avrebbero se fossero stati concretamente formalizzati.

Per spiegare come uno strumento possa usare questi dati, si è scelto di creare uno *use case* orientato specificatamente alla risposta di alcune delle domande che concretamente un utente, oppure un metodo analitico di configurazione, potrebbero porre al VFF. Le specifiche domande che si vogliono analizzare in questo *use case* sono:

- Trasportatori disponibili per la movimentazione
- Raggiungibilità di un punto di interesse
- Calcolo dei tempi di movimentazione
- Identificazione del mezzo più vicino al punto di interesse dal quale parte il trasportatore
- Numero e identità dei trasportatori coinvolti in una movimentazione
- Zone occupate dal trasportatore (o dai trasportatori) durante lo spostamento

3.1 Strumenti per Interrogare la Conoscenza

Per lo sviluppo di un modello ontologico, il VFDM adotta in particolare le tecnologie del Semantic Web che offrono la possibilità al VFF di rappresentare una semantica formale, un modello efficiente, una gestione distribuita dei dati, una interoperabilità delle diverse applicazioni. In particolare, come detto nel paragrafo 1.1.2, il VFDM utilizza il vocabolario OWL [15] che permette di scrivere delle ontologie tramite classi, relazioni fra classi e individui appartenenti a classi.

La conoscenza così formalizzata è processabile automaticamente da un calcolatore, tramite un ragionatore automatico che implementa i processi inferenziali e deduttivi. Questo ragionatore, direttamente legato alle ontologie, tratta proposizioni logiche del primo ordine (rapporto tra nomi e oggetti, l'esistenza e la negazione).

In riferimento all'elenco di domande sopracitato, quanto detto implica che, grazie al reasoner, sia possibile rispondere direttamente per esempio alla questione dei trasportatori disponibili per la movimentazione (se, cioè, gli oggetti interrogati appartengono al dominio di competenza dello stesso trasportatore), oppure se un punto di interesse sia raggiungibile o meno. Diversamente, il reasoner del primo ordine non è in grado di rispondere ai quesiti più articolati come la valutazione di percorsi alternativi o l'identificazione dei mezzi di trasporto più veloci per effettuare la movimentazione, poiché le logiche che entrano in gioco sono di ordine di complessità superiore al primo.

Per far fronte a questa problematica, si è pensato di sovrapporre alla formalizzazione della conoscenza, un altro livello di complessità, vale a dire associare all'ontologia un grafo, isolato alla parte relativa ai trasporti, che è un metodo conosciuto e consolidato per trattare il problema della movimentazione dei materiali, e che permette di risolvere le questioni più complesse, come ad esempio la ricerca del *cammino minimo*.

3.2 Configurazione dei Dati Formalizzati Secondo la Teoria dei Grafi

Sin dagli anni 60, molti problemi di natura non solo ingegneristica sono stati formulati attraverso il *linguaggio* della teoria dei grafi. Intuitivamente questi problemi sono quelli di natura logistica e distributiva, come quelli qui trattati, ma non solo, in una visione più ampia, si potrebbe far riferimento anche a problemi legati al disegno e progettazione di reti, a problemi di affidabilità oppure problemi di studio di interdipendenze logico-temporali

tra attività interconnesse, come nel caso di gestione dei progetti di medie e grandi dimensioni [19] .

Ogni sistema o struttura che può essere ricondotta in modo astratto a un set di elementi, alcune coppie dei quali sono relazionate in maniera specifica, può essere rappresentata con un grafo o con un digrafo. Per questo motivo, la teoria dei grafi è in realtà la teoria delle relazioni, dove i grafi rappresentano le relazioni simmetriche e i digrafi quelle asimmetriche [20].

Il primo passo da compiere per poter manipolare la conoscenza formalizzata e quindi verso la formulazione del modello, è la rappresentazione del sistema produttivo come network composto da nodi e archi opportuni. Quindi si creano tanti grafi quanti sono i trasportatori, e per ognuno di essi: (1) si rappresenta con un **nodo** ogni *punto di interesse raggiungibile* appartenente al Dominio Geometrico; (2) se due nodi sono collegati dal trasportatore si crea un **arco** che li unisce; (3) per ogni arco si associa un **peso**.

I particolare, i nodi possono rappresentare:

- un punto di interesse generico
- una load/unload station
- un macchinario
- una generica stazione
- un parcheggio mezzi di trasporto
- un magazzino prodotti finiti
- un magazzino semilavorati
- un magazzino materie prime
- un buffer

I pesi associati all'arco possono essere di diversa natura:

- tempo di trasporto impiegato dal nodo di ingresso al nodo di arrivo
- spazio percorso dal nodo di ingresso al nodo di arrivo

Una volta scelto il tipo di peso da associare, è necessario mantenere la coerenza di unità di misura dei pesi per tutti i trasportatori.

Inoltre, siccome esiste la possibilità che più trasportatori in un impianto possano condividere lo stesso punto di interesse si è pensato di creare un tipo di nodo e arco ad hoc, ossia il nodo riferito alla *zona adibita al cambio mezzo*

di trasporto e l'arco il cui peso associato è il *tempo impiegato dai trasportatori per effettuare il cambio.*

Una volta definiti e posizionati i nodi su una mappa che schematizza la fabbrica e i relativi archi che indicano se i nodi sono connessi, è necessario formulare delle ipotesi sulle quali basare il modello dei grafi:

Hp. i focus sul mezzo di trasporto: la formalizzazione del modello attraverso l'uso della teoria dei grafi e degli algoritmi ad essa legati si riferirà al mezzo di trasporto, alle sue "caratteristiche" e al "percorso" che deve effettuare.

Hp. ii l'utente deve fornire a priori l'elenco dei mezzi di trasporto liberi e *feasible* che si intendono "valutare" in fase di runtime.

Quindi, come si è detto, ogni trasportatore avrà il proprio grafo associato, per questo il sistema produttivo avrà una configurazione che potremmo chiamare "Non Planare". Si tratta di una rappresentazione *tridimensionale* che prevede la sovrapposizione di ognuno dei livelli riferiti ai trasportatori presenti nell'impianto.

Riprendendo l'esempio riportato nel paragrafo 2.2.2 (Figura 2.4), la configurazione si presenta come in Figura 3.1.

Messa a confronto con una configurazione ad esempio di tipo "Planare" (Figura 3.2), la prima configurazione è preferibile, perché pagando il prezzo di una maggiore, ma comunque irrisoria, difficoltà nella rappresentazione grafica, la "Non Planare" definisce in modo chiaro ed esplicito la questione del cambio mezzo di trasporto e dei nodi condivisi.

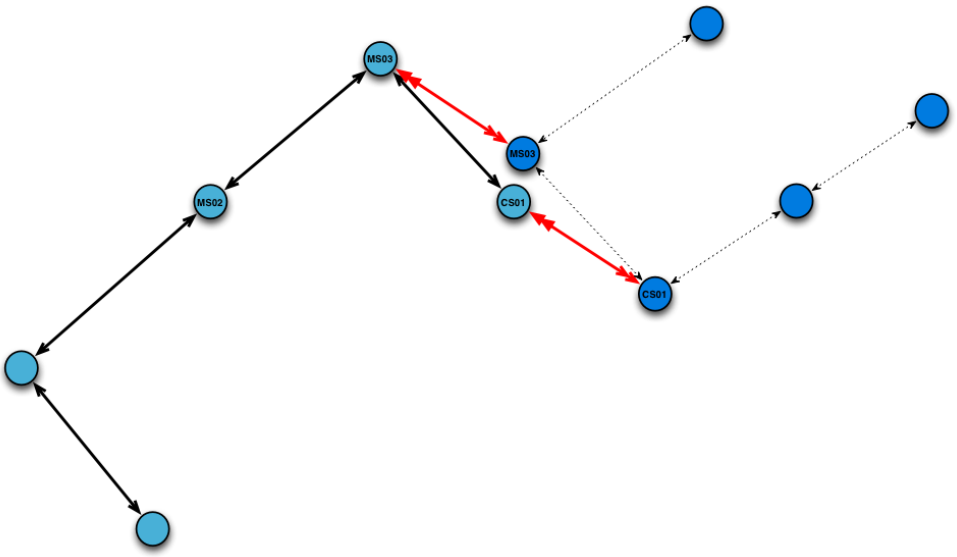


Figura 3.1: Configurazione Non Planare per i Domini Geometrici dell'esempio in Figura 2.4.

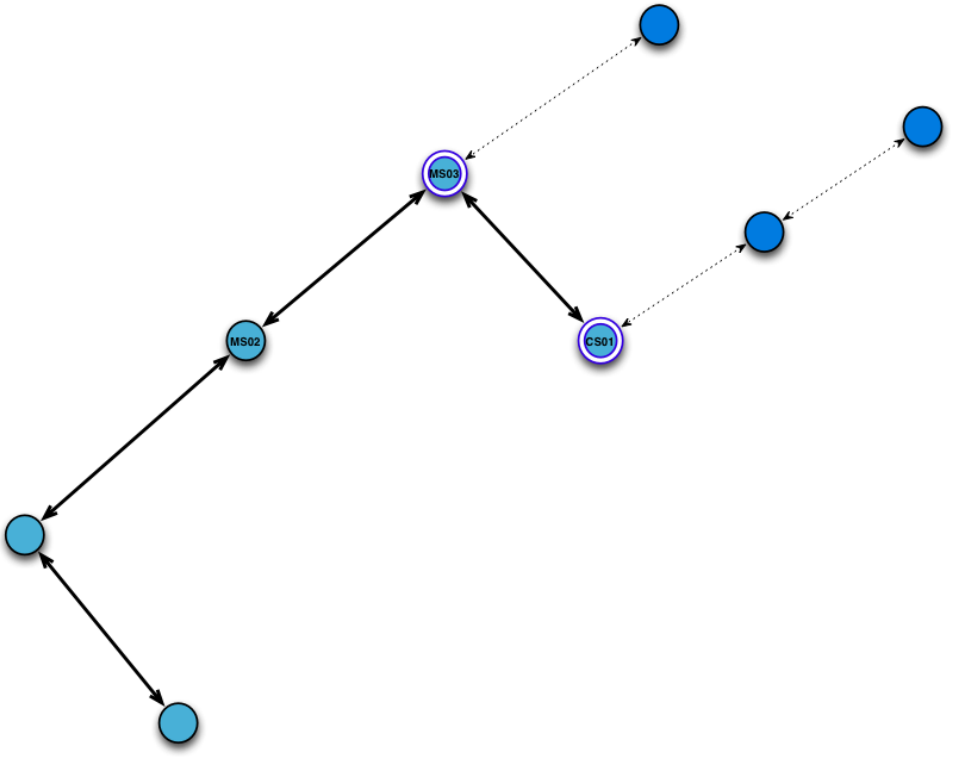


Figura 3.2: Configurazione Planare per i Domini Geometrici dell'esempio in Figura 2.4.

3.3 Scelta del Tool per la Manipolazione della Conoscenza

Per poter manipolare i dati configurati attraverso la teoria dei grafi, è possibile utilizzare differenti tool.

Tra questi è possibile citare software come **Mathematica**, che mette a disposizione un ambiente interattivo con cui definire oggetti di tipo matematico ed eseguire calcoli utilizzando funzioni offerte dall'ambiente di base o da uno dei suoi package aggiuntivi, ovvero consentendo all'utente di costruire delle funzioni utilizzando un linguaggio di programmazione interno. Sono numerosissime le funzioni disponibili nel package DiscreteMath::Combinatoria che consentono di operare su grafi ed alberi. Si tratta però di un software proprietario e rigido nell'utilizzo.

Per poter avere la massima flessibilità, quindi, si è scelto di legare al VFF programmi in linguaggio C++ che sono modificabili a seconda delle necessità e che sono perfettamente adattabili alla effettiva configurazione della conoscenza. Ciò significa che, senza effettuare alcuna modifica alla formalizzazione definita in precedenza, il programma prende in input i dati e, a seconda delle domande che l'utente o il modello analitico pongono, sfrutta un algoritmo differente.

Nello specifico, si è scelto di sfruttare librerie già esistenti che trattano la questione dei grafi. Boost dedica, infatti, particolare attenzione ai grafi nella libreria chiamata Boost::Graph. Per la costruzione del grafo che prende in input i nodi, gli archi e i relativi pesi, si utilizza il *Directed Acyclic Graph*.

Per la manipolazione delle informazioni, invece, esistono numerosi algoritmi utilizzabili, ognuno con funzionalità differenti.

Come si è visto dalla numerosità degli algoritmi proposti dalla stessa libreria Boost::Graph, esistono molti problemi di ottimizzazione che possono essere formulati e risolti come problemi di *cammino minimo*, o *shortest path*, (si faccia riferimento al paragrafo 1.3.1). Problemi di questo tipo sono probabilmente i più importanti tra tutti i problemi di ottimizzazione combinatoria [20].

Non a caso le domande poste nello use case sono inerenti al calcolo del cammino minimo, proprio perché si tratta del problema più naturale e intuitivamente interessante nell'ambito dei grafi e di riflesso nell'ambito della movimentazione dei materiali.

Nello specifico, per rispondere alle questioni dello use case, si è scelto di usare l'algoritmo Bellman-Ford [22] [23] [20] [24] che risolve un problema single-source per un grafo.

3.3.1 Il Programma in Linguaggio C++

Per poter "leggere" la conoscenza formalizzata, è stato necessario creare un programma che usando il linguaggio C++ e le librerie Boost, leggesse il grafo e fosse in grado di fornire le risposte che sono state poste nello use case.

Come è stato detto, si è usata la libreria Boost::Graph per definire i grafi e poi si è sfruttata la funzione già esistente fornita dalla stessa libreria per manipolare tali grafi.

Nella fattispecie, prima di chiamare la funzione bellman_ford_shortest_paths(), è stato necessario inizializzare il grafo attraverso l'utilizzo delle funzionalità appartenenti al *MutableGraph*, tra cui:

- G un type che è un modello di Graph.
- g un oggetto del type G.
- e un oggetto del type boost::graph_traits < G >::edge_descriptor.
- u,v oggetti del type boost::graph_traits< G >::vertex_descriptor.
- iter un oggetto del type boost::graph_traits< G >::out_edge_iterator.
- p un oggetto del type che modella Predicate e il cui argomento type unisce il type edge_descriptor.
- Property un type usato per specificare le proprietà dei nodi, degli archi e dei sottografi.

Il programma usa le classi già fornite dalla libreria, quindi lo scheletro è pre-definito. Per creare il grafo e per utilizzare la funzione ci si è attenuti a quanto scritto nel manuale delle librerie Boost.

Per prima cosa si sono definite le Property, e si è inizializzato il Grafo come Graph G0(N) (con N =numero totale di nodi presenti nel grafo completo) con l'elenco relativo agli N nodi. Dopo di che si inizializzano i sottografi, uno per ogni trasportatore. In questo modo, siccome l'utente deve poter scegliere in runtime quali trasportatori siano o meno disponibili, è possibile di volta in volta aggiungere il sottografo precedentemente definito.

Ciò implica che esisteranno dei *macro-nodi* generici che rappresentano tutti i possibili punti di interesse raggiungibili da tutti i trasportatori presenti in fabbrica, ma anche dei *sotto-nodi*, ossia tutti i nodi riferiti in modo specifico al Dominio Geometrico del singolo trasportatore. Per questo motivo alcuni *macro-nodi* essendo raggiungibili da più trasportatori saranno replicati con numerazioni o nomi differenti tante volte quanti saranno i trasportatori che servono quello specifico punto.

Nella fattispecie add_edge() si riferisce a tutti gli archi possibili tra i *sotto-nodi* e add_vertex() a tutti i *sotto-nodi* che appartengono allo specifico sottografo.

Per assegnare ad ogni arco il proprio peso, è stato creato, poi, un vettore peso che, seguendo l'ordine di inizializzazione degli archi (add_vertex(u, g)), viene associato all'arco di riferimento.

Per facilitare la scrittura si è voluto arricchire il programma con la lettura da file delle coppie di *sotto-nodi* per add_vertex(u, g) e del peso relativo ad esse.

Siccome si vuole avere la funzionalità di poter definire in runtime i mezzi di trasporti disponibili (quindi i sottografi disponibili), è stato creato un set di comandi che permettono al programma di prendere in input i trasportatori disponibili e, sulla base di tali informazioni di "accendere" i sottografi effettivamente utilizzabili.

La libreria Boost relativa ai grafi permette, poi, di creare la weight_map che contiene le distanze e i predecessori dei singoli nodi appartenenti al "nuovo grafo" che è $G0$ ma con i soli vertici appartenenti ai sottografi "accesi".

Quindi, prima di poter usare effettivamente l'algoritmo Bellman-Ford, serve decidere quale sia il *macro-nodo* di ingresso e il *macro-nodo* di arrivo del cammino minimo che si vuole calcolare (anche in questo caso è stato creato un set di comandi per avere in input tale informazione). Il compilatore prenderà questi dati e, sulla base dei trasportatori disponibili calcolerà il cammino minimo riferito ai *sotto-nodi*.

Un mezzo di trasporto disponibile può essere o no in grado di effettuare la movimentazione nella sua interezza, ossia essere capace di caricare il cilindro dal nodo di ingresso e scaricarlo al nodo di arrivo. Quando, per esempio, è in grado solamente di caricare il cilindro ma non di scaricarlo, intervengono gli archi di cambio trasportatore. Nella fattispecie, dati più mezzi disponibili, è possibile effettuare il trasporto attraverso la combinazione di più trasportatori.

Il programma, leggendo i sottonodi attraversati, li confronta con i sottografi "accesi". Se rileva che tali nodi appartengono a sottonodi differenti, allora indica l'identità dei trasportatori utilizzati e il numero di volte che è stato effettuato un cambio mezzo di trasporto.

Ricapitolando, il programma, basandosi sulla lista dei trasportatori disponibili (definita in runtime dall'utente) e sulla posizione di ognuno di questi, è in grado di definire quale sia il trasportatore più indicato (perché più vicino) ad effettuare almeno il caricamento del materiale.

Inoltre è capace di valutare se un punto di interesse sia raggiungibile o meno dai trasportatori "accesi" grazie all'utilizzo dei sottografi relativi al singolo trasportatore. Sulla base di ciò, permette di calcolare i tempi di mo-

vimentazione, da un nodo di partenza a un nodo di arrivo, per tutte le possibili combinazioni di trasporto (disponibili), indicando il cammino minimo. Infine, facendo riferimento alle zone occupate dal o dai trasportatori per compiere tale cammino minimo, identifica il numero e l'identità dei trasportatori coinvolti nella movimentazione.

Capitolo 4

Caso Applicativo di Studio: Impianto FMS

4.1 Sistemi di Produzione Automatizzati: FMS

Il *National Bureau of Standards (USA)* definisce i Flexible Manufacturing System (FMS) come *"complesso di macchine (generalmente* **centri di lavoro** *CNC con cambiutensili) interconnesse mediante un sistema di trasporto pezzi/utensili (...)* **Un computer centrale controlla sia le macchine sia i sistemi di trasporto***"*.

Mentre, *CECIMO (Commitè Europeenne de Cooperation des Industries de la Machine Outil)* lo definisce così: *"Sistema* **automatico** *di fabbricazione in grado, con un minimo di interventi manuali, di produrre un tipo qualunque di pezzo compreso in una gamma o* **famiglia definita***; questi sistemi vengono generalmente studiati per la produzione* **di piccola o media serie** *di una famiglia di pezzi, in lotti di dimensioni variabili e di composizione variabile; la flessibilità del sistema è generalmente limitata alla famiglia di pezzi per i quali il sistema è stato concepito. Esso comprende mezzi per la* **programmazione della produzione** *e per il pilotaggio dei pezzi o dei componenti, nell'ambito del sistema, e comprende, generalmente, anche mezzi per la stesura di protocolli di produzione e per la registrazione di*

dati".

In generale un sistema FMS con trasportatore pallet può essere schematizzato come in Figura fig:FMS.

