ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA

Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

Fancy Title

Tesi di laurea in:
Supervisor's course name

Relatore
Prof. Supervisor Here

Candidato
Candidate Name

Correlatori
Dott. CoSupervisor 1
Dott. CoSupervisor 2

I Sessione di Laurea
Anno Accademico 2022-2023

Abstract

Max 2000 characters, strict.

Optional. Max a few lines.

Contents

List of Figures

List of Listings

Chapter 1

Introduction

Tradizionali approcci di ingegneria del software mostrano limiti crescenti nel rispondere alle esigenze dinamiche del business moderno, e numerosi studi empirici suggeriscono che l'agilità rappresenta una delle soluzioni più promettenti per superarli [**?**, **?**]. I metodi Agile, tra cui *Scrum* ed *eXtreme Programming* (XP), si fondano sui dodici principi del Manifesto Agile [**?**] e sono ormai adottati da un numero sempre maggiore di aziende allo scopo di rendere lo sviluppo software più rapido, adattivo e focalizzato sul valore. In particolare, il metodo Scrum è stato ampiamente discusso in letteratura [**?**], che ne descrive le idee fondamentali e il ciclo di vita operativo.

Un elemento comune ai metodi Agile è la progettazione e implementazione incrementale, in cui il software è suddiviso in funzionalità specifiche (*user stories*) e, ad ogni iterazione (*sprint*), il team è chiamato a fornire il set di user stories che massimizza il valore per l'utente, soddisfacendo al tempo stesso vincoli di durata, dipendenza e relazioni tra le storie. Secondo il principio della *team awareness*, la pianificazione dello sprint si basa sulla condivisione e la mediazione delle stime fornite dai membri del team riguardo complessità delle storie, utilità, rischio di mancata consegna e dipendenze. Avanzare storie ad alto valore può anticipare risultati significativi per l'utente, rafforzando la consapevolezza del team; analogamente, anticipare storie critiche consente di limitare rischi di impatto tardivo, bilanciando però una maggiore probabilità di ritardi nelle fasi iniziali. Il contributo di [**?**] offre indicazioni pratiche per valutare complessità e valore delle user

stories.

Una pianificazione di sprint efficace è unanimemente riconosciuta come fattore chiave per il successo di un progetto agile [**?**]. L'efficacia del piano dipende sia dalla precisione delle stime (aspetto legato all'esperienza del team), sia dalla capacità di considerare correttamente variabili e vincoli progettuali: quest'ultimo obiettivo si traduce in un problema di ottimizzazione la cui complessità cresce con la dimensione del progetto, e il cui mancato raggiungimento può generare inefficienze, extracosti e ritardi.

Questa tesi prende come riferimento il lavoro [**?**], su cui vengono sviluppate e approfondite nuove tecniche di ottimizzazione e strumenti matematici per il supporto alla sprint planning in ambiente Agile.

Chapter 2

Descrizione del problema

Poiché la soddisfazione del cliente rappresenta uno dei principali indicatori di successo di un progetto, i metodi agili si propongono di rispondere in modo più efficace alle esigenze degli utenti, riducendo i tempi di consegna e aumentando la flessibilità del processo di sviluppo. Accelerare il time-to-market consente infatti di ridurre la pressione competitiva, mentre la flessibilità permette di adattarsi rapidamente sia ai progressi tecnologici sia ai cambiamenti nei requisiti degli utenti. Per conseguire tali obiettivi, l'approccio agile adotta una serie di pratiche complementari, tra cui lo sviluppo incrementale e iterativo, il coinvolgimento continuo degli utenti, la consapevolezza del team, e una documentazione leggera ma efficace. Dall'esperienza dei professionisti del settore sono nati diversi framework agili basati su questi principi. Tra questi, Scrum e gli approcci ibridi ,che combinano pratiche di Scrum ed eXtreme Programming (XP), risultano i più diffusi nelle aziende: Scrum si focalizza sugli aspetti organizzativi e di gestione del processo, mentre XP introduce pratiche tecniche specifiche, come la programmazione in coppia, per migliorare la qualità e l'efficienza dello sviluppo del codice.