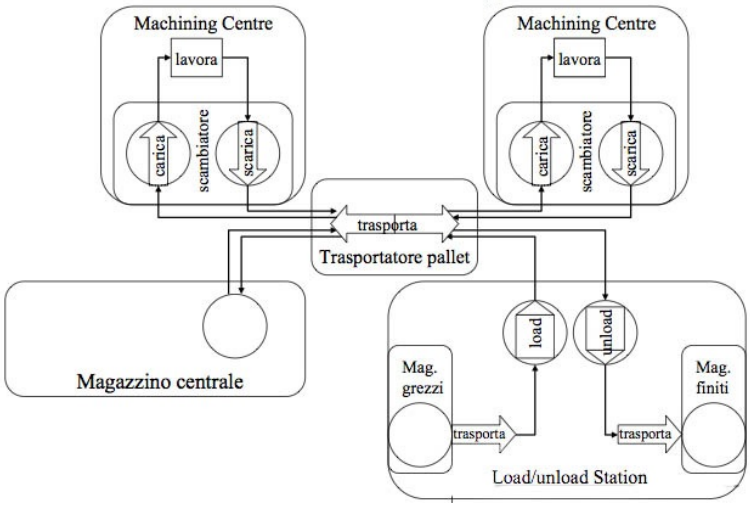


Figura 4.1: Schema generale per un sistema FMS. [28]

4.2 Il Problema Industriale

L'impianto realistico FMS che si è voluto analizzare, oltre ai *carrier* che collegano i centri di lavoro, ha anche i trasportatori pallet, nella fattispecie gli AGV (Automated Guided Vehicles).

Considerando le domande cui si vuole rispondere attraverso l'utilizzo del modello funzionale, i carrier possono essere trascurati. Per questo motivo, il problema industriale trattato considera solo il trasporto effettuato con mezzi AGV (cfr. paragrafo 1.2) . E' possibile vedere in Figura 4.2 un AGV con trasportatore pallet che carica il pallet da un'unità operativa. Come si è detto, gli impianti FMS hanno "un computer centrale che controlla sia le macchine sia i sistemi di trasporto". Questo perché i sistemi di questo tipo vogliono limitare al massimo l'intervento dell'uomo, rendendo tutto automatizzato.

Applicando, quindi, il modello VFF, si garantisce un ulteriore livello di automazione, grazie soprattutto all'integrazione di tutte le informazioni inerenti sia ai processi di trasformazione sia ai trasporti interni.

Per questi motivi il risultato ottenuto dalla tesi apre la strada ad un valido miglioramento nella pianificazione e/o configurazione dei sistemi produttivi.

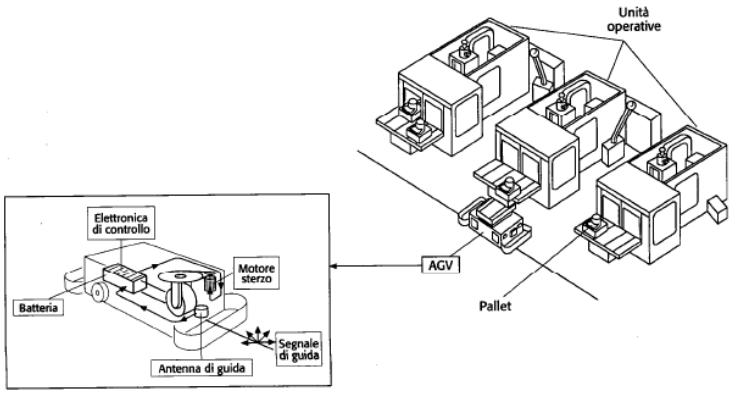


Figura 4.2: AGV che carica un pallet dall'unità operativa. [28]

Il problema industriale che è stato creato per poter effettuare una prima validazione, prende in analisi l'impianto schematizzato in Figura 4.3 che consiste in:

- un magazzino materie prime(MP), uno per i semilavorati (SL), uno per i prodotti finiti (PF)
- un parcheggio dal quale possono partire tutti gli AGV presenti nell'impianto
- tre centri di lavoro (CdL 1, 2 e 3) di dimensione mediograndе e due (CdL 4 e 5) di piccola dimensione

Inoltre, i mezzi di trasporto che possono servire gli "elementi" sopracitati sono tre AGV.

- AGV detto *m1* la cui velocità è di 1,5 $[m/s]$.

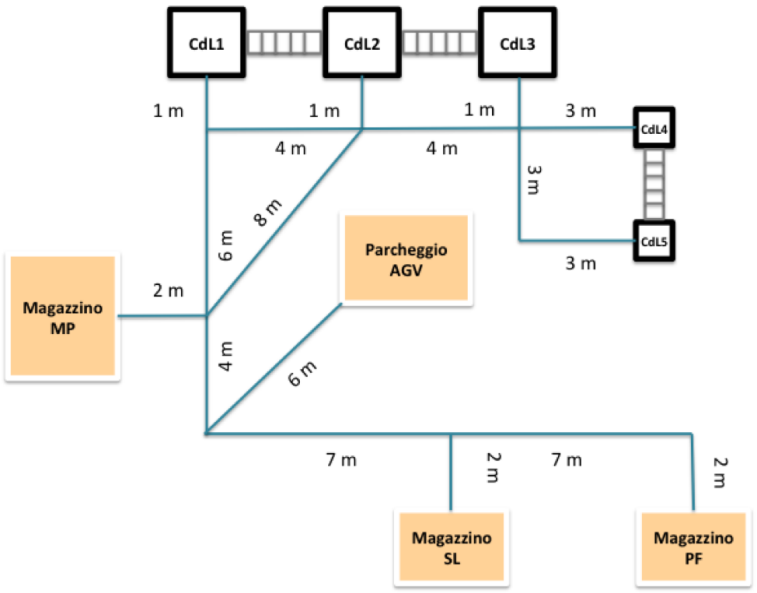


Figura 4.3: Impianto FMS preso in analisi.

- AGV detto *m2* la cui velocità è di 1,5 $[m/s]$.
- AGV detto *m3* la cui velocità è di 1,6 $[m/s]$.

Il tempo che gli AGV impiegano per effettuare un cambio mezzo di trasporto è mediamente 15 secondi, mentre quello per effettuare il carico/scarico dei pallet è pari a 20 secondi.

Ogni trasportatore si muove lungo il tracciato che collega tutte le zone dell'impianto, nello specifico caso proposto in Figura 4.3 si tratta del "collegamento" in blu.

Il caso preso in analisi pone il problema della necessità di avere in modo sincronizzato e veloce un AGV che possa servire un centro di lavoro che richiede un carico/scarico pallet. Alcuni AGV, poi, non potendo raggiungere tutte le zone dell'impianto, devono poter effettuare dei cambi mezzo di trasporto nelle zone adibite a far questo. Perciò la formulazione del modello funzionale calza perfettamente al caso, perché tiene proprio in considerazione la disponibilità dei mezzi e soprattutto la velocità di ognuno di essi.

4.2.1 Formalizzazione della Conoscenza: il Dominio Geometrico

Come si è detto non c'è ancora un collegamento diretto tra il tool e il VFF, ma il modello funzionale creato prende comunque in input i dati come se fossero formalizzati con il modello ontologico.

Ragionando in termini di ontologia, quindi, è necessario prima di tutto definire il Dominio Geometrico. I mezzi di trasporto AGV che vengono considerati sono a guida vincolata, perciò possono effettuare lo spostamento unicamente seguendo la banda magnetica posta a terra (le linee blu in Figura 4.3).

Nel caso specifico, quindi, i punti di interesse sono: (a) in prossimità dei centri di lavoro, le zone di carico/scarico pallet; (b) in prossimità dei magazzini o parcheggio l'ingresso; (c) punti di interesse generici come zone di diramazione del percorso; (d) zone dove è possibile effettuare il cambio mezzo di trasporto (munite di buffer).

4.2.2 Inizializzazione del Modello Funzionale

I punti di interesse definiti e strutturati con il modello formalizzato, sono stati specificati in 17 *macto-nodi* (cfr. Figura 4.4):

- *macro-nodo* 0: magazzino PF;
- *macro-nodo* 1: punto di interesse generico;
- *macro-nodo* 2: punto di interesse generico;
- *macro-nodo* 3: magazzino SL;
- *macro-nodo* 4: punto di interesse generico;
- *macro-nodo* 5: parcheggio AGV;
- *macro-nodo* 6: punto di interesse generico;

- *macro-nodo* 7: magazzino MP;
- *macro-nodo* 8: punto di interesse generico;
- *macro-nodo* 9: centro di lavoro CdL1;
- *macro-nodo* 10: zona cambio mezzo di trasporto;
- *macro-nodo* 11: centro di lavoro CdL2;
- *macro-nodo* 12: zona cambio mezzo di trasporto;
- *macro-nodo* 13: centro di lavoro CdL3;
- *macro-nodo* 14: centro di lavoro CdL4;
- *macro-nodo* 15: punto di interesse generico;
- *macro-nodo* 16: centro di lavoro CdL5;

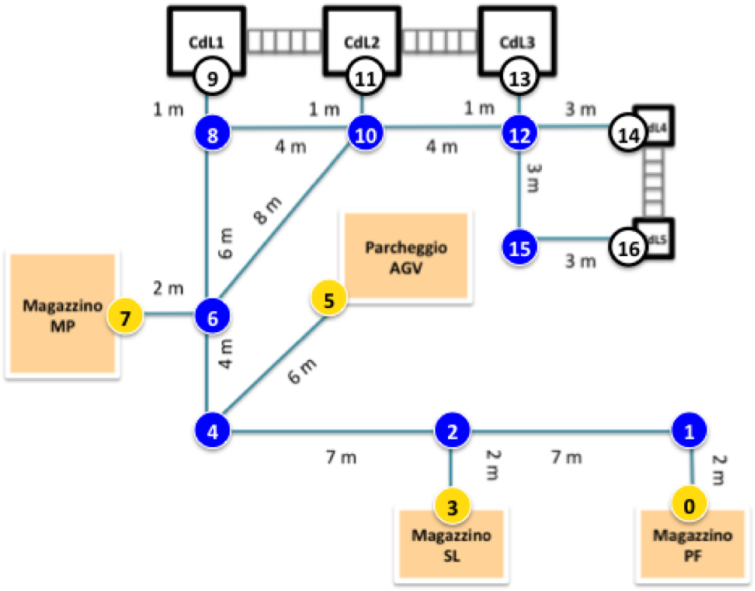


Figura 4.4: Assegnazione dei nodi ai punti di interesse.

L'AGV1, detto *m1*, è dedicato ai CdL1 e 2 e i suoi sotto-nodi "di competenza" appartengono ai macro-nodi {0, 1, 2, 3, 4, 5,

6, 7, 8, 9, 10, 11} e costruiscono il sottografo G1. L'AGV2 detto *m2*, invece, è dedicato ai CdL2 e 3 e i suoi sotto-nodi "di competenza" appartengono ai macro-nodi {0, 1, 2, 3, 4, 5, 6, 7, 10, 11, 12, 13} e costruiscono il sottografo G2. Mentre, l'AGV3 detto *m3*, serve i CdL 4 e 5 e i suoi sotto-nodi "di competenza" appartengono ai macro-nodi {0, 1, 2, 3, 4, 5, 6, 7, 10, 12, 14, 15, 16} che, a loro volta, definiscono il sottografo G3.

Per configurare il programma si è scelto di definire un sottografo, chiamato GZ, che considerasse tutti i possibili spostamenti all'interno dello stabilimento da parte di qualsiasi mezzo di trasporto. Così facendo, assegnando agli archi di GZ un nuovo peso, ossia lo **spazio percorso tra un nodo e l'altro**, è possibile verificare quale sia il mezzo più vicino al nodo di partenza.

Sulla base di tale informazione si sceglie il mezzo di trasporto che certamente sarà il più vicino e con il quale si effettuerà il primo spostamento. Inoltre, se dovesse essere necessario effettuare un cambio mezzo di trasporto, perché, per esempio, il nodo di arrivo non appartiene al sottografo del mezzo scelto inizialmente, il programma in C++ fornirà tale informazione.

4.2.3 Validazione del Modello

In Appendice A è possibile prendere visione del programma utilizzato per rispondere alle domande dello use case.

In Figura 4.5, è fornita la sovrapposizione dei tre sotto-grafi con configurazione "Non Planare". Gli archi in nero sono differenziati per indicare l'appartenenza al sotto-grafo, ossia al trasportatore di riferimento, mentre quelli in rosso riguardano il cambio mezzo di trasporto. I pesi assegnati a queste due tipologie di arco sono il tempo impiegato dal trasportatore per effettuare una movimentazione ([secondi]) e il tempo impiegato

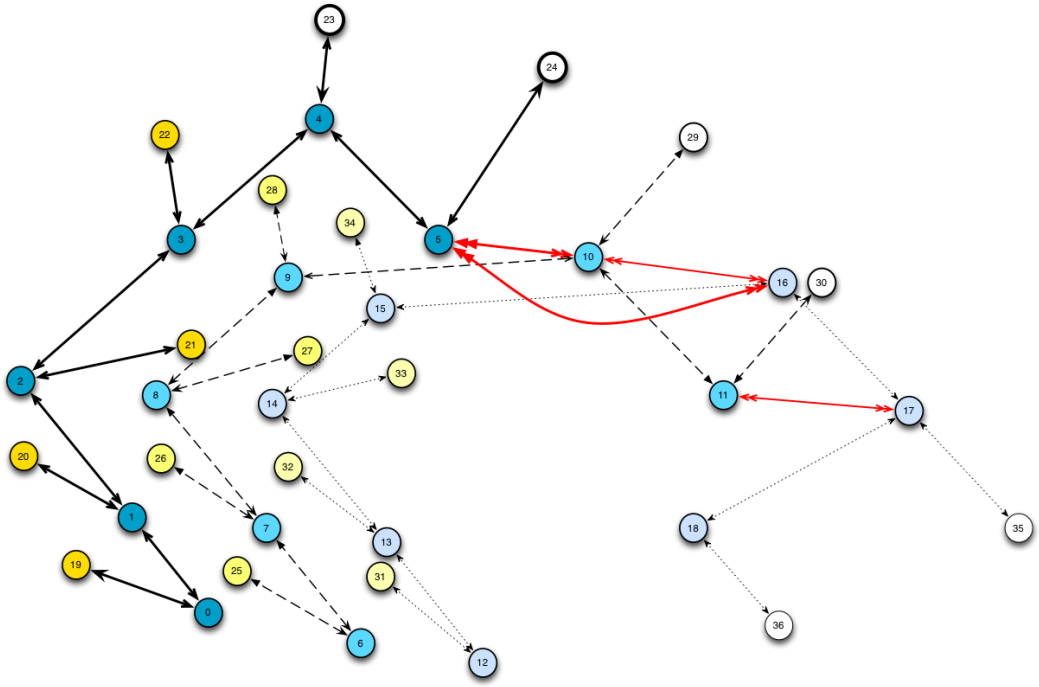


Figura 4.5: Assegnazione dei nodi ai punti di interesse dei trasportatori. Il primo livello (colori più scuri) è riferito al trasportatore m1, il livello intermedio al trasportatore m2, mentre l'utimo livello (colori più tenui) rappresenta il sottografo del trasportatore m3.

per effettuare il cambio mezzo ([secondi]), rispettivamente.

Dopo aver definito i nodi e gli archi, quindi, si è trascritto il file di testo che leggesse i nodi connessi e il relativo peso (cfr. Appendice A).

Per validare il modello si sono scelti dei casi specifici sicuramente tra i più interessanti, ossia:

Caso 1

Si vuole effettuare lo spostamento dal punto generico di interesse '6' alla "CdL4" (mezzi disponibili: m1, m2 ed m3).

Dato il programma in Appendice B, si ottiene:

Selezionare il nodo di ingresso tra :
0 1 2 3 4 5 6 7 8 9 10 11 12 13
Inserire il nodo scelto (0-13): **6**

Selezionare il nodo destinazione tra:
0 1 2 3 4 5 6 7 8 9 10 11 12 13
Inserire il nodo scelto (0-13): **14**

Il mezzo di trasporto 1 e' disponibile? yes [Y] - no [N] : **y**

Mezzo di trasporto 1 aggiunto

Inserire il nodo in cui si trova il trasportatore '1' :
0 1 2 3 4 5 6 7 8 9 10 11 12 13
Inserire il nodo scelto: **5**

Il mezzo di trasporto 2 e' disponibile? yes [Y] - no [N] : **y**

Mezzo di trasporto 2 aggiunto

Inserire il nodo in cui si trova il trasportatore '2' :
0 1 2 3 4 5 6 7 8 9 10 11 12 13

Inserire il nodo scelto: **7**

Il mezzo di trasporto 3 e' disponibile? yes [Y] - no [N] : **y**

Mezzo di trasporto 3 aggiunto

Inserire il nodo in cui si trova il trasportatore '3' :
0 1 2 3 4 5 6 7 8 9 10 11 12 13
Inserire il nodo scelto: **5**

Distanza dalla posizione del trasportatore '1' al nodo di ingresso: **10** metri.
Distanza dalla posizione del trasportatore '2' al nodo di ingresso: **2** metri.
Distanza dalla posizione del trasportatore '2' al nodo di ingresso: **10** metri.
Mezzo di trasporto più vicino al nodo di ingresso: **'2'**.
Spazio percorso dal mezzo più vicino: **2** metri.
Shortest path con il mezzo selezionato : **24** secondi.
Nodi attraversati: **18 19 20 26 7**
Lista mezzi di trasporto utilizzati: **2 2 2 3 3**
Numero nodi visitati: **5**
Numero cambi mezzo di trasporto effettuati: **1**
Mezzo di trasporto utilizzato per il primo spostamento è il mezzo: **' 2'**.
Il mezzo di trasporto utilizzato per l'ultimo spostamento è il mezzo: **'3'**

[Processo completato]

Quindi il mezzo di trasporto che compie il primo spostamento, ossia il trasportatore più vicino al nodo di ingresso, è il mezzo di trasporto 1. Il programma, partendo da questa informazione calcola il valore del cammino minimo, e indica quali zone sono

state attraversate.

Tempo di compilazione del programma: 10 secondi.

Caso 2

Si vuole effettuare lo spostamento dalla "CdL1" al magazzino "PF" (mezzi disponibili: m1, m2 ed m3).

Selezionare il nodo di ingresso tra :
0 1 2 3 4 5 6 7 8 9 10 11 12 13
Inserire il nodo scelto (0-13): **9**

Selezionare il nodo destinazione tra:
0 1 2 3 4 5 6 7 8 9 10 11 12 13
Inserire il nodo scelto (0-13): **7**

Il mezzo di trasporto 1 e' disponibile? yes [Y] - no [N] : **y**

Mezzo di trasporto 1 aggiunto

Inserire il nodo in cui si trova il trasportatore '1' :
0 1 2 3 4 5 6 7 8 9 10 11 12 13
Inserire il nodo scelto: **5**

Il mezzo di trasporto 2 e' disponibile? yes [Y] - no [N] : **y**

Mezzo di trasporto 2 aggiunto

Non è possibile raggiungere il nodo di ingresso con il mezzo di trasporto selezionato

Il mezzo di trasporto 3 e' disponibile? yes [Y] - no [N] : **y**

Mezzo di trasporto 3 aggiunto

Non è possibile raggiungere il nodo di ingresso con il mezzo di trasporto selezionato

Distanza dalla posizione del trasportatore '1' al nodo di ingresso: **17** metri.
Mezzo di trasporto più vicino al nodo di ingresso: **'1'**.
Spazio percorso dal mezzo più vicino: **17** metri.
Shortest path con il mezzo selezionato : **47** secondi.
Nodi attraversati: **4 13 12 3**
Lista mezzi di trasporto utilizzati: 1 1 1 1
Numero nodi visitati: **4**
Numero cambi mezzo di trasporto effettuati: **0**
Mezzo di trasporto utilizzato per il primo spostamento è il mezzo: **' 1'**.
Il mezzo di trasporto utilizzato per l'ultimo spostamento è il mezzo: **'1'**
[Processo completato]

Quindi il mezzo di trasporto che compie il primo spostamento, ossia il trasportatore più vicino al nodo di ingresso, è il mezzo di trasporto 1. Come si vede, sebbene i mezzi di trasporto 2 e 3 siano disponibili, il programma non li considera perché non sono in grado di effettuare il trasporto richiesto.

Tempo di compilazione del programma: 7 secondi.

Caso 3

Si vuole effettuare lo spostamento dalla "CdL3" al magazzino "SL"(si "accende" m1).