Il ciclo di vita di un progetto Scrum è suddiviso in varie fasi. Durante la definizione delle user story, il team di sviluppo e gli utenti collaborano per delineare la struttura generale del sistema e individuare un insieme di funzionalità. Una user story rappresenta una funzionalità di piccole dimensioni ma di elevato valore per l'utente [**?**], descritta in modo semplice e sintetico. Essa costituisce una specifica leggera, che può essere ulteriormente approfondita attraverso un dialogo continuo

con l'utente, ma deve al contempo contenere informazioni sufficienti a permettere al team di stimarne la complessità di sviluppo e pianificarne l'implementazione in modo efficace.

Durante la fase di stima delle user story, a ciascuna storia vengono attribuiti due principali parametri: l'utilità e la complessità. Le valutazione possono essere effettuate sulla base di esperienze pregresse, pareri di esperti o tramite tecniche basate sul consenso come il planning poker. L'utilità rappresenta il valore per l'azienda percepito dall'utente che ha definito la user story. In alcuni casi è sufficiente stabilire un ordine di priorità tra le storie, mentre in altri l'utilità viene quantificata mediante un punteggio numerico. La complessità di sviluppo, invece, viene espressa in story point, un'unità adimensionale che viene preferita rispetto a misure di tempo o risorse per ridurre la soggettività e favorire la comparabilità tra le stime. L'assegnazione degli story point è effettuata dai membri del team in base alla loro esperienza, alla conoscenza del dominio e alle caratteristiche specifiche del progetto. Sono utilizzabili diverse tecniche di dimensionamento [**?**] tra cui l'utilizzo di scale numeriche, come la sequenza di Fibonacci, intervalli di valori o l'adozioni di sistemi che si basano su categorie qualitative, come l'utilizzo delle taglie delle magliette.

Durante la fase di prioritizzazione delle user story, il team assegna a ciascuna storia un livello di priorità basato principalmente sulla sua utilità e sul rischio associato, oltre a identificare le eventuali dipendenze tra le diverse storie. Vi sono due categorie di fattori di rischio, vi sono le storie critiche: si tratta di user story che esercitano un forte impatto sulle altre e rappresentano componenti fondamentali del progetto. Una scelta errata nell'assegnazione o nella stima di queste storie può compromettere il successo complessivo dello sprint e dei cicli successivi, creando effetti a cascata su altre dipendenze. E vi sono le storie incerte, caratterizzate da un elevato grado di difficoltà nella stima del tempo e dello sforzo richiesto. Questa incertezza nasce da potenziali imprevisti, come cambiamenti nei requisiti espressi dall'utente, problemi tecnici inaspettati durante l'implementazione, o complessità che emergeranno solo durante lo sviluppo. Entrambe le categorie di rischio vengono generalmente rappresentate tramite valori numerici. Le dipendenze, invece,

esprimono vincoli di sequenza nello sviluppo, indicando che una user story può essere avviata solo dopo il completamento di una o più storie precedenti. Sebbene i metodi agili tendano a limitare le dipendenze per mantenere elevata la flessibilità del progetto, alcune di esse risultano inevitabili e devono essere rispettate per garantire la coerenza del processo di sviluppo.

L'elenco delle user story, una volta definite le priorità, costituisce il product backlog, che viene poi suddiviso in sprint durante la fase di pianificazione dello sprint. Gli sprint devono avere una durata breve e costante, solitamente compresa tra una e le quattro settimane, in modo da assicurare un feedback rapido e continuo da parte degli utenti. L'assegnazione delle user story agli sprint si basa su tre elementi fondamentali: la velocità di sviluppo del team, la complessità delle singole story e le possibili correlazioni o affinità tra le story. La velocità di sviluppo rappresenta il numero stimato di story point, che rappresenta l'unita minima per stimare lo sforzo necessario a completare una user story o un'attività, che il team è in grado di completare giornalmente. Questo valore viene utilizzato per calcolare la capacità dello sprint, ovvero il numero massimo di story point che il team può realisticamente erogare durante l'intera durata dello sprint. Infine, l'affinità tra story si riferisce alle correlazioni logiche o funzionali tra user story che, se incluse nello stesso sprint, generano un valore maggiore per l'utente finale. Quando story correlate vengono realizzate insieme, gli utenti percepiscono più chiaramente il valore complessivo della funzionalità erogata e possono sperimentare un'esperienza più coerente e completa. Considerare l'affinità nella pianificazione permette quindi di massimizzare l'utilità percepita e di ottimizzare l'impatto di ogni rilascio incrementale.

Durante la fase di sviluppo dello sprint gli sviluppatori eseguono un'analisi dettagliata delle user story selezionate e producono lo sprint backlog, ovvero l'elenco operativo delle attività da completare. Successivamente, ogni membro del team si occupa di un sottoinsieme di story, seguendo l'intero ciclo di progettazione, implementazione e test. Al termine dello sprint si svolge la sprint review, durante la quale gli utenti e gli stakeholder esaminano le story completate per verificare che le funzionalità implementate corrispondano effettivamente ai requisiti richiesti.

Le story che superano la revisione vengono considerate completate e consegnate agli utenti come incremento del prodotto, mentre quelle non approvate vengono reinserite nel product backlog. In questa fase conclusiva, il team conduce anche una retrospettiva, analizzando i problemi riscontrati durante lo sprint e le soluzioni adottate, al fine di identificare opportunità di miglioramento per le iterazioni successive. Questo può includere, ad esempio, l'aggiornamento delle stime sulla base dell'esperienza maturata. Se dal feedback degli utenti emergono nuove user story o modifiche significative a quelle esistenti, oppure se le stime vengono riviste in modo sostanziale, viene eseguita una nuova fase di prioritizzazione del backlog prima dell'avvio dello sprint successivo, garantendo così che la pianificazione rimanga sempre allineata alle reali esigenze del progetto.

Il problema della pianificazione dello sprint può essere descritto come dato un insieme di user story e un insieme di sprint, allocare ogni story a uno sprint in modo da soddisfare tutti i vincoli relativi alla capacità dello sprint e alle dipendenze tra le storie e raggiungere anche come obbiettivi la soddisfazione del cliente, che può essere raggiunta fornendo prima le user story con maggiore utilità per aumentare la consapevolezza e la fiducia dell'utente e includendo storie affini nello stesso sprint per aumentarne il valore per gli utenti, la gestione del rischio ottenibile promuovendo le user story critiche consentendo di identificare tempestivamente eventuali problemi o effetti collaterali che potrebbero propagarsi ad altre funzionalità, evitando impatti negativi tardivi e distribuendo le user story incerte in diversi sprint e posticipandole riducendo il rischio che più incertezze si concentrino nello stesso sprint, minimizzando la probabilità di ritardi nella consegna e garantendo una maggiore stabilità e prevedibilità del ciclo di sviluppo.

Il problema della pianificazione dello sprint può essere formulato come un Generalized Assignment Problem (GAP) con vincoli laterali. Questo tipo di problema consiste nell'assegnare un insieme di compiti a un insieme di risorse con l'obiettivo di ottimizzare una funzione (nel nostro caso, massimizzare l'utilità totale), rispettando i limiti di capacità di ciascuna risorsa e soddisfacendo vincoli aggiuntivi che impongono condizioni specifiche sulle assegnazioni.

Nel contesto della pianificazione agile, gli sprint rappresentano le risorse (o

zaini), mentre le user story costituiscono i compiti (o elementi) da assegnare. Ogni user story è caratterizzata da due attributi fondamentali: gli story point, che misurano il peso dell'elemento in termini di sforzo e complessità richiesti, e l'utilità, che ne rappresenta il valore per il progetto e per l'utente finale. La capacità dello sprint corrisponde al limite di capacità della risorsa ed è determinata dal numero massimo di story point che il team può completare, calcolato in base alla durata prevista dello sprint e alla velocità di sviluppo del team stesso. Questa capacità rappresenta quindi il vincolo principale che regola quante e quali user story possono essere assegnate a ciascun sprint.

La funzione obiettivo da massimizzare è l'utilità cumulativa dell'intero progetto. L'utilità di ogni singola user story viene incrementata se alcune storie affini sono incluse nello stesso sprint, poiché questo aumenta il valore percepito dall'utente, e/o se la storia è classificata come critica, riconoscendone l'importanza strategica per il successo del progetto.

Per quanto riguarda la gestione del rischio, nella formulazione del vincolo di capacità, gli story point delle user story vengono aumentati in proporzione alla loro incertezza. Questo meccanismo penalizza l'inclusione di due o più storie incerte nello stesso sprint (obiettivo 2-ii), favorendo una loro distribuzione equilibrata su sprint diversi e riducendo così il rischio di ritardi nella consegna.

È importante sottolineare che il Problema di Assegnazione Generalizzato appartiene alla classe dei problemi NP-hard, il che significa che trovare una soluzione ottimale richiede tempi computazionali che crescono esponenzialmente con le dimensioni del problema, rendendo necessario l'impiego di algoritmi euristici o metaeuristici per progetti di scala reale.