Selezionare il nodo di ingresso tra :
0 1 2 3 4 5 6 7 8 9 10 11 12 13
Inserire il nodo scelto (0-13): **13**

Selezionare il nodo destinazione tra:
0 1 2 3 4 5 6 7 8 9 10 11 12 13
Inserire il nodo scelto (0-13): **0**

Il mezzo di trasporto 1 e' disponibile? yes [Y] - no [N] : **y**

Mezzo di trasporto 1 aggiunto

Inserire il nodo in cui si trova il trasportatore '1' :
0 1 2 3 4 5 6 7 8 9 10 11 12 13
Inserire il nodo scelto: **5**

Non è possibile raggiungere il nodo di ingresso con il mezzo di trasporto selezionato

Il mezzo di trasporto 2 e' disponibile? yes [Y] - no [N] : **n**

Mezzo di trasporto 2 NON aggiunto

Il mezzo di trasporto 3 e' disponibile? yes [Y] - no [N] : **n**

Mezzo di trasporto 3 NON aggiunto

[Processo completato]

Quindi non è possibile raggiungere il nodo di ingresso selezionato con nessuno dei mezzi di trasporto disponibili.

Tempo di compilazione del programma: 6 secondi.

In Appendice A sono disponibili altri esempi applicativi.

Come volevasi dimostrare, attraverso un modello formalizzato e uno funzionale è stato possibile rispondere concretamente alle domande poste all'inizio della tesi, ossia:

- Trasportatori disponibili per la movimentazione
- Raggiungibilità di un punto di interesse
- Calcolo dei tempi di movimentazione
- Identificazione del mezzo più vicino al punto di interesse dal quale parte il trasportatore
- Numero e identità dei trasportatori coinvolti in una movimentazione
- Zone occupate dal trasportatore (o dai trasportatori) durante lo spostamento

Inoltre, i tempi di compilazione del programma (in tutti i casi testati, sebbene non riportati in questa tesi) non superano mai i 20 secondi.

Quindi, è possibile assodare che, non solo si è giunti all'obiettivo di risposta alle domande specifiche, ma è stato identificato uno strumento molto veloce per fare ciò.

Capitolo 5

Applicazione al Caso Reale per la Validazione del Modello: Impianto Pomini

5.1 L'azienda Pomini

L'azienda è stata fondata da Luigi Pomini nel 1886. Nel 1984 acquisisce la società GIUSTINA, divisione *rettifiche per cilindri*, spostando nella sede centrale di Castellanza il personale tecnico strategico, il know-how e il materiale tecnico. Nel 1988 la Pomini entra nel gruppo Techint e nel 2008 diventa Tenova Pomini Spa.

Attraverso il marchio Pomini, Tenova è leader a livello mondiale nella progettazione e fornitura di rettifiche per cilindri per laminatoi di prodotti piani e rettificatrici speciali per componenti pesanti. La gamma dei prodotti Tenova Pomini include rettificatrici per cilindri a controllo numerico interamente automatizzate che trovano impiego in applicazioni pesanti, medie e leggere, ma anche caricatori di cilindri a controllo numerico completamente automatizzati.

Fornisce anche supporti per stoccaggio cilindri, inseritori ed estrattori di guarniture per tutti i tipi di cilindri, sviluppati per velocizzare le operazioni di rettifica, movimentazione e manutenzione, oltre che per garantire il livello di precisione richiesto dai laminatoi più sofisticati.

Tenova Pomini fornisce inoltre numerose apparecchiature ausiliarie, come impianti di lavaggio e ribaltatori per guarniture, sistemi di raffreddamento per cilindri, nonché diversi altri dispositivi impiegati nelle operazioni di roll shop giornaliere.

Tenova Pomini vanta inoltre competenze consolidate nel ricondizionamento, aggiornamento e modernizzazione completamente automatizzata di rettificatrici per cilindri usate di qualsiasi marca. La ricerca costante di soluzioni innovative per l'automazione dei laminatoi e l'integrazione delle macchine,

insieme alla precisione e all'affidabilità che hanno sempre contraddistinto il marchio Tenova Pomini, consente di offrire ai clienti prodotti all'avanguardia nel panorama internazionale del settore.

5.2 La Laminazione

L'ambito in cui si inserisce questo problema industriale è la laminazione e, nello specifico, la rettifica dei rulli usati per effettuare questa trasformazione.

La laminazione è uno dei processi industriali di deformazione massiva in cui i componenti realizzati hanno un rapporto limitato tra superficie e volume. In particolare, tale processo consiste nella riduzione dell'altezza (quindi un cambio di sezione) di un pezzo attraverso la pressione applicata da due rulli rotanti. In base alla temperatura T di lavoro, i processi di formatura per deformazione plastica si distinguono in processi a:

- freddo ($T/T_{fusione} \leq 0.3$)
- tiepido ($0.3 < T/T_{fusione} < 0.6$)
- caldo ($T/T_{fusione} \geq 0.6$)

Inoltre, in riferimento al risultato, si possono classificare le lavorazioni in:

- primarie: permettono di ottenere dei semilavorati sbozzati in varie forme;
- secondarie: a partire dalle preforme ottenute nelle lavorazioni primarie permettono di ottenere un prodotto semifinito o finito.

La laminazione è tipicamente la prima trasformazione a valle di quelle siderurgiche: ad esempio, il lingotto e la barra generata da colata continua vengono trasformati mediante laminazione.

Nel caso più semplice, la laminazione viene eseguita tramite **rulli cilindrici** non sagomati, caso tipico della produzione di lamiere. In altri casi i **rulli** sono **sagomati** in modo da imporre una particolare geometria alla sezione in uscita.

I principali prodotti finiti sono barre e piatti, travi, profilati speciali, rotaie, lamiere, nastri larghi e stretti e vergelle (si veda lo schema generale in Figura 5.1) [27].

Gli impianti di laminazione possono essere:

i laminatoio **"duo"**: il laminato deve entrare nella gabbia di laminazione dallo stesso lato e l'impianto ha una capacità produttiva limitata. I rulli che vengono usati sono due *work rolls*.

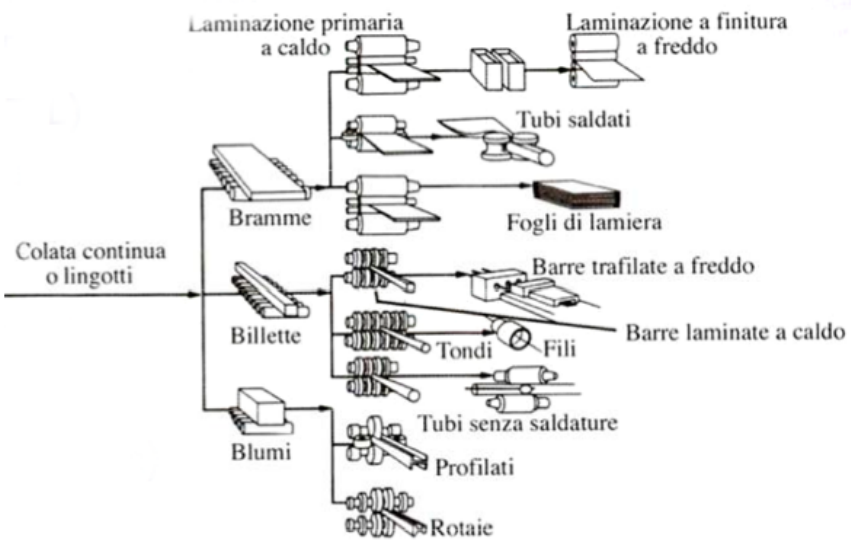


Figura 5.1: Schema generale per il processo di laminazione. [26]

ii laminatoio **"trio"**: la lamiera passa alternativamente (o in continuo) tra le due luci. Così si ha una maggiore capacità produttiva rispetto alla gabbia a due rulli.

iii laminatoio **"quarto"**: i rulli di piccolo diametro sono a contatto con la lamiera e sono motorizzati. Come si vede in Figura fig:quarti, il compito di laminare è affidata alla coppia di cilindri di lavoro (*work rolls*), i quali sono di piccolo diametro; mentre il compito di supportare il carico di laminazione è affidata alla coppia di cilindri di sostegno (*back-up rolls*) i quali presentano diametri maggiori. I rulli di grande diametro hanno, quindi, la sola funzione di irrigidimento. Questa tipologia viene largamente utilizzata nella lavorazione delle lamiere.

iv laminatoio **"multicilindro"** o **"Sendzimir"**: la sollecitazione viene distribuita su cilindri di diametro crescente ((*work rolls* all'interno, in mezzo gli *intermediate rolls* e i più esterni *back-up rolls*). I cilindri motorizzati sono quelli intermedi.

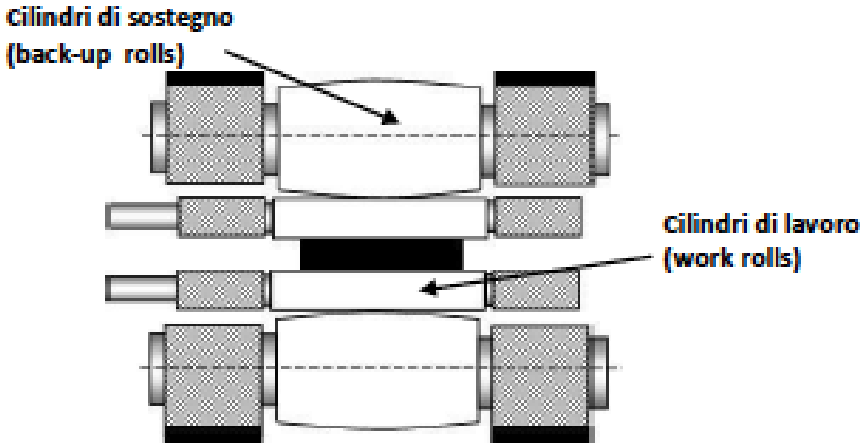


Figura 5.2: Schema del laminatoio "quarto"

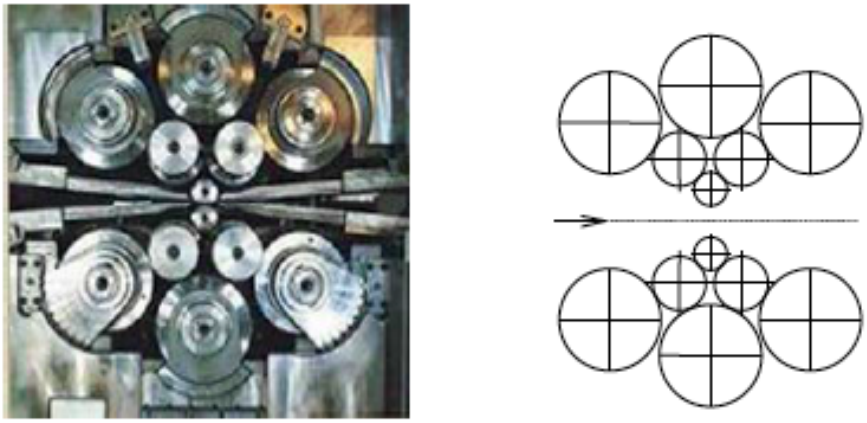


Figura 5.3: Laminatoio multicilindro. [27]

5.3 Il Problema Industriale

La parte su cui ci si è focalizzati in questa tesi è il sistema completamente automatizzato di movimentazione dei cilindri nella parte di stabilimento dedicata alla rettifica che Tenova Pomini Spa fornisce ai propri clienti con la formula "chiavi in mano". Come anticipato, infatti, l'azienda, oltre a progettare e fornire le rettifiche per cilindri (Figura 5.4), offre ai propri clienti anche gli impianti comprensivi di macchine per la rettifica e di tutto l'impianto di movimentazione.

Figura 5.4: Un esempio di macchina Tenova Pomini

In generale, la rettifica è la procedura eseguita con una macchina chiamata rettificatrice e che ha come utensile una mola a grana fine ed estremamente dura. Durante la lavorazione, infatti, i cilindri si usurano e quindi aumenta il loro diametro e non sono più nelle condizioni di tenere la compressione. La rettifica riporta al livello di finitura definito da progetto la superficie del cilindro. Si tratta della fase che segue la sgrossatura. Mentre la sgrossatura toglie il grosso dei residui, la rettifica fa sì che tutti i residui o il materiale in eccesso vengano eliminati garantendo alla superficie lavorata un alto grado di finitura.

Sebbene si tratti di un'operazione costosa, a causa di una veloce usura dei cilindri (roll) è necessario ricorrere a rettifica molto spesso. Nello specifico caso industriale trattato in questa tesi:

- i *back-up rolls* devono essere rettificati ogni 20/30 giorni
- i *work rolls* di laminatoi a caldo nell'ordine delle ore
- i *work rolls* di laminatoi a freddo richiedono la rettifica ogni 20 minuti circa

Risulta, quindi, molto interessante testare quanto creato in questo elaborato su questo specifico contesto industriale. Infatti all'interno di questi impianti di rettifica è fondamentale la gestione dei trasporti sincronizzata sugli step di lavorazione.

Si è quindi scelto una delle possibili strutture dell'impianto di rettifica fornito dall'azienda e si è schematizzato come illustrato in Figura 5.5.

Tale impianto (molto simile a quello in Figura 5.6) presenta:

- tre zone di "parcheggio" per i cilindri in attesa di rettifica o di spostamento verso l'impianto di laminazione;
- tre ingressi in cui arrivano le navette (una dedicata ai WORK (work rolls) e due dedicate ai cosiddetti BUR (back-up rolls)) che spostano i

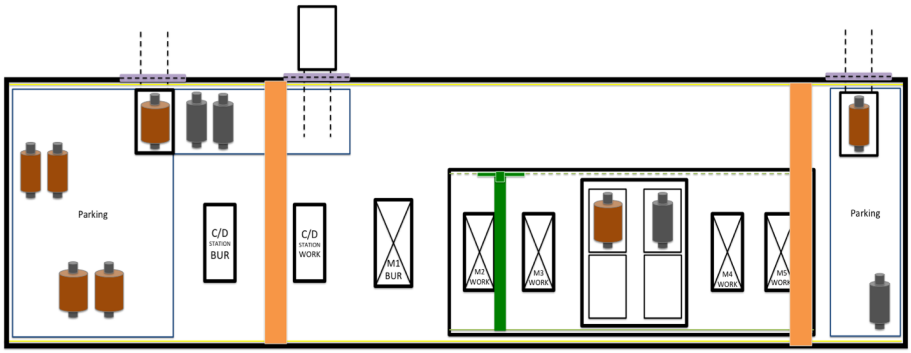


Figura 5.5: Schema per un impianto fornito da Tenova Pomini

cilindri da rettificare o rettificati dall'impianto produttivo all'impianto di rettifica;

- una macchina per la rettifica dei BUR, una chocking/de-chocking station per i BUR e una per i WORK;
- quattro macchine per la rettifica dei WORK;
- una *exchange* che funge da zona franca in cui il carroponte scarica i WORK che possono essere portati alle macchine (WORK) solo dalla gru zoppa. In questa zona vengono portati da uno dei due carriponte i cilindri WORK da rettificare, o viceversa vengono caricati per essere spostati dalla zona *exchange* una volta terminata la lavorazione. Ciò implica che nessun mezzo a parte la gru zoppa, può accedere direttamente alle macchine "work".

Ricapitolando, i mezzi di trasporto che sono stati presi in considerazione in questo caso di studio sono:

- due carriponte: uno dedicato al trasporto dei WORK (perché meno potente) la cui velocità è di 0,8 $[m/s]$, e l'altro che può trasportare anche gli ingombranti e pesanti BUR (velocità 0,7 $[m/s]$). Il tempo di scarico e carico è per entrambi 90 [secondi].
- una gru zoppa che si muove solo dalla zona *exchange* alle macchine WORK. La sua velocità è di 1 $[m/s]$.

5.3.1 Formalizzazione della Conoscenza: il Dominio Geometrico

Ragionando in termini di ontologia è necessario prima di tutto definire il Dominio Geometrico.

Figura 5.6: Un impianto di rettifica fornito da Tenova Pomini

Dovendo avere a che fare con mezzi di trasporto che hanno movimenti verticali di sollevamento (il gancio del carroponte per esempio), movimenti trasversali (dello stesso gancio per posizionarsi sopra il punto di interesse) e movimenti longitudinali, ossia lungo tutta la lunghezza dell'impianto, i punti di interesse da prendere in considerazione sono estremamente numerosi (Figura 5.7) e, in ottica di configurazione con i grafi, i nodi risulterebbero in numero infinito.

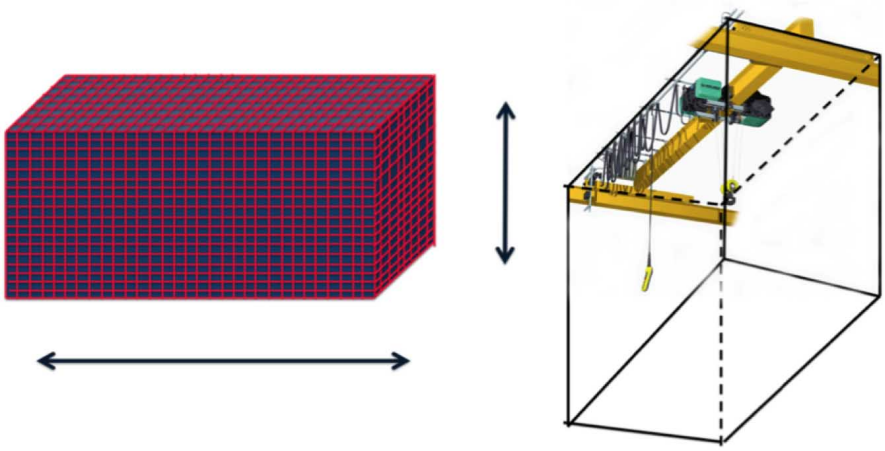


Figura 5.7: Dominio Geometrico Pomini

Per questo motivo si è ipotizzato che:

Hp. i siano trascurabili i tutti punti raggiungibili attraverso gli spostamenti verticali

Hp. ii siano trascurabili tutti i punti raggiungibili attraverso gli spostamenti trasversali

Hp. iii siano considerati soltanto i punti di interesse localizzati lungo l'asse longitudinale

5.3.2 Inizializzazione del Modello Funzionale

Si è scelto, per questi motivi, di definire come punti di interesse tutti gli elementi elencati precedentemente. Quindi si avranno 14 *macro-nodi* (Figura 5.9), e nello specifico il:

- *macro-nodo* 0: parcheggio denominato P1;
- *macro-nodo* 1: navetta N1 per i BUR;
- *macro-nodo* 2: parcheggio denominato P2;
- *macro-nodo* 3: chocking/de-chocking station per i BUR;
- *macro-nodo* 4: navetta N2 per i BUR;
- *macro-nodo* 5: chocking/de-chocking station per i WORK;
- *macro-nodo* 6: macchina (M0) per la rettifica dei BUR;
- *macro-nodo* 7: macchina (M1) per la rettifica dei WORK;
- *macro-nodo* 8: macchina (M2) per la rettifica dei WORK;
- *macro-nodo* 9: zona exchange;
- *macro-nodo* 10: macchina (M3) per la rettifica dei WORK;
- *macro-nodo* 11: macchina (M4) per la rettifica dei WORK;
- *macro-nodo* 12: navetta N3 per i WORK;
- *macro-nodo* 13: parcheggio denominato P3 per i WORK;

Come già detto, il carroponte che chiameremo **m1** può movimentare i BUR e i WORK, quindi può effettuare una movimentazione dei cilindri tra i nodi: {0, 1, 2, 3, 4, 5, 6, 9}, il sotto-grafo che si riferisce a *m1* è G1. Il carroponte dedicato solo ai WORK **m2**, invece, può caricare e scaricare i cilindri nei nodi: {2, 5, 9, 12, 13}, il sotto-grafo che si riferisce a *m2* è G2. La gru zoppa **m3** ha accesso solo ai nodi {7,8,9,10,11} e il sotto-grafo riferito ad essa è G3. I vincoli inerenti ai nodi, sono stati definiti sulla base di un diagramma di flusso (Figura 5.8) che esplicita tutti gli step che un cilindro deve (o può) effettuare da quando entra nell'impianto di rettifica a quando esce.