Chapter 3

stato dell'arte

I principi Agile hanno registrato una diffusione crescente nell'ultimo decennio, stimolando lo sviluppo di numerosi approcci metodologici proposti tanto dai professionisti quanto dalla comunità di ricerca [**?**]. Una revisione sistematica presentata in letteratura [**?**] confronta diversi metodi Agile analizzandone sia le caratteristiche organizzative che quelle tecniche, evidenziando in particolare la crescente adozione delle pratiche Scrum e XP nei contesti industriali. L'approccio Scrum è approfondito ulteriormente in [**?**], dove vengono descritti i fondamenti metodologici e il ciclo di vita caratteristico, contribuendo così alla comprensione dei meccanismi e delle peculiarità che contraddistinguono questo framework.

Per quanto concerne la stima del valore del software, la letteratura propone molteplici approcci senza tuttavia convergere verso un modello condiviso. Nel contesto più ampio dell'ingegneria del software basata sul valore, in [**?**] viene introdotta una matrice di decomposizione del valore che integra tre aspetti del software (tecnologia, progettazione e artefatto) con tre componenti del valore (valore intrinseco, esternalità e valore dell'opzione), fornendo in ciascuna cella domande specifiche che guidano gli analisti nell'interpretazione delle diverse combinazioni. Alternativamente, in [**?**] viene proposto un modello per esprimere il valore del software in termini monetari, sfruttando la relazione tra fattori tecnologici e di mercato. Una sintesi più completa è fornita da [**?**], che distinguono quattro prospettive di valore: finanziaria, cliente, processo aziendale interno e innovazione e apprendimento. Gli autori sviluppano una Software Value Map (SVM) in cui ciascuna prospettiva è

scomposta nelle sue componenti e sottocomponenti principali, offrendo agli analisti uno strumento per costruire una comprensione condivisa del valore. Sulla base della SVM, propongono inoltre un modello pratico articolato in tre fasi per ottenere la stima finale dell'utilità: selezione dello scenario caratterizzante il progetto specifico, identificazione del pattern di componenti di valore che meglio si adatta allo scenario selezionato, e definizione di una valutazione quantitativa del pattern attraverso la stima delle componenti individuate.

Nell'ampio contesto della pianificazione dei progetti, la maggior parte degli sforzi di ricerca degli ultimi anni si è concentrata sullo sviluppo di procedure esatte o euristiche per la generazione di una pianificazione di base praticabile in ambiente deterministico, con numerosi modelli e algoritmi proposti in letteratura come documentato nella panoramica di [**?**]. La complessità aumenta significativamente quando un team deve pianificare simultaneamente più progetti, situazione frequente nella pratica che richiede approcci specifici come evidenziato in [**?**]. In questo scenario, [**?**] distingue due livelli di pianificazione: il primo livello, denominato pianificazione approssimativa della capacità, affronta il problema di pianificazione a medio termine, mentre il secondo livello, chiamato pianificazione di progetto con vincoli di risorse, si occupa della pianificazione operativa a breve termine. Questo approccio a due livelli, esplorato anche in [**?**], utilizza una scomposizione top-down delle attività in grandi pacchetti di lavoro per ridurre la complessità della pianificazione a medio termine.

Secondo le classificazioni dei problemi di pianificazione di progetto proposte in [**?**] e [**?**], il problema affrontato può essere classificato come vincolato dalle risorse, con risorse rinnovabili (quali la manodopera) disponibili periodo per periodo. Analogamente al modello PERT/CPM di base, vengono considerate precedenze di tipo finish-to-start con sfasamento temporale nullo e non è consentita alcuna prelazione sulle attività. La funzione obiettivo adottata differisce tuttavia dall'approccio tradizionale: anziché minimizzare la durata complessiva del progetto, si persegue una misura di completamento incrementale che mira a massimizzare il valore aziendale percepito dagli utenti, riflettendo una prospettiva orientata al valore piuttosto che alla sola efficienza temporale.

Il problema della pianificazione ha acquisito crescente interesse nell'ambito delle metodologie iterative e incrementali. In [**?**] viene proposta una strategia di sviluppo software basata su fattori finanziari, applicabile in contesti iterativi, che genera una sequenza ottimale di requisiti massimizzando nel tempo il valore attuale netto, ovvero una combinazione di ricavi, costi e rischi di ciascun requisito. Gli autori presentano due strategie risolutive: un algoritmo greedy che seleziona il successivo requisito da soddisfare considerando quelli senza precursori non soddisfatti e con il massimo valore attuale netto, e un approccio look-ahead che estende la strategia greedy analizzando sottoinsiemi di sequenze di precedenza redditizie.