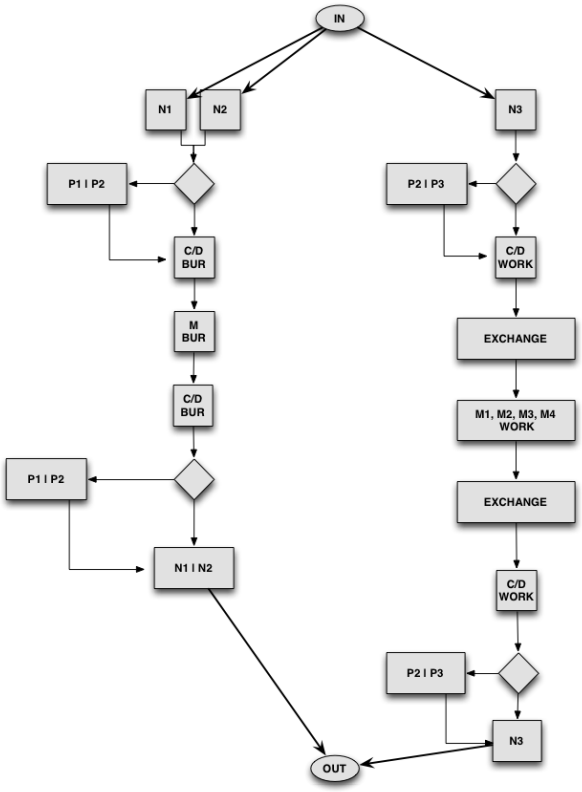


Figura 5.8: Diagramma di flusso per i cilindri. A sinistra per quelli BUR, a destra per i WORK

Sebbene ci siano queste limitazioni di carico/scarico, il carroponte più potente può spostarsi lungo tutta la lunghezza dello stabilimento, ma non può arrivare ai nodi 12 e 13 per via dell'ingombro del secondo carroponte. Vale lo stesso per quest'ultimo che a sua volta non può raggiungere il nodo 0. Come già detto la gru zoppa può muoversi unicamente all'interno della zona recintata.

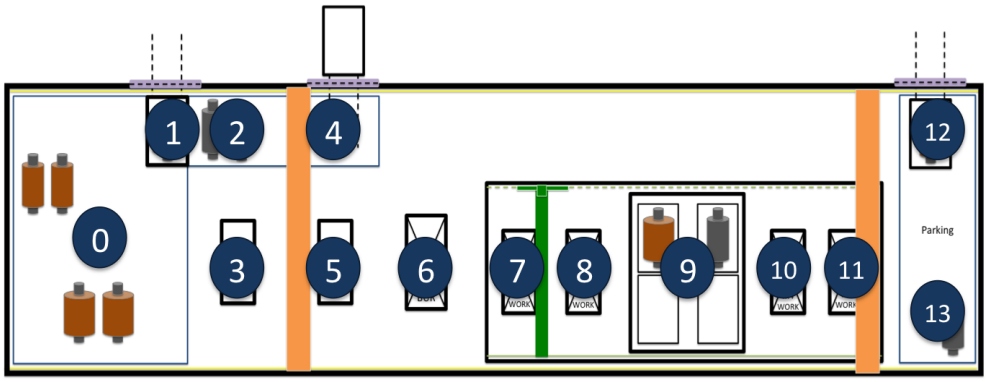


Figura 5.9: Nodi associati ai punti di interesse dell'impianto di rettifica Tenova Pomini

Per configurare il programma si è scelto di definire un sottografo, chiamato GZ, che considerasse tutti i possibili spostamenti all'interno dello stabilimento da parte di qualsiasi mezzo di trasporto. Così facendo, assegnando agli archi un nuovo peso, ossia lo **spazio percorso tra un nodo e l'altro**, è possibile verificare quale sia il mezzo più vicino al nodo di carico del cilindro.

Sulla base di tale informazione si sceglie il mezzo di trasporto che certamente sarà il più vicino e con il quale si effettuerà il primo spostamento. Inoltre, se dovesse essere necessario effettuare un cambio mezzo di trasporto, perché, per esempio, il nodo di arrivo non appartiene al sottografo del mezzo scelto inizialmente, il programma in C++ fornirà tale informazione.

5.3.3 Validazione del Modello

In Appendice B è possibile prendere visione del programma utilizzato per rispondere alle domande dello use case.

In Figura 5.10 si riporta la configurazione "Non Planare" che considera tutti i *sotto-nodi* relativi ai trasportatori presenti nell'impianto. Agli archi viene assegnato un peso in [secondi], per quelli di colore nero si fa riferimento al tempo di spostamento del singolo trasportatore (distinti nel disegno sulla base del tipo di tratto usato per le frecce), mentre per quelli in rosso ci si riferisce al tempo impiegato per effettuare il cambio mezzo, ossia il tempo di scarico e di carico dei due trasportatori coinvolti ([secondi]).

Dopo aver definito i nodi, si è trascritto il file di testo che leggesse i nodi connessi e il relativo peso (Appendice B).

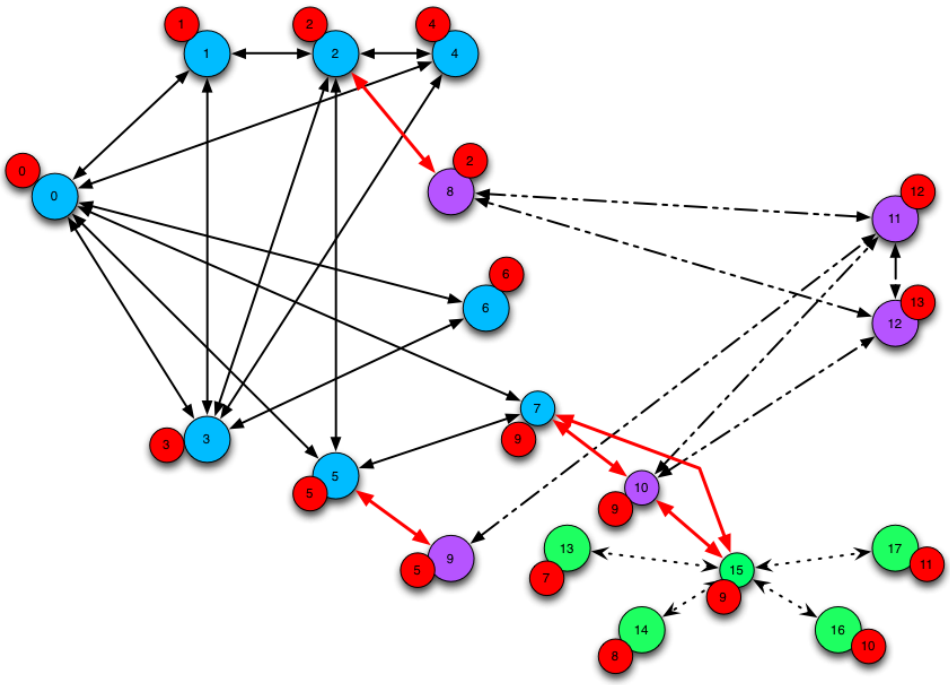


Figura 5.10: Rappresentazione con i grafi per gli archi dei *macro-nodi* e relativi *sotto-nodi*. In celeste i nodi riferiti al carroponte m1, in viola quelli del carroponte m2 e in verde i nodi della gru zoppa (m3).

Per validare il modello si sono scelti dei casi specifici sicuramente tra i più interessanti, ossia:

Caso 1

Si vuole caricare un cilindro dalla "C/D station" dei WORK e scaricarlo nella "exchange" con tutti e tre i mezzi disponibili (macchine disponibili: m1, m2 ed m3).

Dato il programma in Appendice B, si ottiene:

Selezionare il nodo di ingresso tra :
0 1 2 3 4 5 6 7 8 9 10 11 12 13
Inserire il nodo scelto (0-13): **5**

Selezionare il nodo destinazione tra:
0 1 2 3 4 5 6 7 8 9 10 11 12 13
Inserire il nodo scelto (0-13): **9**

Il mezzo di trasporto 1 e' disponibile? yes [Y] - no [N] : **y**

Mezzo di trasporto 1 aggiunto

Inserire il nodo in cui si trova il trasportatore '1' :
0 1 2 3 4 5 6 7 8 9 10 11 12 13
Inserire il nodo scelto: **1**

Il mezzo di trasporto 2 e' disponibile? yes [Y] - no [N] : **y**

Mezzo di trasporto 2 aggiunto

Inserire il nodo in cui si trova il trasportatore '2' :
2 3 4 5 6 7 8 9 10 11 12 13
Inserire il nodo scelto: **6**

Il mezzo di trasporto 3 e' disponibile? yes [Y] - no [N] : **y**

Mezzo di trasporto 3 aggiunto

Inserire il nodo in cui si trova il trasportatore '3' :
7 8 9 10 11
Inserire il nodo scelto: **7**

Non è possibile raggiungere il nodo di ingresso con il mezzo di trasporto selezionato

Distanza dalla posizione del trasportatore '1' al nodo di ingresso: **50** metri.
Distanza dalla posizione del trasportatore '2' al nodo di ingresso: **40** metri.
Mezzo di trasporto più vicino al nodo di ingresso: **'2'**.
Spazio percorso dal mezzo più vicino: **40** metri.
Shortest path con il mezzo selezionato : **113** secondi.
Nodi attraversati: **9 10**
Numero nodi visitati: **2**
Numero cambi mezzo di trasporto effettuati: **0**
Mezzo di trasporto utilizzato per il primo spostamento è il mezzo: **' 2'**.
Il mezzo di trasporto utilizzato per l'ultimo spostamento è il mezzo: **'2'**
[Processo completato]

Quindi il mezzo di trasporto che compie il primo spostamento, ossia il trasportatore più vicino al nodo di ingresso, è il mezzo di trasporto 2. Il programma, partendo da questa informazione calcola il valore del cammino minimo, e indica quali zone sono state attraversate: la 9 e la 10, senza mai

dover cambiare mezzo di trasporto.

Tempo di compilazione del programma: 7 secondi.

Caso 2

Si vuole caricare un cilindro dal "P2" parcheggio accessibile sia al carroponte m1 sia al m2, e scaricarlo nella macchina "M1", con tutti e tre i mezzi disponibili (si "accende" m1, m2 ed m3).

Selezionare il nodo di ingresso tra :
0 1 2 3 4 5 6 7 8 9 10 11 12 13
Inserire il nodo scelto (0-13): **2**

Selezionare il nodo destinazione tra:
0 1 2 3 4 5 6 7 8 9 10 11 12 13
Inserire il nodo scelto (0-13): **7**

Il mezzo di trasporto 1 e' disponibile? yes [Y] - no [N] : **y**

Mezzo di trasporto 1 aggiunto

Inserire il nodo in cui si trova il trasportatore '1' :
0 1 2 3 4 5 6 7 8 9 10 11 12 13
Inserire il nodo scelto: **0**

Il mezzo di trasporto 2 e' disponibile? yes [Y] - no [N] : **y**

Mezzo di trasporto 2 aggiunto

Inserire il nodo in cui si trova il trasportatore '2' :
1 2 3 4 5 6 7 8 9 10 11 12 13
Inserire il nodo scelto: **2**

Non è possibile raggiungere il nodo di ingresso con il mezzo di trasporto selezionato

Il mezzo di trasporto 3 e' disponibile? yes [Y] - no [N] : **y**

Mezzo di trasporto 3 aggiunto

Inserire il nodo in cui si trova il trasportatore '3' :
7 8 9 10 11 Inserire il nodo scelto: **8**

Non è possibile raggiungere il nodo di ingresso con il mezzo di trasporto selezionato

Distanza dalla posizione del trasportatore '1' al nodo di ingresso: **58** metri.
Distanza dalla posizione del trasportatore '2' al nodo di ingresso: **0** metri.
Mezzo di trasporto più vicino al nodo di ingresso: **'2'**.
Spazio percorso dal mezzo più vicino: **0** metri.
Shortest path con il mezzo selezionato : **371** secondi.
Nodi attraversati: **8 10 15 13**
Numero nodi visitati: **4**
Lista mezzi di trasporto utilizzati: **2 2 3 3**
Numero cambi mezzo di trasporto effettuati: **1**
Mezzo di trasporto utilizzato per il primo spostamento è il mezzo: **'2'**.
Il mezzo di trasporto utilizzato per l'ultimo spostamento è il mezzo: **'3'**
[Processo completato]

Quindi il mezzo di trasporto che compie il primo spostamento, ossia il trasportatore più vicino al nodo di ingresso, è il mezzo di trasporto 2. Il programma, partendo da questa informazione calcola il valore del cammino minimo, e indica quali zone sono state attraversate: la 8, 10, 15, 13. Come si nota è stato effettuato un cambio mezzo di trasporto nel "passaggio" da 10 a 15, quindi dal mezzo di trasporto 2 al mezzo 3. Tempo di compilazione del programma: 4 secondi.

Caso 3

Si vuole caricare un cilindro dalla navetta "N1" dedicata ai BUR, e scaricarlo nella "C/D station" dei WORK, "accendendo" (m2 ed m3).

Selezionare il nodo di ingresso tra :
0 1 2 3 4 5 6 7 8 9 10 11 12 13
Inserire il nodo scelto (0-13): **1**

Selezionare il nodo destinazione tra:
0 1 2 3 4 5 6 7 8 9 10 11 12 13
Inserire il nodo scelto (0-13): **5**

Il mezzo di trasporto 1 e' disponibile? yes [Y] - no [N] : **n**

Mezzo di trasporto 1 NON aggiunto

Il mezzo di trasporto 2 e' disponibile? yes [Y] - no [N] : **y**

Mezzo di trasporto 2 aggiunto

Inserire il nodo in cui si trova il trasportatore '2' :
1 2 3 4 5 6 7 8 9 10 11 12 13
Inserire il nodo scelto: **4**

Non è possibile raggiungere il nodo di ingresso con il mezzo di trasporto selezionato

Il mezzo di trasporto 3 e' disponibile? yes [Y] - no [N] : **y**

Mezzo di trasporto 3 aggiunto

Inserire il nodo in cui si trova il trasportatore '3' :
7 8 9 10 11
Inserire il nodo scelto: **11**

Non è possibile raggiungere il nodo di ingresso con il mezzo di trasporto selezionato

[Processo completato]

Quindi non è possibile raggiungere il nodo di ingresso selezionato con nessuno dei mezzi di trasporto disponibili.

Tempo di compilazione del programma: 2 secondi.

Quelli elencati precedentemente sono solo alcuni dei casi di validazione effettuati. In Appendice B si riporta lo screenshot del terminale per altri casi valutati (ma non tutti).

Si è così dimostrato ancora che, attraverso un modello formalizzato e uno funzionale è stato possibile rispondere concretamente alle domande poste all'inizio della tesi, ossia:

- Trasportatori disponibili per la movimentazione

- Raggiungibilità di un punto di interesse
- Calcolo dei tempi di movimentazione
- Identificazione del mezzo più vicino al punto di interesse dal quale parte il trasportatore
- Numero e identità dei trasportatori coinvolti in una movimentazione
- Zone occupate dal trasportatore (o dai trasportatori) durante lo spostamento

Anche in questo caso i tempi di compilazione del programma (in tutti i casi testati, sebbene non riportati in questa tesi) non superano mai i 20 secondi.

Quindi, è possibile ribadire che, non solo si è giunti all'obiettivo di risposta alle domande specifiche, ma si tratta di uno strumento molto veloce.

Conclusioni

In questo lavoro di tesi si è voluto prendere in esame uno strumento come il Virtual Factory Framework che è frutto di un Progetto Europeo, appena conclusosi, coordinato dall'ITIA-CNR in collaborazione con numerosi partner europei d'eccellenza.

Il VFF implementa la struttura per un ambiente virtuale collaborativo e *object-oriented*, grazie alla rappresentazione delle informazioni attraverso un modello ontologico (il Virtual Factory Data Model) che usa differenti standard tecnici per definire gli oggetti, gli elementi e le relazioni che entrano in gioco in un contesto industriale, in maniera univoca e condivisibile.

In particolare in questo elaborato è stato analizzato tale progetto partendo da un problema preciso: la formalizzazione della conoscenza sulla base del modello VFDM a supporto della valutazione delle prestazioni di un sistema produttivo e, in particolare, in relazione ai sistemi di trasporto interno alla fabbrica.

Questa tematica è stata scelta poiché, pur essendo un problema noto e molto importante, l'effetto che i mezzi di trasporto interno hanno sulle performance dei sistemi produttivi spesso è tralasciato per via della numerosità di vincoli ed elementi da tenere in considerazione. Tale decisione è stata avvalorata dal fatto che nel VFDM non è ancora stata strutturata questa parte. Infatti il VFDM, pur tenendo in considerazione i trasporti come sottoclasse di processo, limita le caratteristiche associate ad essi al solo tempo impiegato per effettuare un dato processo di trasporto.

Quindi, l'intento della tesi ha assunto una doppia interpre-

tazione: la volontà di dimostrare che l'utilizzo del VFF fosse una via concretamente percorribile, e il tentativo di definire un "concetto" espresso in modo non del tutto completo.

Questo elaborato di tesi è stato sviluppato in più fasi. La prima è stata dedicata allo studio del modello Virtual Factory Data Model, sul quale ci si è poi ispirati per la formalizzazione della conoscenza "mancante". Successivamente, attraverso l'utilizzo di questo modello ontologico, è stata creata la classe relativa ai trasporti interni messa in relazione agli step che compongono un processo produttivo. Ogni step è eseguito su macchine diverse, perciò al fine di permettere lo spostamento da uno all'altro è necessario un trasportatore. La classe creata formalizza tale trasporto, e a questa vengono assegnate le risorse di trasporto sulla base degli step di processo a monte e a valle.

I trasportatori assegnati al tipo risorsa richiesto, sono quelli in grado di servire almeno una delle macchine appartenenti agli step a monte e agli step a valle del processo di trasporto. Ciò implica che non tutti i trasportatori siano in grado soddisfare tutte le operazioni di trasporto possibili; nasce perciò il problema di fattibilità di utilizzo del trasportatore assegnato. Per questo motivo è stato definita a priori e in maniera formale tale *fattibilità*, e per questo si è creato un *Dominio Geometrico* del singolo mezzo di trasporto che definisse illuogo dei punti che può essere servito dal trasportatore.

Le informazioni relative ai trasporti interni alla fabbrica influenzano le performance di sistema, perciò il passo successivo dello studio condotto, è stato quello di estrapolare la conoscenza così formalizzata per manipolarla in fase di valutazione delle prestazioni di sistema.

Per fare ciò esistono molti strumenti, l'ontologia stessa sfrutta un ragionatore per le inferenze riguardanti proposizioni logiche del primo ordine. Però, per supportare concretamente la valuta-

zione delle prestazioni in termini di trasporti interni all'azienda, è stato necessario rintracciare uno strumento per rispondere a domande specifiche basate sulla ricerca del cammino minimo, ossia sulla ricerca del mezzo di trasporto che, se in grado di raggiungere una specifica macchina a monte e una a valle, effettui lo spostamento nel minor tempo possibile.

Le domande poste sono:

- Trasportatori disponibili per la movimentazione
- Raggiungibilità di un punto di interesse
- Calcolo dei tempi di movimentazione
- Identificazione del mezzo più vicino al punto di interesse dal quale parte il trasportatore
- Numero e identità dei trasportatori coinvolti in una movimentazione
- Zone occupate dal trasportatore (o dai trasportatori) durante lo spostamento

E' stato creato un modello funzionale per rispondere concretamente a tali domande, che sovrappone un ulteriore livello di formalizzazione al modello ontologico precedentemente creato, ossia associa i grafi all'ontologia. Nello specifico, sono stati creati tanti grafi quanti sono i trasportatori, e per ognuno di essi: (1) si è rappresentato con un **nodo** ogni *punto di interesse raggiungibile* appartenente al Dominio Geometrico; (2) per ogni coppia di nodi connessi, è stato creato un **arco** che li unisse; (3) per ogni arco è stato associato un **peso** (tempo di movimentazione da un nodo all'altro oppure spazio percorso).

Questi grafi sono diventati input del programma creato per rispondere alle specifiche domande. Il linguaggio utilizzato è C++ che, sfruttando librerie dedicate già esistenti e nello specifico l'algoritmo di Bellman-Ford, è capace di valutare se un punto

di interesse sia raggiungibile o meno dai trasportatori "accesi" grazie all'utilizzo dei sottografi relativi al singolo trasportatore. Sulla base di ciò, permette anche di calcolare i tempi di movimentazione, da un nodo di partenza a un nodo di arrivo, per tutte le possibili combinazioni di trasporto (disponibili), indicando il cammino minimo. Infine, facendo riferimento alle zone occupate dal o dai trasportatori per compiere tale cammino minimo, identifica il numero e l'identità dei trasportatori coinvolti nella movimentazione.