Un modello specificamente orientato ai metodi agili è descritto in [**?**], dove viene presentato un modello concettuale per la pianificazione dei rilasci e sviluppato un modello di ottimizzazione finalizzato all'assegnazione dei requisiti alle diverse iterazioni di un rilascio, con l'obiettivo di massimizzare l'utilità complessiva fornita pur tenendo conto delle precedenze e delle condizioni di accoppiamento. In questo contesto, gli autori illustrano un algoritmo branch-and-bound per la risoluzione del modello, che incorpora esplicitamente la gestione del rischio. Un ulteriore contributo rilevante nel contesto agile è quello di [**?**], focalizzato primariamente sulla gestione del rischio e dell'incertezza nei progetti XP. L'approccio proposto stima la velocità di sviluppo del team e considera più set di user story con rilevanza decrescente secondo la classificazione MoSCoW:

- set *must have* (requisiti indispensabili)
- set *should have* (requisiti importanti)
- set *could have* (requisiti desiderabili)

L'obiettivo è assegnare ciascuna user story al set più appropriato, massimizzando l'utilità complessiva e rispettando le precedenze e gli accoppiamenti tra le user story mediante un algoritmo branch-and-bound per identificare la soluzione ottimale. Tuttavia, il numero limitato di set considerati determina un piano a grana grossa che necessita di successivo raffinamento per ottenere una pianificazione operativa, ad esempio suddividendo i set secondo limiti di budget o frammentando le user story in attività più granulari.

In un precedente lavoro è stato inoltre proposto un approccio per la pianificazione dello sprint di base nei progetti di data warehouse agile [**?**]. Il modello presentato in quel contributo si distingue per una struttura più semplice rispetto a quella qui descritta, non includendo storie forzate e adottando una modellazione dell'accoppiamento meno espressiva; inoltre, il problema della ripianificazione non viene affrontato in quella formulazione.

Nei contesti reali, l'ambiente di progetto difficilmente può essere considerato deterministico a causa della varietà di eventi imprevisti che possono verificarsi e dell'imprecisione intrinseca delle stime, rendendo necessarie politiche di gestione del cambiamento. Sebbene nessuno dei lavori precedentemente menzionati si occupi specificamente di gestione del cambiamento, un approccio rilevante in questa direzione è rappresentato in [**?**], orientato a contesti iterativi e incrementali. In questo framework, un piano di rilascio include diversi incrementi e in ogni fase un insieme di requisiti viene assegnato agli incrementi attuali e futuri in modo da restituire il miglior compromesso tra le priorità degli stakeholder e i vincoli di sviluppo, quali capacità di incremento, precedenze e condizioni di accoppiamento. Il modello è formalizzato come un problema di zaino multiplo e risolto mediante un algoritmo genetico. Per gestire il cambiamento, Evolve implementa una strategia parziale di ripianificazione: a ogni incremento sono consentiti nuovi requisiti e modifiche alle priorità o ai vincoli, e una nuova soluzione viene generata ex novo.

Nel contesto specifico della pianificazione Scrum, in [**?**] viene fornita una formulazione a zaino di un modello di ottimizzazione per la pianificazione a singola iterazione che seleziona i requisiti massimizzando il profitto dell'iterazione successiva e rispettando i vincoli di sviluppo. L'evoluzione viene gestita consentendo modifiche ai parametri dopo ogni iterazione e valutandone l'impatto sul modello, con una nuova soluzione per l'iterazione successiva generata da zero incorporando modifiche e storie aggiuntive. Un approccio più sofisticato è rappresentato dalla pianificazione bi-obiettivo proposta in [**?**], in cui l'iterazione successiva viene pianificata considerando l'impatto di nuovi requisiti o modifiche sul sistema esistente dal punto di vista aziendale e di sviluppo. Viene generato un insieme di piani alternativi, ciascuno dei quali riflette una diversa importanza relativa degli aspetti azien-

dali e di implementazione; il piano ottimale viene quindi selezionato come quello che meglio soddisfa un gruppo di interdipendenze, denominate SD-coupling, tra i requisiti identificati attraverso l'analisi di impatto. Questo approccio permette di bilanciare in modo esplicito le esigenze di business con le complessità tecniche derivanti dall'evoluzione del sistema.