Il programma, infine, è stato validato attraverso due casi applicativi, il primo "realistico" riguarda un problema industriale per un impianto FMS, mentre il secondo è relativo a un caso reale: l'impianto di rettifica Tenova Pomini.

La validazione ha permesso di affermare che, prendendo in input i dati strutturati come specificato precedentemente, è possibile restituire informazioni relative ai trasporti interni alla fabbrica a supporto di utenti o metodi analitici che si trovano a dover valutare le prestazioni del sistema produttivo.

Il limite di questo lavoro risiede nel fatto che non è stato creato il collegamento tra l'ontologia e lo strumento. Di certo, però, si può dire che, con la formalizzazione proposta in questo elaborato, è stato definito un punto di partenza concreto, un tracciato percorribile e validato verso l'utilizzo del Virtual Factory Framework come ambiente integrato di strumenti a supporto della valutazione delle prestazioni.

Lista degli acronimi

CPDM	Collaborative Product Development and Management
FMS	Flexible Manufacturing Systems
IFC	Industry Foundation Classes
MH	Material Handling - Movimentazione dei materiali
OWL	Web Ontology Language
PLM	Product Lifecycle Management
u.d.c	Unità di carico
VF	Virtual Factory
VFDM	Virtual Factory Data Model
VFF	Virtual Factory Framework (Project)
VFM	Virtual Factory Manager
VFm	Virtual Factory modules

Appendice A

Programma in Linguaggio C++ per il Caso Realistico: Impianto FMS

Listing A.1: Programma C++ per il Caso Reale Tenova Pomini

```
#include <boost/config.hpp>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <vector>
#include <vector.h>
#include <iomanip>
#include <utility>
#include <cmath>
#include <boost/graph/subgraph.hpp>
#include <boost/graph/adjacency_list.hpp>
#include <boost/graph/graph_utility.hpp>
#include <boost/graph/bellman_ford_shortest_paths.hpp>

using namespace boost;



template < typename Graph, typename ParentMap >
struct edge_writer
{
edge_writer(const Graph & G0, const ParentMap & pp)
: m_g(G0), m_parent(pp)
{
}
```

```
template < typename Edge > void operator()
(std::ostream & out, const Edge & ee) const
{
out << "[label=\"" << get(edge_weight, m_g, ee) << "\"";
typename graph_traits < Graph >::vertex_descriptor
B= source(ee, m_g), C = target(ee, m_g);
if (m_parent[C] == B)
out << ", color=\"black\"";
else
out << ", color=\"grey\"";
out << "]";
}
const Graph & m_g;
ParentMap m_parent;
};
template < typename Graph, typename Parent >
edge_writer < Graph, Parent >
make_edge_writer(const Graph & G0, const Parent & pp)
{
return edge_writer < Graph, Parent > (G0, pp);
}

struct EdgeProperties {

int weight;
};

typedef adjacency_list_traits<vecS, vecS, directedS> Traits;

typedef property <edge_index_t, int, EdgeProperties>Eproperties;
typedef subgraph < adjacency_list<vecS, vecS, directedS,
property<vertex_color_t, int>, Eproperties > > Graph;

bool NewGraphPos (int in, vector<vector<int> >
Tran, vector<int>m1, vector<int>m2, vector<int>m3,
 vector<int>zx, vector<int>&obj_value, vector<int>&pos_tran,
 vector<int>AvTran, vector<int>PT);

bool NewGraph (int in, int out, int mez_sel,
vector<int> &traccia, vector<vector<int> >Tran,
```

```
vector<int>m1, vector<int>m2, vector<int>m3,
vector<int>&in_value,
 vector<int>&out_value, vector<int>AvTran);

bool se (int name[], int cgr, vector<int>AvTran,
Graph &G0, Graph &G1, Graph &G2,
Graph &G3, Graph &GZ);

int main(int,char*[])
{
using namespace std;
int Nvert=54;
  enum { A, B, C, D, E, F, G, H, I, J, K, L, M, N, O,
  P, Q, R, S, T, U, V, W, X, Y, Z, a, b, c, d,
   e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v,
   w, x, y, z, A1, A2};
  int name[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
  11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
  21, 22, 23, 24, 25, 26, 27, 28, 29, 30,
   31, 32, 33, 34, 35, 36, 37, 38, 39,
   40, 41, 42, 43, 44, 45, 46, 47, 48, 49,
   50, 51, 52, 53};

  int nvert=17;
  int sel[]={0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
  10, 11, 12, 13, 14, 15, 16};

  int tran1[]={0, 9, 10, 1, 11, 2, 12,
  3, 13, 4, 14, 5, -1, -1, -1, -1, -1};

  int tran2[]={28, 15, 16, 29, 17, 30,
   18, 31, -1, -1, 19, 32, 20, 6, -1, -1, -1};

  int tran3[]={33, 21, 22, 34, 23, 35,
   24, 36, -1, -1, 25, -1, 26, -1, 7, 27, 8};

  int zxx[]={37, 38, 39, 40, 41, 42, 43,
  44, 45, 46, 47, 48, 49, 50, 51, 52, 53};


vector<vector<int> >Tran;
```

```
vector<int>m1, m2, m3, zx;
vector<int> obj_value,pos_tran, in_value, out_value;
vector<int>AvTran, PT, segna;
int in, out;
int cgr=0;
int mez_sel;


Graph G0(Nvert);
Graph& G1=G0.create_subgraph();
Graph& G2=G0.create_subgraph();
Graph& G3=G0.create_subgraph();
Graph& GZ=G0.create_subgraph();

std::vector <float> peso;
peso.clear();

ifstream file("/Users/Liliana/Desktop/FMS_POS/GRAPH.txt");
string line;

bool startReading=false;
while (getline(file, line)) {

if (startReading) {
//stai leggendo i dati giusti
if (line.find("[") != string::npos) {
cout << "Fine sezione" <<endl;
startReading =false;
break;
}
istringstream ss(line);
int x, y;
int value;

ss >> x >> y >> value;

add_edge(x, y, G0);

peso.push_back(value);

}
```

```
else {

if (line.find("[GRAPH]") != string::npos) {
cout << "Sezione trovata!" <<endl;
startReading=true;
continue;
}

}

}
do{
cout << "Selezionare il nodo di ingresso tra : "
<< endl;
for (int ii = 0; ii < nvert; ++ii)
std::cout << sel[ii] << std::endl;

cout << "Inserire il nodo scelto (0-13): ";
cin >> in;
cout <<"in: "<<in << "\n";
if(in>nvert-1)
{
cout << "E' stato inserito un valore errato,
inserire un valore tra 0 e 13"<<endl;
}
}while(in>nvert-1);

do{
cout << "Selezionare il nodo destinazione tra: "
<< endl;
for (int ii = 0; ii < nvert; ++ii)
std::cout << sel[ii] << std::endl;

cout << "Inserire il nodo scelto (0-13): ";
cin >> out;
cout << "\n";
cout <<"out: "<<out << "\n";
cout <<"in: "<<in << "\n";
if(out>nvert-1)
{
cout << "E' stato inserito un valore errato,
```

```
 inserire un valore tra 0 e 13"<<endl;
}
}while(out>nvert-1);


for(int ij=0; ij<=nvert; ij++)
{
m1.push_back(tran1[ij]);
m2.push_back(tran2[ij]);
m3.push_back(tran3[ij]);
zx.push_back(zxx[ij]);
}
Tran.push_back(m1);
Tran.push_back(m2);
Tran.push_back(m3);
Tran.push_back(zx);

int Pos=0;
cout << " \n";

for(int ii=1; ii<=3; ii++)
        {
 cout << "Il mezzo di trasporto "
  << ii << " e' disponibile? yes [Y] - no [N] ";
 char answer;
 cin >> answer;

 if (answer=='N' || answer == 'n') {
        cout << "\n Mezzo di trasporto "
         << ii << " NON aggiunto  \n";

        AvTran.push_back(0);
        continue; // va al prossimo mezzo

 }
 if (answer=='Y' || answer == 'y') {
        cout << "\n Mezzo di trasporto "
        << ii << "  aggiunto \n\n";


        if(ii==1){
```

```
AvTran.push_back(ii);
do{
        cout << "Inserire il nodo in cui si trova il
        trasportatore '" << ii <<"' : " <<  endl;

        for (int ij = 0; ij < nvert; ++ij)
 std::cout << sel[ij] << std::endl;

        cout << "Inserire il nodo scelto: ";
        cin >> Pos;
        cout << "\n";
        if((Pos>nvert-1)||(Pos<0))
        {
 cout << "E' stato inserito un valore errato"<<endl;
        }
}while((Pos>nvert-1)||(Pos<0));


if((in==0)||(in==1)||(in==2)||(in==3)||(in==4)||
(in==5)||(in==6)||(in==7)||(in==8)||(in==9)
||(in==10)||(in==11))
{
        PT.push_back(Pos);
        segna.push_back(ii);

}
else {cout << "Non è possibile raggiungere
il nodo di ingresso con il mezzo di trasporto selezionato"
<<endl;
}
        }

        if(ii==2){
AvTran.push_back(ii);
do{
        cout << "Inserire il nodo
         in cui si trova il trasportatore '" << ii
         <<"' : "
          <<  endl;

        for (int ij = 0; ij < nvert; ++ij)
```

```
 std::cout << sel[ij] << std::endl;

        cout << "Inserire il nodo scelto: ";
        cin >> Pos;
        cout << "\n";
        if((Pos>nvert-1)||(Pos<0))
        {
 cout << "E' stato inserito un
valore errato"<<endl;
        }
}while((Pos>nvert-1)||(Pos<0));


if((in==0)||(in==1)||(in==2)||(in==3)||(in==4)||
(in==5)||(in==6)||(in==7)||(in==10)||(in==12))
{
       PT.push_back(Pos);
        segna.push_back(ii);
}
else {cout << "Non è possibile raggiungere
il nodo di ingresso con il mezzo di trasporto selezionato"
<<endl;
}
        }

        if(ii==3){
AvTran.push_back(ii);
do{
        cout << "Inserire il nodo in cui si trova
        il trasportatore '" << ii <<"' : " << endl;

        for (int ij = 0; ij < nvert; ++ij)
 std::cout << sel[ij] << std::endl;

        cout << "Inserire il nodo scelto: ";
        cin >> Pos;
        cout << "\n";
        if((Pos>nvert-1)||(Pos<0))
        {
 cout << "E' stato inserito un valore errato"<<endl;
        }
```

```
 }while((Pos>nvert-1)||(Pos<0));


 if((in==0)||(in==1)||(in==2)||(in==3)||(in==4)||(in==5)
 ||(in==6)||(in==7)||(in==10)||(in==12)||(in==13)||(in==14)
 ||(in==15)||(in==16))
 {
        PT.push_back(Pos);
        segna.push_back(ii);
 }
 else {cout << "Non è possibile raggiungere il
 nodo di ingresso con il mezzo di trasporto selezionato"<<endl;
 }
        }

 }
        }
NewGraphPos(in, Tran, m1, m2, m3, zx,
obj_value, pos_tran, AvTran, PT);

se (name, cgr, AvTran, G0, G1, G2, G3, GZ);

graph_traits < Graph >::edge_iterator ei, ei_end;
property_map<Graph, int EdgeProperties::*>
::type weight_pmap = get(&EdgeProperties::weight, G0);


int ii = 0;
for (boost::tie(ei, ei_end) = edges(G0);
ei != ei_end; ++ei, ++ii)
weight_pmap[*ei] = peso[ii];



std::vector<int> distance1(Nvert,
(std::numeric_limits < short >::max)());
std::vector<std::size_t> parent1(Nvert);


for (int ii = 0; ii < Nvert; ++ii)
parent1[ii] =ii;
```

```
vector<int>save_obj;



for(int jj=0; jj<pos_tran.size(); ++jj)
{
for (size_t kj=0; kj<Nvert; ++kj)
{
distance1[kj]=(std::numeric_limits
 < short >::max)();
}

distance1[pos_tran[jj]] = 0;


bool kk = bellman_ford_shortest_paths
(G0, int (Nvert), weight_map(weight_pmap).
distance_map(&distance1[0]).
 predecessor_map(&parent1[0]));


if (!kk)
{
std::cout << "negative cycle"
<< std::endl;

}

else
{

if(obj_value[jj] != -1){
save_obj.push_back(distance1[obj_value[jj]]);

std::vector<int> cammino;
cammino.clear();

std::cout << "\n Distanza dalla posizione del trasportatore '"
```

```
<< jj+1 << "' al nodo di ingresso:"
<< distance1[obj_value[jj]] << " \n ";

}
}
}
int minimo=30000;
int start=0;


for(int kj=0; kj<obj_value.size(); ++kj)
{
if ((save_obj[kj]<minimo)&&(obj_value[kj] != -1))
{
minimo=save_obj[kj];
start=kj;
}
mez_sel=segna[start]-1;
}

if(minimo!=30000)
{

std::cout << "\n Mezzo di trasporto più vicino
al nodo di ingresso: " << mez_sel+1 << ". \n ";

std::cout << "\n Spazio percorso dal mezzo
più vicino: " << minimo << " metri. \n ";
}

else {
std::cout << "\n Non è possibile raggiungere
 il nodo destinazione.  \n ";
}

cout << " \n";


distance1.clear();

vector<int>traccia;
```

```
NewGraph(in, out, mez_sel, traccia, Tran, m1,
 m2, m3, in_value, out_value, AvTran);

std::vector<int> distance(Nvert, (std::numeric_limits
< short >::max)());
std::vector<std::size_t> parent(Nvert);


for (int ii = 0; ii < Nvert; ++ii)
parent[ii] =ii;


for(int jj=0; jj<in_value.size(); ++jj)
distance[in_value[jj]] = 0;

bool r = bellman_ford_shortest_paths
(G0, int (Nvert), weight_map(weight_pmap).
distance_map(&distance[0]).
 predecessor_map(&parent[0]));

if (!r)
{
std::cout << "negative cycle" << std::endl;

}

else
{
cout << "\n";

int minimo=30000;
int start=0;
int bid;

for(int ii=0; ii<out_value.size(); ii++)
{
if ((distance[out_value[ii]]<minimo)&&
(out_value[ii] >= 0))
{
minimo=distance[out_value[ii]];
```

```
start=ii;
}
}
bid=out_value[start];
std::vector<int> cammino;
cammino.clear();

int stop;
int key;

std::cout << "\n Valore del cammino : "
 << minimo << " \n ";
stop=parent[bid];


do{
key=parent[stop];
stop=key;}
while(distance[key]!=0);

do{
cammino.push_back(name[bid]);
bid=parent[bid];

}
while (name[bid]!= stop);

cammino.push_back(stop);
int cam;
cam=cammino.size()-1;

/////////

cout << "\n";

vector<int>transport;
vector<int>conta;

cout << "\n Nodi attraversati:" ;

for (int ik=cam; ik>=0; --ik)
```

```
cout << cammino[ik]<< " " ;

cout << "\n Numero nodi visitati: " << cammino.size()
 << endl;

if(cammino.size()==2)
{
cout << "\n Numero cambi mezzo di trasporto effettuati: 0"
<< endl;

}
else{

for(int ii=0; ii<=cam; ii++)
{
for(int jj=0; jj<m1.size(); ++jj)
{
if(cammino[ii]==m1[jj])

transport.push_back(1);
}
}
for(int ii=0; ii<=cam; ii++)
{
for(int jj=0; jj<m2.size(); ++jj)
{
if(cammino[ii]==m2[jj])

transport.push_back(2);
}
}
for(int ii=0; ii<=cam; ii++)
{
for(int jj=0; jj<m3.size(); ++jj)
{
if(cammino[ii]==m3[jj])

transport.push_back(3);
}
}
```

```
cam=transport.size()-1;
int ij=0;
do{
if(transport[ij]!=transport[ij+1])
{
conta.push_back(1);
}
ij=ij++;}
while(ij!=cam);


cout << "\n";

cout << "Lista mezzi di trasporto utilizzati: ";
for (int ik=0; ik<=cam; ++ik)
cout<< transport[ik]<< " ";

cout << "\n";

cout << "Numero cambi mezzo di trasporto effettuati: "
<< conta.size()<<endl;
cout << "\n";

}

std::cout << "\n Mezzo di trasporto utilizzato
per il primo spostamento è il mezzo: ': " << mez_sel+1
 << "'. \n ";
cout << "Il mezzo di trasporto utilizzato
per l'ultimo spostamento è il mezzo: '"
 <<AvTran[traccia[start]] << "' "<<endl;

}

std::ofstream dot_file("figs/bellman-eg.dot");
dot_file << "digraph D {\n"
<< "  rankdir=LR\n"
<< "  size=\"5,3\"\n"
<< "  ratio=\"fill\"\n"
<< "  edge[style=\"bold\"]\n" << "
 node[shape=\"circle\"]\n";
```

```
{
for (boost::tie(ei, ei_end) = edges(G0);
ei != ei_end; ++ei) {
graph_traits < Graph >::edge_descriptor ee = *ei;
graph_traits < Graph >::vertex_descriptor
B = source(ee, G0), C = target(ee, G0);
// VC++ doesn't like the 3-argument get function, so here
// we workaround by using 2-nested get()'s.
dot_file << name[B] << " -> " << name[C]
<< "[label=\"" << get(get(&EdgeProperties::weight, G0), ee)
<< "\"";
if (parent[B] == C)
dot_file << ", color=\"black\"";
else
dot_file << ", color=\"grey\"";
dot_file << "]";
}
}
dot_file << "}";
return EXIT_SUCCESS;
}

bool NewGraphPos (int in, vector<vector<int> >
Tran, vector<int>m1, vector<int>m2, vector<int>m3,
vector<int>zx, vector<int>&obj_value,
vector<int>&pos_tran, vector<int>AvTran, vector<int>PT)

{
for(int jj=0; jj<PT.size(); ++jj)
{
pos_tran.push_back(Tran[3][PT[jj]]);
cout << endl;
}

for(int ii=0; ii<AvTran.size(); ii++)
{
if(AvTran[ii]!=0)
{

obj_value.push_back(Tran[AvTran[ii]-1][in]);
```

```
}
}
return true;
}

bool NewGraph(int in, int out, int mez_sel,
vector<int>&traccia, vector<vector<int> >Tran,
vector<int>m1, vector<int>m2, vector<int>m3,
vector<int>&in_value, vector<int>&out_value,
 vector<int>AvTran)
{
for(int ii=0; ii<AvTran.size(); ii++)
{
if(AvTran[ii]!=0)
{
out_value.push_back(Tran[ii][out]);
traccia.push_back(ii);// tiene traccia del
mezzo scelto per dire il mezzo che uso
}
}

in_value.push_back(Tran[mez_sel][in]);

return true;
}

bool se (int name[], int cgr, vector<int>AvTran,
Graph &G0, Graph &G1, Graph &G2, Graph &G3,
Graph &GZ)
{
bool se (int name[], int cgr, vector<int>AvTran,
 Graph &G0, Graph &G1, Graph &G2, Graph &G3,
 Graph &GZ)
{
        if((AvTran[cgr] == 1) && (AvTran[cgr+1] == 0)
        && (AvTran[cgr+2] == 0))
        {
                add_vertex(0, G1);
                add_vertex(1, G1);
                add_vertex(2, G1);
```

```
            add_vertex(3, G1);
            add_vertex(4, G1);
            add_vertex(5, G1);
            add_vertex(9, G1);
            add_vertex(10, G1);
            add_vertex(11, G1);
            add_vertex(12, G1);
            add_vertex(13, G1);
            add_vertex(14, G1);
            add_vertex(37, GZ);
            add_vertex(38, GZ);
            add_vertex(39, GZ);
            add_vertex(40, GZ);
            add_vertex(41, GZ);
            add_vertex(42, GZ);
            add_vertex(43, GZ);
            add_vertex(44, GZ);
            add_vertex(45, GZ);
            add_vertex(46, GZ);
            add_vertex(47, GZ);
            add_vertex(48, GZ);
            add_vertex(49, GZ);
            add_vertex(50, GZ);
            add_vertex(51, GZ);
            add_vertex(52, GZ);
            add_vertex(53, GZ);
}


if((AvTran[cgr] == 0) && (AvTran[cgr+1] == 2)
 && (AvTran[cgr+2] == 0))

{

            add_vertex(6, G2);
            add_vertex(15, G2);
            add_vertex(16, G2);
            add_vertex(17, G2);
            add_vertex(18, G2);
            add_vertex(19, G2);
            add_vertex(20, G2);
```

```
            add_vertex(28, G2);
            add_vertex(29, G2);
            add_vertex(30, G2);
            add_vertex(31, G2);
            add_vertex(32, G2);
            add_vertex(37, GZ);
            add_vertex(38, GZ);
            add_vertex(39, GZ);
            add_vertex(40, GZ);
            add_vertex(41, GZ);
            add_vertex(42, GZ);
            add_vertex(43, GZ);
            add_vertex(44, GZ);
            add_vertex(45, GZ);
            add_vertex(46, GZ);
            add_vertex(47, GZ);
            add_vertex(48, GZ);
            add_vertex(49, GZ);
            add_vertex(50, GZ);
            add_vertex(51, GZ);
            add_vertex(52, GZ);
            add_vertex(53, GZ);

}


if((AvTran[cgr] == 0) && (AvTran[cgr+1] == 0)
&& (AvTran[cgr+2] == 3))

{

            add_vertex(7, G3);
            add_vertex(8, G3);
            add_vertex(21, G3);
            add_vertex(22, G3);
            add_vertex(23, G3);
            add_vertex(24, G3);
            add_vertex(25, G3);
            add_vertex(26, G3);
            add_vertex(27, G3);
            add_vertex(33, G3);
```

```
        add_vertex(34, G3);
        add_vertex(35, G3);
        add_vertex(36, G3);
        add_vertex(37, GZ);
        add_vertex(38, GZ);
        add_vertex(39, GZ);
        add_vertex(40, GZ);
        add_vertex(41, GZ);
        add_vertex(42, GZ);
        add_vertex(43, GZ);
        add_vertex(44, GZ);
        add_vertex(45, GZ);
        add_vertex(46, GZ);
        add_vertex(47, GZ);
        add_vertex(48, GZ);
        add_vertex(49, GZ);
        add_vertex(50, GZ);
        add_vertex(51, GZ);
        add_vertex(52, GZ);
        add_vertex(53, GZ);
}


if((AvTran[cgr] == 1) && (AvTran[cgr+1] == 2)
 && (AvTran[cgr+2] == 0))

{
        add_vertex(0, G1);
        add_vertex(1, G1);
        add_vertex(2, G1);
        add_vertex(3, G1);
        add_vertex(4, G1);
        add_vertex(5, G1);
        add_vertex(6, G2);
        add_vertex(9, G1);
        add_vertex(10, G1);
        add_vertex(11, G1);
        add_vertex(12, G1);
        add_vertex(13, G1);
        add_vertex(14, G1);
```

```
            add_vertex(15, G2);
            add_vertex(16, G2);
            add_vertex(17, G2);
            add_vertex(18, G2);
            add_vertex(19, G2);
            add_vertex(20, G2);
            add_vertex(28, G2);
            add_vertex(29, G2);
            add_vertex(37, GZ);
            add_vertex(38, GZ);
            add_vertex(39, GZ);
            add_vertex(40, GZ);
            add_vertex(41, GZ);
            add_vertex(42, GZ);
            add_vertex(43, GZ);
            add_vertex(44, GZ);
            add_vertex(45, GZ);
            add_vertex(46, GZ);
            add_vertex(47, GZ);
            add_vertex(48, GZ);
            add_vertex(49, GZ);
            add_vertex(50, GZ);
            add_vertex(51, GZ);
            add_vertex(52, GZ);
            add_vertex(53, GZ);
}

if((AvTran[cgr] == 1) && (AvTran[cgr+1] == 0)
&& (AvTran[cgr+2] == 3))


{
            add_vertex(0, G1);
            add_vertex(1, G1);
            add_vertex(2, G1);
            add_vertex(3, G1);
            add_vertex(4, G1);
            add_vertex(5, G1);
            add_vertex(7, G3);
            add_vertex(8, G3);
            add_vertex(9, G1);
```

```
            add_vertex(10, G1);
            add_vertex(11, G1);
            add_vertex(12, G1);
            add_vertex(13, G1);
            add_vertex(14, G1);
            add_vertex(21, G3);
            add_vertex(22, G3);
            add_vertex(23, G3);
            add_vertex(24, G3);
            add_vertex(25, G3);
            add_vertex(26, G3);
            add_vertex(27, G3);
            add_vertex(33, G3);
            add_vertex(34, G3);
            add_vertex(35, G3);
            add_vertex(36, G3);
            add_vertex(37, GZ);
            add_vertex(38, GZ);
            add_vertex(39, GZ);
            add_vertex(40, GZ);
            add_vertex(41, GZ);
            add_vertex(42, GZ);
            add_vertex(43, GZ);
            add_vertex(44, GZ);
            add_vertex(45, GZ);
            add_vertex(46, GZ);
            add_vertex(47, GZ);
            add_vertex(48, GZ);
            add_vertex(49, GZ);
            add_vertex(50, GZ);
            add_vertex(51, GZ);
            add_vertex(52, GZ);
            add_vertex(53, GZ);
}


if((AvTran[cgr] == 0) && (AvTran[cgr+1] == 2)
 && (AvTran[cgr+2] == 3))

{
```

```
add_vertex(6, G2);
add_vertex(7, G3);
add_vertex(8, G3);
add_vertex(15, G2);
add_vertex(16, G2);
add_vertex(17, G2);
add_vertex(18, G2);
add_vertex(19, G2);
add_vertex(20, G2);
add_vertex(21, G3);
add_vertex(22, G3);
add_vertex(23, G3);
add_vertex(24, G3);
add_vertex(25, G3);
add_vertex(26, G3);
add_vertex(27, G3);
add_vertex(28, G2);
add_vertex(29, G2);
add_vertex(30, G2);
add_vertex(31, G2);
add_vertex(32, G2);
add_vertex(33, G3);
add_vertex(34, G3);
add_vertex(35, G3);
add_vertex(36, G3);
add_vertex(37, GZ);
add_vertex(38, GZ);
add_vertex(39, GZ);
add_vertex(40, GZ);
add_vertex(41, GZ);
add_vertex(42, GZ);
add_vertex(43, GZ);
add_vertex(44, GZ);
add_vertex(45, GZ);
add_vertex(46, GZ);
add_vertex(47, GZ);
add_vertex(48, GZ);
add_vertex(49, GZ);
add_vertex(50, GZ);
add_vertex(51, GZ);
add_vertex(52, GZ);
```

```
            add_vertex(53, GZ);


}

if((AvTran[cgr] == 1) && (AvTran[cgr+1] == 2)
 && (AvTran[cgr+2] == 3))
{

            add_vertex(0, G1);
            add_vertex(1, G1);
            add_vertex(2, G1);
            add_vertex(3, G1);
            add_vertex(4, G1);
            add_vertex(5, G1);
            add_vertex(6, G2);
            add_vertex(7, G3);
            add_vertex(8, G3);
            add_vertex(9, G1);
            add_vertex(10, G1);
            add_vertex(11, G1);
            add_vertex(12, G1);
            add_vertex(13, G1);
            add_vertex(14, G1);
            add_vertex(15, G2);
            add_vertex(16, G2);
            add_vertex(17, G2);
            add_vertex(18, G2);
            add_vertex(19, G2);
            add_vertex(20, G2);
            add_vertex(21, G3);
            add_vertex(22, G3);
            add_vertex(23, G3);
            add_vertex(24, G3);
            add_vertex(25, G3);
            add_vertex(26, G3);
            add_vertex(27, G3);
            add_vertex(28, G2);
            add_vertex(29, G2);
            add_vertex(30, G2);
            add_vertex(31, G2);
```

```
                add_vertex(32, G2);
                add_vertex(33, G3);
                add_vertex(34, G3);
                add_vertex(35, G3);
                add_vertex(36, G3);
                add_vertex(37, GZ);
                add_vertex(38, GZ);
                add_vertex(39, GZ);
                add_vertex(40, GZ);
                add_vertex(41, GZ);
                add_vertex(42, GZ);
                add_vertex(43, GZ);
                add_vertex(44, GZ);
                add_vertex(45, GZ);
                add_vertex(46, GZ);
                add_vertex(47, GZ);
                add_vertex(48, GZ);
                add_vertex(49, GZ);
                add_vertex(50, GZ);
                add_vertex(51, GZ);
                add_vertex(52, GZ);
                add_vertex(53, GZ);


        }

        if((AvTran[cgr] == 0) && (AvTran[cgr+1] == 0) &&
        (AvTran[cgr+2] == 0))
        {
                cout << "Non è stato selezionato alcun mezzo
                di trasporto";
        }
        return true;

}
```

Il file.txt che legge gli archi e i pesi è il seguente:

```
[GRAPH]
0 9 22
1 10 22
2 11 24
3 12 22
4 13 21
5 14 21
6 20 21
7 26 23
8 27 23
9 0 22
9 10 5
10 1 22
10 9 5
10 11 5
11 2 24
11 10 5
11 12 3
12 3 22
12 11 3
12 13 4
13 4 21
13 12 4
13 14 3
14 5 21
14 13 3
14 19 6
14 25 6
15 16 5
15 28 22
16 15 5
16 17 5
16 29 22
17 16 5
17 18 3
17 30 24
18 17 3
18 19 6
18 31 22
19 14 6
19 18 6
19 20 3
19 25 6
19 32 27
20 6 22
20 19 7
20 26 2
21 22 7
21 33 27
22 21 2
22 23 7
22 34 24
23 22 7
23 24 4
23 35 26
24 23 4
24 25 6
24 36 22
25 14 6
25 19 6
25 24 8
25 26 4
26 7 23
26 20 6
26 25 4
26 27 3
27 8 23
27 26 3
28 15 22
29 16 22
30 17 24
31 18 22
32 19 21
33 21 22
34 22 22
35 23 26
36 24 22
37 0 0
37 1 11
37 2 22
37 3 22
37 4 27
37 5 34
37 6 34
37 7 36
37 8 39
37 9 2
37 10 9
37 11 16
37 12 20
37 13 26
37 14 29
37 15 2
37 16 9
37 17 16
37 18 20
37 19 29
37 20 33
37 21 2
37 22 9
37 23 16
37 24 20
37 25 29
37 26 33
37 27 36
37 28 0
37 29 11
37 30 22
37 31 22
37 32 34
37 33 0
37 34 11
37 35 22
37 36 22
37 38 16
37 39 20
37 40 22
37 41 26
37 42 27
37 43 29
37 44 34
37 45 2
37 46 34
37 47 9
37 48 36
37 49 9
37 50 39
37 51 2
37 52 11
37 53 9
38 0 16
38 1 9
38 2 20
38 3 20
38 4 25
38 5 30
38 6 35
38 7 37
38 8 43
38 9 0
38 10 7
38 11 14
38 12 18
38 13 24
38 14 27
38 15 0
38 16 14
38 17 21
38 18 25
38 19 34
38 20 38
38 21 0
38 22 7
38 23 14
38 24 18
38 25 27
```

38 26 31
38 27 34
38 28 20
38 29 25
38 30 20
38 31 20
38 32 43
38 33 20
38 34 25
38 35 20
38 36 20
38 37 20
38 39 7
38 40 25
38 41 14
38 42 20
38 43 18
38 44 20
38 45 0
38 46 37
38 47 27
38 48 43
38 49 31
38 50 0
38 51 7
38 52 34
38 53 14
39 0 9
39 1 2
39 2 13
39 3 13
39 4 18
39 5 21
39 6 25
39 7 27
39 8 27
39 9 30
39 10 0
39 11 7
39 12 11
39 13 17
39 14 20
39 15 7
39 16 0
39 17 7
39 18 11
39 19 20
39 20 24
39 21 7
39 22 0
39 23 0
39 24 4
39 25 13
39 26 17
39 27 20
39 28 9
39 29 2
39 30 21
39 31 25
39 32 27
39 33 9
39 34 18
39 35 21
39 36 25
39 37 9
39 38 0
39 40 18
39 41 0
39 42 21
39 43 4
39 44 25
39 45 7
39 46 27
39 47 13
39 48 27
39 49 17
39 50 30
39 51 0
39 52 20
39 53 7
40 0 11
40 1 0
40 2 15
40 3 15
40 4 20
40 5 23
40 6 27
40 7 29
40 8 32
40 9 9
40 10 2
40 11 9
40 12 13
40 13 19
40 14 22
40 15 9
40 16 2
40 17 9
40 18 13
40 19 22
40 20 26
40 21 9
40 22 2
40 23 9
40 24 13
40 25 22
40 26 26
40 27 29
40 28 11
40 29 0
40 30 15
40 31 15
40 32 23
40 33 11
40 34 0
40 35 15
40 36 15
40 37 11
40 38 9
40 39 2
40 41 9
40 42 15
40 43 13
40 44 15
40 45 19
40 46 20
40 47 22
40 48 23
40 49 26
40 50 27
40 51 29
40 52 29
40 53 32
41 0 16
41 1 9
41 2 6
41 3 6
41 4 11
41 5 14
41 6 18
41 7 20
41 8 23
41 9 14
41 10 7
41 11 0
41 12 4
41 13 10
41 14 13
41 15 14
41 16 7
41 17 0
41 18 4
41 19 13
41 20 17
41 21 14
41 22 7
41 23 0
41 24 4
41 25 13
41 26 17
41 27 20
41 28 16
41 29 9
41 30 6

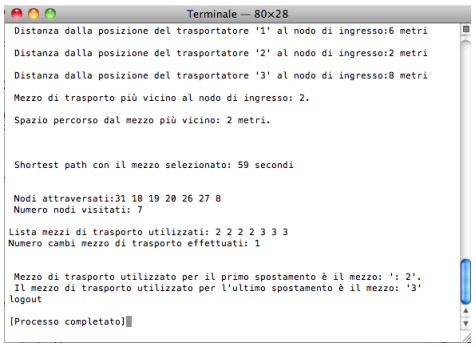
41 31 6
41 32 14
41 33 16
41 34 9
41 35 6
41 36 6
41 37 16
41 38 14
41 39 7
41 40 9
41 42 6
41 43 4
41 44 6
41 45 10
41 46 11
41 47 13
41 48 14
41 49 17
41 50 18
41 51 20
41 52 20
41 53 23
42 0 22
42 1 15
42 2 0
42 3 12
42 4 17
42 5 20
42 6 24
42 7 26
42 8 29
42 9 20
42 10 13
42 11 6
42 12 10
42 13 16
42 14 9
42 15 20
42 16 13
42 17 6
42 18 10
42 19 19
42 20 23
42 21 20
42 22 13
42 23 6
42 24 10
42 25 19
42 26 23
42 27 26
42 28 22
42 29 15
42 30 0
42 31 12
42 32 20
42 33 22
42 34 15
42 35 0
42 36 12
42 37 22
42 38 20
42 39 13
42 40 15
42 41 6
42 43 10
42 44 12
42 45 16
42 46 17
42 47 19
42 48 20
42 49 23
42 50 24
42 51 26
42 52 26
42 53 29
43 0 20
43 1 13
43 2 10
43 3 2
43 4 7
43 5 11
43 6 15
43 7 17
43 8 20
43 9 18
43 10 11
43 11 4
43 12 0
43 13 6
43 14 9
43 15 18
43 16 11
43 17 4
43 18 0
43 19 9
43 20 13
43 21 18
43 22 11
43 23 4
43 24 0
43 25 9
43 26 13
43 27 16
43 28 20
43 29 13
43 30 10
43 31 2
43 32 11
43 33 20
43 34 13
43 35 10
43 36 2
43 37 10
43 38 18
43 39 11
43 40 13
43 41 4
43 42 10
43 44 2
43 45 6
43 46 7
43 47 9
43 48 11
43 49 13
43 50 15
43 51 17
43 52 16
43 53 20
44 0 22
44 1 15
44 2 12
44 3 0
44 4 9
44 5 12
44 6 16
44 7 18
44 8 21
44 9 20
44 10 13
44 11 6
44 12 12
44 13 8
44 14 11
44 15 20
44 16 13
44 17 6
44 18 12
44 19 11
44 20 15
44 21 20
44 22 13
44 23 6
44 24 12
44 25 11
44 26 15
44 27 18
44 28 22
44 29 15
44 30 12
44 31 0
44 32 12
44 33 22
44 34 15
44 35 12

```
44 36 0
44 37 22
44 38 20
44 39 13
44 40 15
44 41 6
44 42 12
44 43 12
44 45 8
44 46 9
44 47 11
44 48 12
44 49 15
44 50 16
44 51 18
44 52 18
44 53 21
45 0 26
45 1 19
45 2 16
45 3 8
45 4 1
45 5 5
45 6 9
45 7 11
45 8 14
45 9 24
45 10 17
45 11 10
45 12 6
45 13 0
45 14 4
45 15 24
45 16 27
45 17 10
45 18 6
45 19 4
45 20 8
45 21 24
45 22 17
45 23 10
45 24 6
45 25 4
45 26 8
45 27 11
45 28 26
45 29 19
45 30 16
45 31 8
45 32 5
45 33 26
45 34 19
45 35 16
45 36 8
45 37 26
45 38 24
45 39 17
45 40 19
45 41 10
45 42 16
45 43 6
45 44 8
45 46 1
45 47 4
45 48 5
45 49 8
45 50 9
45 51 11
45 52 11
45 53 14
46 0 27
46 1 20
46 2 17
46 3 9
46 4 0
46 5 6
46 6 10
46 7 12
46 8 15
46 9 25
46 10 18
46 11 11
46 12 7
46 13 1
46 14 5
46 15 25
46 16 18
46 17 11
46 18 7
46 19 5
46 20 9
46 21 25
46 22 18
46 23 11
46 24 7
46 25 5
46 26 9
46 27 12
46 28 27
46 29 20
46 30 17
46 31 9
46 32 6
46 33 27
46 34 20
46 35 17
46 36 9
46 37 27
46 38 25
46 39 18
46 40 20
46 41 11
46 42 17
46 43 7
46 44 9
46 45 1
46 47 5
46 48 6
46 49 9
46 50 10
46 51 12
46 52 12
46 53 15
47 0 29
47 1 22
47 2 19
47 3 11
47 4 5
47 5 1
47 6 5
47 7 7
47 8 10
47 9 27
47 10 20
47 11 13
47 12 9
47 13 4
47 14 0
47 15 27
47 16 20
47 17 13
47 18 9
47 19 0
47 20 4
47 21 27
47 22 20
47 23 13
47 24 9
47 25 0
47 26 4
47 27 7
47 28 29
47 29 22
47 30 19
47 31 11
47 32 1
47 33 29
47 34 22
47 35 19
47 36 11
47 37 29
47 38 27
47 39 20
47 40 22
```