Sebbene questi approcci affrontino specificamente le problematiche relative all'incertezza e alla ripianificazione, nessuno di essi si occupa di evitare che il nuovo piano interferisca con quello precedente, aspetto che costituisce uno dei contributi distintivi di questo lavoro. Per identificare contributi in questa direzione, è necessario esaminare l'area di ricerca sulla pianificazione in condizioni di incertezza, i cui sforzi si sono concentrati su due fronti principali: la pianificazione proattiva e la pianificazione reattiva [**?**].

La pianificazione proattiva, o robusta, si concentra sullo sviluppo di una pianificazione di base che incorpora un certo grado di anticipazione della variabilità durante l'esecuzione del progetto al fine di proteggere la pianificazione stessa da perturbazioni. Ad esempio, in [**?**] viene definita una pianificazione iniziale aggressiva a cui vengono successivamente aggiunti buffer di risorse e tempo che proteggono i percorsi critici. Oltre alle regole empiriche, come la regola del dimensionamento del buffer al 50%, metodi più sofisticati per il dimensionamento dei buffer sono discussi in [**?**].

La schedulazione reattiva si riferisce invece alle modifiche che potrebbero dover essere apportate alla schedulazione durante l'esecuzione del progetto e può basarsi su diverse strategie. Da un lato, l'approccio reattivo può utilizzare tecniche semplici volte a ripristinare rapidamente la coerenza della schedulazione: ad esempio, la regola dello spostamento a destra proposta in [**?**] posticipa tutte le attività interessate dall'interruzione della schedulazione. D'altro canto, un approccio di schedulazione reattiva può comportare una rischedulazione completa delle attività rimanenti. In caso di rischedulazione, la nuova schedulazione può differire notevolmente da quella di base, circostanza non auspicabile poiché annullerebbe gli impegni precedentemente stabiliti generando costi aggiuntivi, tensioni e insoddisfazione sia da parte dei clienti che dei membri del team. Per questo motivo, gli approcci di rischedulazione naïve si basano spesso su euristiche che effettuano

riorganizzazioni locali dei piani. In alternativa, la rischedulazione può adottare una strategia di minima perturbazione che mira alla stabilità ex post, basandosi su algoritmi esatti o subottimali il cui obiettivo è la minimizzazione di una funzione delle differenze tra gli istanti di inizio di ciascuna attività nella schedulazione nuova e in quella originale [**?**], oppure la minimizzazione del numero di attività da svolgere in sprint diversi [**?**].

Chapter 4

Formulazione matematica

In questa sezione definiamo matematicamente il problema di pianificazione agile. Consideriamo l'insieme $U = \{1, \ldots, n\}$ degli indici delle n user story da realizzare durante il progetto. Ogni user story $j \in U$ è caratterizzata dai seguenti attributi principali:

- u_j: l'utilità, che rappresenta il valore apportato dalla user story;
- r_j^{cr}: il rischio critico, che misura l'impatto potenziale della story sulle altre;
- r_j^{unc}: l'incertezza, che quantifica la difficoltà di stima dovuta a possibili imprevisti;
- p_j: la complessità, espressa in story point, che indica lo sforzo necessario per completarla.

Definiamo inoltre $Y_j \subseteq U$ come l'insieme delle user story affini alla story j, ovvero quelle che, se assegnate allo stesso sprint, generano un valore aggiunto. Per ognuna di esse, a_j rappresenta l'incremento di utilità ottenuto per ogni user story affine assegnata allo stesso sprint. Nel caso in cui $Y_j = \emptyset$, poniamo $a_j = 0$.

Per gestire le dipendenze tra user story, distinguiamo due sottoinsiemi U^{OR} e U^{AND}, rispettivamente contenenti le user story con dipendenze di tipo OR e AND.

- Per ogni $j \in U^{OR}$, indichiamo con $D_j^{OR} \subseteq U$ l'insieme delle user story dalle quali j dipende secondo una logica OR: almeno una delle storie in D_j^{OR} deve

essere assegnata allo stesso sprint di j o a uno sprint precedente affinché j possa essere eseguita.

- Per ogni $j \in U^{AND}$, definiamo $D_j^{AND} \subseteq U$ come l'insieme delle user story da cui j dipende in modalità AND: tutte le storie in D_j^{AND} devono essere assegnate allo stesso sprint di j o a sprint precedenti.

Consideriamo infine l'insieme $S = \{1, \ldots, m\}$ degli m sprint pianificati per il progetto, dove ogni sprint $i \in S$ ha una capacità massima p_i^{max}, espressa in story point, che rappresenta il limite di lavoro che il team può completare durante quello sprint.

Le variabili decisionali che modellano il problema sono:

- $x_{ij} \in \{0, 1\}$, variabile binaria che vale 1 se la user story j viene assegnata allo sprint i, 0 altrimenti;

- $y_{ij} \in \mathbb{Z}_{\geq 0}$, variabile intera non negativa che indica il numero di user story affini a j (cioè appartenenti a Y_j) assegnate allo stesso sprint i.

Il modello di programmazione lineare intera mista è:

$$z_P = \max \sum_{k=1}^{m} \sum_{i=1}^{k} \sum_{j=1}^{n} u_j \left(r_j^{cr} x_{ij} + a_j y_{ij} \right) \tag{4.1}$$

$$s.t. \sum_{j=1}^{n} p_j r_j^{un} x_{ij} \leq p_i^{max}, \qquad i \in S \tag{4.2}$$

$$\sum_{i=1}^{m} x_{ij} = 1, \qquad j \in U \tag{4.3}$$

$$\sum_{k=1}^{i} \sum_{z \in D_j^{OR}} x_{kz} \geq x_{ij}, \qquad i \in S, j \in U^{OR} \tag{4.4}$$

$$\sum_{k=1}^{i} \sum_{z \in D_j^{AND}} x_{kz} \geq x_{ij} |D_j^{AND}|, \quad i \in S, j \in U^{AND} \tag{4.5}$$

$$y_{ij} \leq \sum_{k \in Y_j} x_{ik}, \qquad i \in S, j \in U \tag{4.6}$$

$$y_{ij} \leq |Y_j| x_{ij}, \qquad i \in S, j \in U \tag{4.7}$$

$$x_{ij} \in \{0, 1\}, \qquad i \in S, j \in U \tag{4.8}$$

$$y_{ij} \geq 0, \qquad i \in S, j \in U \tag{4.9}$$

Ecco il testo rielaborato in formato LaTeX:

—

La funzione obiettivo (4.1) ha lo scopo di massimizzare l'utilità complessiva del progetto agile. L'utilità base u_j di ciascuna user story j viene incrementata considerando due fattori principali:

1. il **rischio di criticità** r_j^{cr}, che assegna un peso maggiore alle storie critiche e ne incentiva l'esecuzione anticipata negli sprint iniziali, riducendo il rischio di impatti tardivi sul progetto;

2. l'**incremento di affinità** a_j, che aumenta il valore percepito dall'utente quando storie correlate vengono sviluppate congiuntamente nello stesso sprint, migliorando la coerenza funzionale del rilascio.

Per ogni user story j, il numero di storie affini che vengono incluse nello sprint

i è quantificato dalla variabile y_{ij}, calcolata come:

$$y_{ij} = \sum_{k \in Y_j} x_{ik},$$

dove $y_{ij} = 0$ quando $x_{ij} = 0$, ovvero quando la story j non è assegnata allo sprint i. Trattandosi di un problema di massimizzazione, i vincoli (4.6) e (4.7) garantiscono il corretto calcolo delle variabili y_{ij} senza la necessità di imporre vincoli espliciti di integralità su di esse.

I **vincoli di capacità** (4.2) assicurano che la complessità complessiva delle user story assegnate a ciascuno sprint, misurata in story point, non ecceda la capacità massima disponibile $p_i^{\max}$ per quello sprint, rispettando così i limiti operativi del team.

I **vincoli di assegnazione** (4.3) impongono che ogni user story sia assegnata esattamente a uno e un solo sprint, garantendo che nessuna storia venga trascurata o duplicata nella pianificazione.

Le **relazioni di dipendenza** tra user story sono modellate attraverso i vincoli (4.4) e (4.5), che regolano l'ordine di esecuzione delle storie in base ai loro prerequisiti:

- per le dipendenze di tipo **OR**, il vincolo (4.4) permette l'assegnazione della story j allo sprint i solamente se *almeno una* delle user story appartenenti all'insieme D_j^{OR} è stata completata in uno sprint precedente o nello stesso sprint ($i' \leq i$);

- per le dipendenze di tipo **AND**, il vincolo (4.5) richiede invece che *tutte* le user story contenute nell'insieme D_j^{AND} siano state completate entro lo sprint i, assicurando che tutti i prerequisiti necessari siano soddisfatti prima di procedere con l'esecuzione di j.