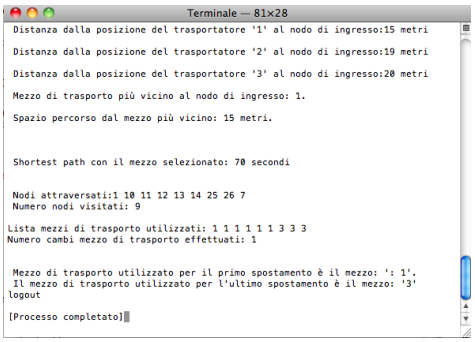
47 41 13
47 42 19
47 43 9
47 44 11
47 45 4
47 46 5
47 48 1
47 49 4
47 50 5
47 51 7
47 52 7
47 53 10
48 0 30
48 1 23
48 2 20
48 3 12
48 4 6
48 5 0
48 6 6
48 7 8
48 8 11
48 9 28
48 10 21
48 11 14
48 12 10
48 13 5
48 14 1
48 15 28
48 16 21
48 17 14
48 18 10
48 19 1
48 20 5
48 21 28
48 22 21
48 23 14
48 24 10
48 25 1
48 26 5
48 27 8
48 28 30
48 29 23
48 30 20
48 31 12
48 32 0
48 33 30
48 34 23
48 35 20
48 36 12
48 37 30
48 38 28
48 39 21
48 40 23
48 41 14
48 42 20
48 43 10
48 44 12
48 45 5
48 46 6
48 49 1
48 49 5
48 50 6
48 51 8
48 52 8
48 53 11
49 0 33
49 1 26
49 2 23
49 3 15
49 4 9
49 5 5
49 6 1
49 7 3
49 8 6
49 9 31
49 10 24
49 11 17
49 12 13
49 13 8
49 14 4
49 15 31
49 16 24
49 17 17
49 18 13
49 19 4
49 20 4
49 21 31
49 22 24
49 23 17
49 24 13
49 25 4
49 26 4
49 27 3
49 28 33
49 29 26
49 30 23
49 31 15
49 32 5
49 33 33
49 34 26
49 35 23
49 36 15
49 37 33
49 38 31
49 39 24
49 40 26
49 41 17
49 42 23
49 43 13
49 44 15
49 45 8
49 46 9
49 47 4
49 48 5
49 50 1
49 51 3
49 52 3
49 53 6
50 0 34
50 1 27
50 2 24
50 3 16
50 4 10
50 5 6
50 6 0
50 7 4
50 8 7
50 9 32
50 10 25
50 11 18
50 12 14
50 13 9
50 14 5
50 15 32
50 16 25
50 17 18
50 18 14
50 19 5
50 20 1
50 21 32
50 22 25
50 23 18
50 24 14
50 25 5
50 26 1
50 27 4
50 28 34
50 29 27
50 30 24
50 31 16
50 32 6
50 33 34
50 34 27
50 35 24
50 36 16
50 37 34
50 38 32
50 39 25
50 40 27
50 41 18
50 42 24
50 43 14
50 44 16
50 45 9

```
50 46 10
50 47 5
50 48 6
50 49 1
50 50 4
50 52 4
50 53 7
51 0 36
51 1 29
51 2 26
51 3 18
51 4 12
51 5 8
51 6 4
51 7 0
51 8 9
51 9 34
51 10 27
51 11 20
51 12 16
51 13 11
51 14 7
51 15 34
51 16 27
51 17 20
51 18 16
51 19 7
51 20 3
51 21 34
51 22 27
51 23 20
51 24 16
51 25 7
51 26 3
51 27 6
51 28 36
51 29 29
51 30 26
51 31 18
51 32 8
51 33 36
51 34 29
51 35 26
51 36 18
51 37 36
51 38 34
51 39 27
51 40 29
51 41 20
51 42 26
51 43 16
51 44 18
51 45 11
51 46 12
51 47 7
51 48 8
51 49 3
51 50 4
51 51 6
51 53 9
52 0 36
52 1 29
52 2 26
52 3 18
52 4 12
52 5 8
52 6 4
52 7 6
52 8 3
52 9 34
52 10 27
52 11 20
52 12 16
52 13 11
52 14 7
52 15 34
52 16 27
52 17 20
52 18 16
52 19 7
52 20 3
52 21 34
52 22 27
52 23 20
52 24 16
52 25 7
52 26 3
52 27 0
52 28 36
52 29 29
52 30 26
52 31 18
52 32 8
52 33 36
52 34 29
52 35 26
52 36 18
52 37 36
52 38 24
52 39 27
52 40 29
52 41 20
52 42 26
52 43 16
52 44 18
52 45 11
52 46 12
52 47 7
52 48 8
52 49 3
52 50 4
52 51 6
52 53 3
53 0 39
53 1 33
53 2 29
53 3 21
53 4 15
53 5 11
53 6 7
53 7 9
53 8 0
53 9 37
53 10 30
53 11 23
53 12 19
53 13 14
53 14 10
53 15 37
53 16 30
53 17 23
53 18 19
53 19 10
53 20 6
53 21 37
53 22 30
53 23 23
53 24 19
53 25 10
53 26 6
53 27 3
53 28 39
53 29 33
53 30 29
53 31 21
53 32 11
53 33 39
53 34 33
53 35 29
53 36 21
53 37 39
53 38 37
53 39 30
53 40 33
53 41 23
53 42 29
53 43 19
53 44 21
53 45 14
53 46 15
53 47 10
53 48 11
53 49 6
53 50 7
53 51 9
53 52 3
```

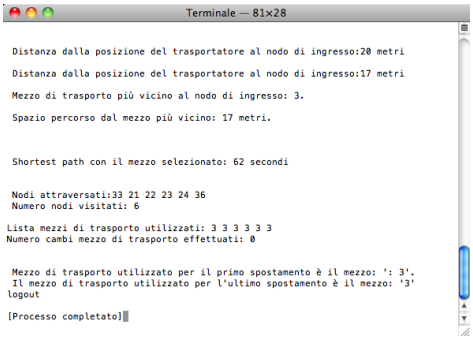
Seguono in Figura A.1, ulteriori esempi di utilizzo del programma per la validazione del modello.



(a) Da 7 a 16 con m1, m2 ed m3



(b) Da 3 a 14 con m1, m2 ed m3



(c) Da 0 a 6 con m1 ed m2

Figura A.1: Ulteriori esempi di compilazione per la validazione del programma in $C++$.

Appendice B

Programma in Linguaggio C++ per il Caso Reale: Impianto di Rettifica Tenova Pomini Spa

Listing B.1: Programma C++ per il Caso Reale Tenova Pomini

```
#include <boost/config.hpp>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <vector>
#include <vector.h>
#include <iomanip>
#include <utility>
#include <cmath>
#include <boost/graph/subgraph.hpp>
#include <boost/graph/adjacency_list.hpp>
#include <boost/graph/graph_utility.hpp>
#include <boost/graph/bellman_ford_shortest_paths.hpp>

using namespace boost;



template < typename Graph, typename ParentMap >
struct edge_writer
{
edge_writer(const Graph & G0, const ParentMap & pp)
: m_g(G0), m_parent(pp)
```

```
{
}

template < typename Edge >
    void operator() (std::ostream & out, const Edge & ee) const
{
out << "[label=\"" << get(edge_weight, m_g, ee) << "\"";
typename graph_traits < Graph >::vertex_descriptor
B= source(ee, m_g), C = target(ee, m_g);
if (m_parent[C] == B)
out << ", color=\"black\"";
else
out << ", color=\"grey\"";
out << "]";
}
const Graph & m_g;
ParentMap m_parent;
};
template < typename Graph, typename Parent >
edge_writer < Graph, Parent >
make_edge_writer(const Graph & G0, const Parent & pp)
{
return edge_writer < Graph, Parent > (G0, pp);
}

struct EdgeProperties {

int weight;
};

typedef adjacency_list_traits<vecS, vecS, directedS> Traits;

typedef property <edge_index_t, int, EdgeProperties >
Eproperties;
typedef subgraph < adjacency_list<vecS, vecS, directedS,
property<vertex_color_t, int>, Eproperties > > Graph;

bool NewGraphPos (int in, vector<vector<int> >
Tran, vector<int>m1, vector<int>m2, vector<int>m3,
 vector<int>zx, vector<int>&obj_value, vector<int>&pos_tran,
 vector<int>AvTran, vector<int>PT);
```

```
bool NewGraph (int in, int out, int mez_sel,
vector<int> &traccia, vector<vector<int> >Tran,
vector<int>m1, vector<int>m2, vector<int>m3,
vector<int>&in_value, vector<int>&out_value,
 vector<int>AvTran);

bool se (int name[], int cgr, vector<int>AvTran, Graph &G0,
Graph &G1, Graph &G2, Graph &G3, Graph &GZ);

int main(int,char*[])
{
using namespace std;

int Nvert=32;
enum { A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P,
Q, R, S, T, U, V, W, X, Y, Z, A1, A2, A3, A4, A5, A6};
int name[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25,
 26, 27, 28, 29, 30, 31};

int nvert=14;
int sel[]={0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
12, 13};

int tran1[]={0, 1, 2, 3, 4, 5, 6, -1, -1, 7, -1, -1,
-1, -1};

int tran2[]={-1, -1, 8, -1, -1, 9, -1, -1, -1, 10, -1,
 -1, 11, 12};

int tran3[]={-1, -1, -1, -1, -1, -1, -1, 13, 14, 15, 16,
17, -1, -1};

int zxx[]={18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29,
 30, 31};

vector<vector<int> >Tran;
vector<int>m1, m2, m3, zx;
vector<int> obj_value,pos_tran, in_value, out_value;
vector<int>AvTran, PT, segna;
```

```
int in, out;
int cgr=0;
int mez_sel;


Graph G0(Nvert);
Graph& G1=G0.create_subgraph();
Graph& G2=G0.create_subgraph();
Graph& G3=G0.create_subgraph();
Graph& GZ=G0.create_subgraph();

std::vector <float> peso;
peso.clear();

ifstream file("/Users/Liliana/Desktop/POMINI_2CP1GZ
/GRAPH.txt");
string line;

bool startReading=false;
while (getline(file, line)) {

if (startReading) {
//stai leggendo i dati giusti
if (line.find("[") != string::npos) {
cout << "Fine sezione" <<endl;
startReading =false;
break;
}
istringstream ss(line);
int x, y;
int value;

ss >> x >> y >> value;

add_edge(x, y, G0);

peso.push_back(value);

}
else {
```

```
if (line.find("[GRAPH]") != string::npos) {
cout << "Sezione trovata!" <<endl;
startReading=true;
continue;
}

}

}
do{
cout << "Selezionare il nodo di ingresso tra : "
<< endl;
for (int ii = 0; ii < nvert; ++ii)
std::cout << sel[ii] << std::endl;

cout << "Inserire il nodo scelto (0-13): ";
cin >> in;
cout <<"in: "<<in << "\n";
if(in>nvert-1)
{
cout << "E' stato inserito un valore errato,
inserire un valore tra 0 e 13"<<endl;
}
}while(in>nvert-1);

do{
cout << "Selezionare il nodo destinazione tra: "
<< endl;
for (int ii = 0; ii < nvert; ++ii)
std::cout << sel[ii] << std::endl;

cout << "Inserire il nodo scelto (0-13): ";
cin >> out;
cout << "\n";
cout <<"out: "<<out << "\n";
cout <<"in: "<<in << "\n";
if(out>nvert-1)
{
cout << "E' stato inserito un valore errato,
 inserire un valore tra 0 e 13"<<endl;
}
```

```
}while(out>nvert-1);


for(int ij=0; ij<=nvert; ij++)
{
m1.push_back(tran1[ij]);
m2.push_back(tran2[ij]);
m3.push_back(tran3[ij]);
zx.push_back(zxx[ij]);
}
Tran.push_back(m1);
Tran.push_back(m2);
Tran.push_back(m3);
Tran.push_back(zx);

int Pos=0;
cout << " \n";

for(int ii=1; ii<=3; ii++)
{
cout << "Il mezzo di trasporto " << ii << "
e' disponibile? yes [Y] - no [N] ";
char answer;
cin >> answer;

if (answer=='N' || answer == 'n') {
cout << "\n Mezzo di trasporto " << ii << "
NON aggiunto  \n";

AvTran.push_back(0);
continue; // va al prossimo mezzo
}
if (answer=='Y' || answer == 'y') {
cout << "\n Mezzo di trasporto " << ii << "  aggiunto \n\n";


if(ii==1){
AvTran.push_back(ii);
do{
cout << "Inserire il nodo in cui si trova il
trasportatore '" << ii <<"' : " <<  endl;
```

```
for (int ij = Pos; ij < nvert; ++ij)
std::cout << sel[ij] << std::endl;

cout << "Inserire il nodo scelto: ";
cin >> Pos;
cout << "\n";
if(Pos>nvert-1)
{
cout << "E' stato inserito un valore errato"<<endl;
}
}while(Pos>nvert-1);

if((in==0)||(in==1)||(in==2)||(in==3)||(in==4)
||(in==5)||(in==6)||(in==9))
{
PT.push_back(Pos);
segna.push_back(ii);

}
else {cout << "Non è possibile raggiungere
il nodo di ingresso con il mezzo di trasporto selezionato"
<<endl;
}

}

if(ii==2){
AvTran.push_back(ii);
do{
cout << "Inserire il nodo in cui si trova il
trasportatore '" << ii <<"' : " <<  endl;

for (int ij = Pos+1; ij < nvert; ++ij)
std::cout << sel[ij] << std::endl;

cout << "Inserire il nodo scelto: ";
cin >> Pos;
cout << "\n";
if(Pos>nvert-1)
{
cout << "E' stato inserito un valore errato"<<endl;
```

```
}
}while(Pos>nvert-1);

if((in==2)||(in==5)||(in==9)||(in==12)||(in==13))
{
PT.push_back(Pos);
segna.push_back(ii);
}
else {cout << "Non è possibile raggiungere il
nodo di ingresso con il mezzo di trasporto selezionato"
<<endl;
}
}

if(ii==3){
AvTran.push_back(ii);
do{
cout << "Inserire il nodo in cui si trova il
 trasportatore '" << ii <<"' : " <<  endl;

for (int ij = 7; ij < 12; ++ij)
std::cout << sel[ij] << std::endl;

cout << "Inserire il nodo scelto: ";
cin >> Pos;
cout << "\n";
if((Pos<7)||(Pos>12))
{
cout << "E' stato inserito un valore errato"
<<endl;
}
}while((Pos<7)||(Pos>12));



if((in==7)||(in==8)||(in==9)||(in==10)||(in==11))
{
PT.push_back(Pos);
segna.push_back(ii);
}
else {cout << "Non è possibile raggiungere il nodo
```

```
 di ingresso con il mezzo di trasporto selezionato"
 <<endl;
}
}

}
}

NewGraphPos(in, Tran, m1, m2, m3, zx,
obj_value, pos_tran, AvTran, PT);

se (name, cgr, AvTran, G0, G1, G2, G3, GZ);

graph_traits < Graph >::edge_iterator ei, ei_end;
property_map<Graph, int EdgeProperties::*>
::type weight_pmap = get(&EdgeProperties::weight, G0);


int ii = 0;
for (boost::tie(ei, ei_end) = edges(G0); ei != ei_end; ++ei, ++ii)
weight_pmap[*ei] = peso[ii];



std::vector<int> distance1(Nvert, (std::numeric_limits < short >
::max)());
std::vector<std::size_t> parent1(Nvert);


for (int ii = 0; ii < Nvert; ++ii)
parent1[ii] =ii;



vector<int>save_obj;



for(int jj=0; jj<pos_tran.size(); ++jj)
{
for (size_t kj=0; kj<Nvert; ++kj)
```

```
{
distance1[kj]=(std::numeric_limits
 < short >::max)();
}

distance1[pos_tran[jj]] = 0;


bool kk = bellman_ford_shortest_paths
(G0, int (Nvert), weight_map(weight_pmap).
distance_map(&distance1[0]).
 predecessor_map(&parent1[0]));


if (!kk)
{
std::cout << "negative cycle" << std::endl;

}

else
{

if(obj_value[jj] != -1){
save_obj.push_back(distance1[obj_value[jj]]);

std::vector<int> cammino;
cammino.clear();

std::cout << "\n Distanza dalla posizione del trasportatore '"
<< jj+1 << "' al nodo di ingresso:"
<< distance1[obj_value[jj]] << " \n ";

}
}
}
int minimo=30000;
int start=0;


for(int kj=0; kj<obj_value.size(); ++kj)
```

```
{
if ((save_obj[kj]<minimo)&&(obj_value[kj] != -1))
{
minimo=save_obj[kj];
start=kj;
}
mez_sel=segna[start]-1;
}

if(minimo!=30000)
{

std::cout << "\n Mezzo di trasporto più vicino
al nodo di ingresso: " << mez_sel+1 << ". \n ";

std::cout << "\n Spazio percorso dal mezzo
più vicino: " << minimo << " metri. \n ";
}

else {
std::cout << "\n Non è possibile raggiungere
 il nodo destinazione.  \n ";
}

cout << " \n";


distance1.clear();

vector<int>traccia;

NewGraph(in, out, mez_sel, traccia, Tran, m1,
 m2, m3, in_value, out_value, AvTran);

std::vector<int> distance(Nvert, (std::numeric_limits
< short >::max)());
std::vector<std::size_t> parent(Nvert);


for (int ii = 0; ii < Nvert; ++ii)
parent[ii] =ii;
```

```
for(int jj=0; jj<in_value.size(); ++jj)
distance[in_value[jj]] = 0;

bool rrr = bellman_ford_shortest_paths
(G0, int (Nvert), weight_map(weight_pmap).
distance_map(&distance[0]).
 predecessor_map(&parent[0]));

if (!rrr)
{
std::cout << "negative cycle" << std::endl;

}

else
{
cout << "\n";

int minimo=30000;
int start=0;
int bid;

for(int ii=0; ii<out_value.size(); ii++)
{
if ((distance[out_value[ii]]<minimo)&&
(out_value[ii] >= 0))
{
minimo=distance[out_value[ii]];
start=ii;
}
}
bid=out_value[start];
std::vector<int> cammino;
cammino.clear();

int stop;
int key;

std::cout << "\n Valore del cammino : "
```

```
 << minimo << " \n ";
stop=parent[bid];


do{
key=parent[stop];
stop=key;}
while(distance[key]!=0);

do{
cammino.push_back(name[bid]);
bid=parent[bid];

}
while (name[bid]!= stop);

cammino.push_back(stop);
int cam;
cam=cammino.size()-1;

/////////

cout << "\n";

vector<int>transport;
vector<int>conta;

cout << "\n Nodi attraversati:" ;

for (int ik=cam; ik>=0; --ik)

cout << cammino[ik]<< " " ;

cout << "\n Numero nodi visitati: " << cammino.size()
 << endl;

if(cammino.size()==2)
{
cout << "\n Numero cambi mezzo di trasporto effettuati: 0"
<< endl;
```

```
}
else{

for(int ii=0; ii<=cam; ii++)
{
for(int jj=0; jj<m1.size(); ++jj)
{
if(cammino[ii]==m1[jj])

transport.push_back(1);
}
}
for(int ii=0; ii<=cam; ii++)
{
for(int jj=0; jj<m2.size(); ++jj)
{
if(cammino[ii]==m2[jj])

transport.push_back(2);
}
}
for(int ii=0; ii<=cam; ii++)
{
for(int jj=0; jj<m3.size(); ++jj)
{
if(cammino[ii]==m3[jj])

transport.push_back(3);
}
}
cam=transport.size()-1;
int ij=0;
do{
if(transport[ij]!=transport[ij+1])
{
conta.push_back(1);
}
ij=ij++;}
while(ij!=cam);
```

```
cout << "\n";

cout << "Lista mezzi di trasporto utilizzati: ";
for (int ik=0; ik<=cam; ++ik)
cout<< transport[ik]<< " ";

cout << "\n";

cout << "Numero cambi mezzo di trasporto effettuati: "
<< conta.size()<<endl;
cout << "\n";

}

std::cout << "\n Mezzo di trasporto utilizzato
per il primo spostamento è il mezzo: ': " << mez_sel+1
 << "'. \n ";
cout << "Il mezzo di trasporto utilizzato
per l'ultimo spostamento è il mezzo: '"
 <<AvTran[traccia[start]] << "' "<<endl;

}

std::ofstream dot_file("figs/bellman-eg.dot");
dot_file << "digraph D {\n"
<< "  rankdir=LR\n"
<< "  size=\"5,3\"\n"
<< "  ratio=\"fill\"\n"
<< "  edge[style=\"bold\"]\n" << "
 node[shape=\"circle\"]\n";

{
for (boost::tie(ei, ei_end) = edges(G0);
ei != ei_end; ++ei) {
graph_traits < Graph >::edge_descriptor ee = *ei;
graph_traits < Graph >::vertex_descriptor
B = source(ee, G0), C = target(ee, G0);
// VC++ doesn't like the 3-argument get function, so here
// we workaround by using 2-nested get()'s.
dot_file << name[B] << " -> " << name[C]
<< "[label=\"" << get(get(&EdgeProperties::weight, G0), ee)
```

```
<< "\"";
if (parent[B] == C)
dot_file << ", color=\"black\"";
else
dot_file << ", color=\"grey\"";
dot_file << "]";
}
}
dot_file << "}";
return EXIT_SUCCESS;
}

bool NewGraphPos (int in, vector<vector<int> >
Tran, vector<int>m1, vector<int>m2, vector<int>m3,
vector<int>zx, vector<int>&obj_value,
vector<int>&pos_tran, vector<int>AvTran, vector<int>PT)

{
for(int jj=0; jj<PT.size(); ++jj)
{
pos_tran.push_back(Tran[3][PT[jj]]);
cout << endl;
}

for(int ii=0; ii<AvTran.size(); ii++)
{
if(AvTran[ii]!=0)
{

obj_value.push_back(Tran[AvTran[ii]-1][in]);

}
}
return true;
}

bool NewGraph(int in, int out, int mez_sel,
vector<int>&traccia, vector<vector<int> >Tran,
vector<int>m1, vector<int>m2, vector<int>m3,
vector<int>&in_value, vector<int>&out_value,
 vector<int>AvTran)
```

```
{
for(int ii=0; ii<AvTran.size(); ii++)
{
if(AvTran[ii]!=0)
{
out_value.push_back(Tran[ii][out]);
traccia.push_back(ii);// tiene traccia del
mezzo scelto per dire il mezzo che uso
}
}

in_value.push_back(Tran[mez_sel][in]);

return true;
}

bool se (int name[], int cgr, vector<int>AvTran,
Graph &G0, Graph &G1, Graph &G2, Graph &G3, Graph &GZ)
{

if((AvTran[cgr] == 1) && (AvTran[cgr+1] == 0)
&& (AvTran[cgr+2] == 0))
{
add_vertex(0, G1);
add_vertex(1, G1);
add_vertex(2, G1);
add_vertex(3, G1);
add_vertex(4, G1);
add_vertex(5, G1);
add_vertex(6, G1);
add_vertex(7, G1);
add_vertex(18, GZ);
add_vertex(19, GZ);
add_vertex(20, GZ);
add_vertex(21, GZ);
add_vertex(22, GZ);
add_vertex(23, GZ);
add_vertex(24, GZ);
add_vertex(25, GZ);
add_vertex(26, GZ);
add_vertex(27, GZ);
```

```
add_vertex(28, GZ);
add_vertex(29, GZ);


}


if((AvTran[cgr] == 0) && (AvTran[cgr+1] == 2)
 && (AvTran[cgr+2] == 0))

{
add_vertex(8, G2);
add_vertex(9, G2);
add_vertex(10, G2);
add_vertex(11, G2);
add_vertex(12, G2);
add_vertex(19, GZ);
add_vertex(20, GZ);
add_vertex(21, GZ);
add_vertex(22, GZ);
add_vertex(23, GZ);
add_vertex(24, GZ);
add_vertex(25, GZ);
add_vertex(26, GZ);
add_vertex(27, GZ);
add_vertex(28, GZ);
add_vertex(29, GZ);
add_vertex(30, GZ);
add_vertex(31, GZ);

}


if((AvTran[cgr] == 0) && (AvTran[cgr+1] == 0)
 && (AvTran[cgr+2] == 3))

{

add_vertex(13, G3);
add_vertex(14, G3);
add_vertex(15, G3);
```

```
add_vertex(16, G3);
add_vertex(17, G3);
add_vertex(25, GZ);
add_vertex(26, GZ);
add_vertex(27, GZ);
add_vertex(28, GZ);
add_vertex(29, GZ);

}


if((AvTran[cgr] == 1) && (AvTran[cgr+1] == 2)
 && (AvTran[cgr+2] == 0))

{
add_vertex(0, G1);
add_vertex(1, G1);
add_vertex(2, G1);
add_vertex(3, G1);
add_vertex(4, G1);
add_vertex(5, G1);
add_vertex(6, G1);
add_vertex(7, G1);
add_vertex(8, G2);
add_vertex(9, G2);
add_vertex(10, G2);
add_vertex(11, G2);
add_vertex(12, G2);
add_vertex(18, GZ);
add_vertex(19, GZ);
add_vertex(20, GZ);
add_vertex(21, GZ);
add_vertex(22, GZ);
add_vertex(23, GZ);
add_vertex(24, GZ);
add_vertex(25, GZ);
add_vertex(26, GZ);
add_vertex(27, GZ);
add_vertex(28, GZ);
add_vertex(29, GZ);
```

```
}

if((AvTran[cgr] == 1) && (AvTran[cgr+1] == 0)
 && (AvTran[cgr+2] == 3))


{
add_vertex(0, G1);
add_vertex(1, G1);
add_vertex(2, G1);
add_vertex(3, G1);
add_vertex(4, G1);
add_vertex(5, G1);
add_vertex(6, G1);
add_vertex(7, G1);
add_vertex(13, G3);
add_vertex(14, G3);
add_vertex(15, G3);
add_vertex(16, G3);
add_vertex(17, G3);
add_vertex(18, GZ);
add_vertex(19, GZ);
add_vertex(20, GZ);
add_vertex(21, GZ);
add_vertex(22, GZ);
add_vertex(23, GZ);
add_vertex(24, GZ);
add_vertex(25, GZ);
add_vertex(26, GZ);
add_vertex(27, GZ);
add_vertex(28, GZ);
add_vertex(29, GZ);

}


if((AvTran[cgr] == 0) && (AvTran[cgr+1] == 2)
&& (AvTran[cgr+2] == 3))
```

```
{
add_vertex(8, G2);
add_vertex(9, G2);
add_vertex(10, G2);
add_vertex(11, G2);
add_vertex(12, G2);
add_vertex(13, G3);
add_vertex(14, G3);
add_vertex(15, G3);
add_vertex(16, G3);
add_vertex(17, G3);
add_vertex(19, GZ);
add_vertex(20, GZ);
add_vertex(21, GZ);
add_vertex(22, GZ);
add_vertex(23, GZ);
add_vertex(24, GZ);
add_vertex(25, GZ);
add_vertex(26, GZ);
add_vertex(27, GZ);
add_vertex(28, GZ);
add_vertex(29, GZ);
add_vertex(30, GZ);
add_vertex(31, GZ);
}

if((AvTran[cgr] == 1) && (AvTran[cgr+1] == 2)
&& (AvTran[cgr+2] == 3))
{
add_vertex(0, G1);
add_vertex(1, G1);
add_vertex(2, G1);
add_vertex(3, G1);
add_vertex(4, G1);
add_vertex(5, G1);
add_vertex(6, G1);
add_vertex(7, G1);
add_vertex(8, G2);
add_vertex(9, G2);
add_vertex(10, G2);
```

```
add_vertex(11, G2);
add_vertex(12, G2);
add_vertex(13, G3);
add_vertex(14, G3);
add_vertex(15, G3);
add_vertex(16, G3);
add_vertex(17, G3);
add_vertex(18, GZ);
add_vertex(19, GZ);
add_vertex(20, GZ);
add_vertex(21, GZ);
add_vertex(22, GZ);
add_vertex(23, GZ);
add_vertex(24, GZ);
add_vertex(25, GZ);
add_vertex(26, GZ);
add_vertex(27, GZ);
add_vertex(28, GZ);
add_vertex(29, GZ);
add_vertex(30, GZ);
add_vertex(31, GZ);

}

if((AvTran[cgr] == 0) && (AvTran[cgr+1] == 0)
&& (AvTran[cgr+2] == 0))
{
cout << "Non è stato selezionato alcun mezzo di trasporto";
}

return true;
}
```

Il file.txt che legge gli archi e i pesi è il seguente:

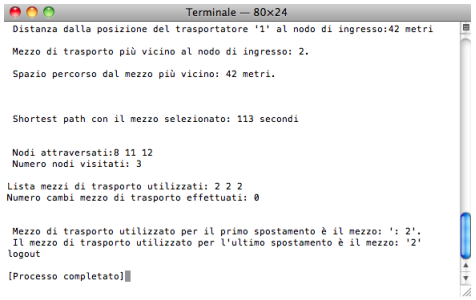
```
[GRAPH]
0 1 57
0 3 87
0 4 120
0 5 120
0 6 147
0 7 228
1 0 57
1 2 42
1 3 42
2 1 87
2 3 0
2 4 63
2 5 63
2 6 90
2 7 171
2 8 180
3 0 87
3 1 42
3 2 0
3 4 63
3 6 90
4 0 120
4 2 75
4 3 63
5 0 120
5 2 63
5 7 141
5 9 180
6 0 147
6 2 90
6 3 90
7 0 228
7 2 171
7 5 141
7 10 180
7 15 180
8 2 180
8 9 50
8 10 137
8 11 113
9 5 180
9 8 50
9 10 113
9 11 170
9 12 170
10 7 180
10 8 137
10 9 113
10 11 86
10 12 86
10 15 180
11 8 194
11 9 170
11 10 86
11 12 0
12 9 170
12 10 86
12 11 0
13 15 54
14 15 40
15 7 180
15 10 180
15 13 54
15 14 40
15 16 40
15 17 54
16 15 40
17 15 54
18 0 0
18 1 38
18 2 58
18 3 58
18 4 80
18 5 80
18 6 98
18 7 152
18 8 58
18 9 80
18 10 152
18 11 200
18 12 200
18 13 112
18 14 128
18 15 152
18 16 168
18 17 182
18 19 38
18 20 58
18 21 58
18 22 80
18 23 80
18 24 98
18 25 112
18 26 128
18 27 152
18 28 168
18 29 182
19 0 38
19 1 0
19 2 28
19 3 28
19 4 50
19 5 50
19 6 68
19 7 122
19 8 28
19 9 50
19 10 122
19 11 170
19 12 170
19 13 82
19 14 98
19 15 122
19 16 138
19 17 152
19 18 38
19 20 28
19 21 28
19 22 50
19 23 50
19 24 68
19 25 82
19 26 98
19 27 122
19 28 138
19 29 152
19 30 170
19 31 170
20 0 58
20 1 28
20 2 0
20 3 0
20 4 42
20 5 42
20 6 60
20 7 114
20 8 0
20 9 42
20 10 114
20 11 162
20 12 162
20 13 74
20 14 90
20 15 114
20 16 130
20 17 144
20 18 58
20 19 28
20 21 0
20 22 42
20 23 42
20 24 60
20 25 74
20 26 90
20 27 114
```

20 28 130
20 29 144
20 30 162
20 31 162
21 0 58
21 1 28
21 2 0
21 3 0
21 4 42
21 5 42
21 6 60
21 7 114
21 8 0
21 9 42
21 10 114
21 11 162
21 12 162
21 13 74
21 14 90
21 15 114
21 16 130
21 17 144
21 18 58
21 19 28
21 20 0
21 22 42
21 23 42
21 24 60
21 25 74
21 26 90
21 27 114
21 28 130
21 29 144
21 30 162
21 31 162
22 0 80
22 1 50
22 2 42
22 3 42
22 4 0
22 5 0
22 6 40
22 7 94
22 8 42
22 9 0
22 10 94
22 11 142
22 12 142
22 13 54
22 14 70
22 15 94
22 16 110
22 17 124
22 18 80
22 19 50
22 20 42
22 21 42
22 23 0
22 24 40
22 25 54
22 26 70
22 27 94
22 28 110
22 29 124
22 30 142
22 31 142
23 0 80
23 1 50
23 2 42
23 3 42
23 4 0
23 5 0
23 6 40
23 7 94
23 8 42
23 9 0
23 10 94
23 11 142
23 12 142
23 13 54
23 14 70
23 15 94
23 16 110
23 17 124
23 18 80
23 19 50
23 20 42
23 21 42
23 22 0
23 24 40
23 25 54
23 26 70
23 27 94
23 28 110
23 29 124
23 30 142
23 31 142
24 0 98
24 1 68
24 2 60
24 3 60
24 4 40
24 5 40
24 6 30
24 7 72
24 8 60
24 9 40
24 10 72
24 11 120
24 12 120
24 13 32
24 14 48
24 15 72
24 16 88
24 17 102
24 18 98
24 19 68
24 20 60
24 21 60
24 22 40
24 23 40
24 25 32
24 26 48
24 27 72
24 28 88
24 29 102
24 30 120
24 31 120
25 0 112
25 1 82
25 2 74
25 3 74
25 4 54
25 5 54
25 6 32
25 7 54
25 8 74
25 9 54
25 10 54
25 11 102
25 12 102
25 13 0
25 14 30
25 15 54
25 16 70
25 17 84
25 18 112
25 19 82
25 20 74
25 21 74
25 22 54
25 23 54
25 24 32
25 26 30
25 27 54
25 28 70
25 29 84
25 30 102
25 31 102
26 0 128
26 1 98
26 2 90
26 3 90
26 4 70

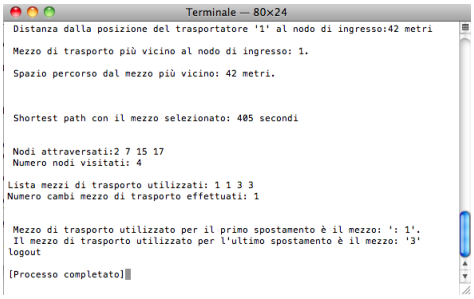
26 5 70
26 6 48
26 7 40
26 8 90
26 9 70
26 10 40
26 11 88
26 12 88
26 13 30
26 14 0
26 15 40
26 16 56
26 17 70
26 18 128
26 19 98
26 20 90
26 21 90
26 22 70
26 23 70
26 24 48
26 25 30
26 27 40
26 28 56
26 29 70
26 30 88
26 31 88
27 0 152
27 1 122
27 2 114
27 3 114
27 4 94
27 5 94
27 6 72
27 7 0
27 8 114
27 9 94
27 10 0
27 11 72
27 12 72
27 13 54
27 14 40
27 15 0
27 16 40
27 17 54
27 18 152
27 19 122
27 20 114
27 21 114
27 22 94
27 23 94
27 24 72
27 25 54
27 26 40
27 28 40
27 29 54
27 30 72
27 31 72
28 0 168
28 1 138
28 2 130
28 3 130
28 4 110
28 5 110
28 6 88
28 7 40
28 8 130
28 9 110
28 10 40
28 11 48
28 12 48
28 13 70
28 14 56
28 15 40
28 16 0
28 17 30
28 18 168
28 19 138
28 20 130
28 21 130
28 22 110
28 23 110
28 24 88
28 25 70
28 26 156
28 27 40
28 29 30
28 30 48
28 31 48
29 0 182
29 1 152
29 2 144
29 3 144
29 4 124
29 5 124
29 6 102
29 7 54
29 8 144
29 9 124
29 10 54
29 11 32
29 12 32
29 13 184
29 14 70
29 15 54
29 16 30
29 17 0
29 18 182
29 19 152
29 20 144
29 21 144
29 22 124
29 23 124
29 24 102
29 25 84
29 26 70
29 27 54
29 28 30
29 30 32
29 31 32
30 0 200
30 1 170
30 2 162
30 3 162
30 4 142
30 5 142
30 6 120
30 7 72
30 8 162
30 9 142
30 10 72
30 11 0
30 12 0
30 13 102
30 14 88
30 15 72
30 16 48
30 17 32
30 19 170
30 20 162
30 21 162
30 22 142
30 23 142
30 24 120
30 25 102
30 26 88
30 27 72
30 28 48
30 29 32
30 31 0
31 0 200
31 1 170
31 2 162
31 3 162
31 4 142
31 5 142
31 6 120
31 7 72
31 8 162
31 9 142
31 10 72
31 11 0
31 12 0
31 13 102
31 14 88

31	15	72	31	20	162	31	24	120	31	28	48
31	16	48	31	21	162	31	25	102	31	29	32
31	17	32	31	22	142	31	26	88	31	30	0
31	19	170	31	23	142	31	27	72			

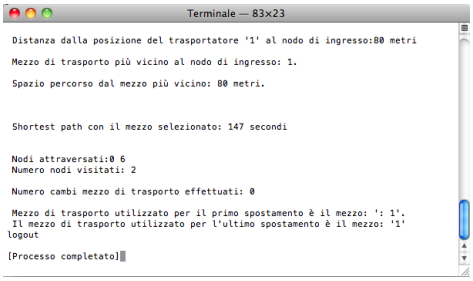
Seguono in Figura B.1, ulteriori esempi di utilizzo del programma per la validazione del modello.



(a) Da 2 a 13 con m2 ed m3



(b) Da 3 a 11con m1 ed m3



(c) Da 0 a 6 con m1 ed m2

Figura B.1: Ulteriori esempi di compilazione per la validazione del programma in $C++$.

Bibliografia

[1] W. Terkaj, G. Pedrielli, M. Sacco, "Virtual Factory Data Model", in Submitted to 2nd OSEMA (Ontology and Semantic Web for Manufacturing) Workshop, 2012.

[2] M. Sacco, P. Pedrazzoli, W. Terkaj, "VFF: Virtual Factory Framework", in Proceedings of ICE - 16th International Conference on Concurrent Enterprising, Lugano, Svizzera.

[3] Official Web Site of Virtual Factory Framework, http://www.vff-project.eu/ .

[4] S. Terzi, "Strumenti e Funzionalità del PLM", paper of lesson "Product Lifecycle Management", Politecnico di Milano, p 11-13, 2013.

[5] G. Ghielmini, P. Pedrazzoli, D. Rovere, W. Terkaj, G. Dal Maso, F. Milella, M. Sacco, C.R. Boer, "Virtual Factory Manager of Semantic Data", in Proceedings of DET2011 7th International Conference on Digital Enterprise Technology, Athens, Greece, 2011.

[6] M. Sacco, G. Dal Maso, F. Milella, P. Pedrazzoli, D. Rovere, W. Terkaj, "Virtual Factory Manager", in Lecture Notes in Computer Science, Ed: Springer, 2011, pp. 397-406.

[7] W. Terkaj, M. Urgo, "Virtual Factory Data Model to support Performance Evaluation of Production Systems",

[8] "International Society of Automation. ISA-95: the international standard for the integration of enterprise and control systems", [Online], http://www.isa-95.com/

[9] "The Organization for Production Technology", Business To Manufacturing Markup Language (B2MML), [Online] www.wbf.org/associations/12553/files/B2MML-BatchML%20v0500%20Schemas-Word-PDF.zip , 2011.

[10] W3C "XML Schema Part 1: Structures Second Edition", [Online]. http://www.w3.org/TR/xmlschema-1/ , 2004

[11] buildingSMART. "Industry Foundation Classes - IFC2x Edition 4 Release Candidate 2, " [Online]. http://buildingsmart-tech.org/ifc/IFC2x4/rc2/html/index.htm .

[12] International Organization for Standardization, ISO 14649 - Industrial automation systems and integration – Physical device control – Data model for computerized numerical controllers, International Organization for Standardization, 2004.

[13] International Organization for Standardization, ISO 10303-11:2004 Industrial automation systems and integration – Product data representation and exchange – Part 11: Description methods: The EXPRESS language reference manual, 2004.

[14] W3C. "OWL 2 Web Ontology Language - Document Overview", [Online]. http://www.w3.org/TR/owl2- overview/ .

[15] W3C. "RDF/XML Syntax Specification (Revised)", [Online]. http://www.w3.org/TR/REC-rdf- syntax/ 2004.

[16] W3C. "OWL Web Ontology Language - Use Cases and Requirement", [Online]. http://www.w3.org/TR/webont-req/#onto-def .

[17] E. Padoano, "La movimentazione dei materiali (material handling)", paper of lesson "Impianti industriali", Università degli studi di Trieste, p 8, 2010.

[18] F. Caron, G. Merchet, R. Wegner, "Impianti di movimentazione e stoccaggio dei materiali", ed. Ulrico Hoepli Editore Spa, Milano, pp 29-65, 2001.

[19] C. Vercellis, "Modelli e decisioni", ed. Società Editrice Esculapio, Bologna, pp 241-244, 1997.

[20] E. Lawler, "Combinatorial Optimization: Networks and Matroids", New York : Holt, Rinehart and Winston, pp. 20-31, 1976.

[21] Librerie Boost, "The Boost Graph Library (BGL)", [Online], http://www.boost.org/doc/libs/1_53_0/libs/graph/doc/index.html

[22] R. Bellman, "On a Routing Problem", Quarterly of Applied Mathematics, pp. 87-90, 1958.

[23] L.R. Ford and D. R. Fulkerson “Flows in networks”, Princeton University Press, 1962.

[24] T. Cormen, C. Leiserson,, R. Rivest, “Introduction to Algorithms’, McGraw-Hill, 1990.

[25] Boost::Graph.“subgraph<Graph>”,[Online], http://www.boost.org/doc/libs/1_38_0/libs/graph/doc/subgraph.html .

[26] L. Lutterotti “Laminazione” paper of lesson “Tecnologie e sistemi di lavorazione I ”, 2006.

[27] S. Kalpakjian, S.R. Schmid “Tecnologia meccanica”, Prentice Hall, cap. 6, 2008.

[28] T.A.M. Tolio, M. Urgo, “FMS” paper of lesson “Sistemi Integrati di Produzione”, Dipartimento di Meccanica - Sezione Tecnologie Meccaniche e Produzione, Politecnico di Milano, 2009.