4.1 Procedure di riduzione

Le procedure di riduzione mirano a rafforzare i vincoli di capacità (4.2) modificando le capacità degli sprint o i pesi $pr_j = p_j r_j^{un}$ delle user story. Approcci simili sono utilizzati per problemi di packing in [**?**, **?**, **?**].

4.1.1 Modifica delle capacità degli sprint

Se non esiste alcuna combinazione di user story che saturi esattamente la capacità p_i^{max} dello sprint $i \in S$, è possibile ridurre tale capacità rimuovendo i "punti storia inutilizzabili" senza alterare il valore ottimale del problema. La nuova capacità può essere trovata risolvendo il problema di Subset Sum:

$$p_i^{max} = \max \left\{ \sum_{k \in U} pr_k \xi_k : \sum_{k \in U} pr_k \xi_k \leq p_i^{max}, \quad \xi_j \in \{0,1\},\, j \in U \right\}$$

dove $pr_j = p_j r_j^{un}$ è il peso effettivo associato alla user story j. Il problema di Subset Sum può essere efficacemente risolto utilizzando una procedura di programmazione dinamica.

4.1.2 Modifica dei pesi delle user story

Un ulteriore miglioramento consiste nell'aumentare il peso pr_j di una user story $j \in U$, mantenendo la fattibilità dello sprint che la contiene. Per ogni sprint $i \in S$, definiamo il valore p''_{ij}:

$$p''_{ij} = \max \left\{ \sum_{h \in U} pr_h \xi_h : \sum_{h \in U} pr_h \xi_h \leq p_i^{max}, \quad \xi_j = 1, \quad \xi_k \in \{0,1\},\, k \in U \setminus \{j\} \right\}$$

ossia la massima somma di pesi che include necessariamente la storia j senza superare la capacità dello sprint. In tal modo, è possibile aggiornare il peso di j come:

$$pr_j := pr_j + (p_i^{max} - p''_{ij}).$$

Per massimizzare il numero di storie a cui aggiornare il peso, si propone di ordinare le user story in ordine decrescente di peso, ovvero:

$$pr_1 \geq pr_2 \geq \cdots \geq pr_n.$$

Queste tecniche di riduzione sfruttano problemi classici come il Subset Sum, risolvibili con algoritmi di programmazione dinamica, contribuendo a semplificare il modello e a ridurre i tempi di calcolo necessari per la soluzione ottimale.

Chapter 5

Euristiche per la pianificazione degli sprint

5.1 Euristiche greedy e di scambio del lavoro precedente

In [**?**] sono state proposte due euristiche greedy per costruire un piano di sprint e una procedura di post ottimizzazione basata su scambi locali. In questa sezione ne riprendiamo brevemente il funzionamento, poiché tali euristiche costituiscono la base anche per il lavoro sviluppato in questa tesi.

5.1.1 GreedyHeuristic: costruzione sprint-per-sprint

L'idea di base della *GreedyHeuristic* è la seguente: il piano viene costruito iterativamente, ottimizzando uno sprint per volta in ordine cronologico (prima lo sprint 1, poi lo sprint 2, e così via). A ogni passo, fissato uno sprint $i \in S$ e l'insieme F_i delle user story già assegnate agli sprint precedenti, si risolve il sottoproblema

(SP_i):

$$(SP_i) \quad z_{SP_i} = \max \sum_{j=1}^{n} (m-i+1) u_j \left(r_j^{cr} x_{ij} + a_j y_{ij} \right) \tag{5.1}$$

$$s.t. \sum_{j=1}^{n} p_j r_j^{un} x_{ij} \leq p_i^{max} \tag{5.2}$$

$$\sum_{k=1}^{i} \sum_{z \in D_j^{OR}} x_{kz} \geq x_{ij} - |D_j \cap F_i|, \qquad j \in U^{OR} \tag{5.3}$$

$$\sum_{k=1}^{i} \sum_{z \in D_j^{AND}} x_{kz} \geq x_{ij} |D_j| - |D_j \cap F_i|, \quad j \in U^{AND} \tag{5.4}$$

$$y_{ij} \leq \sum_{k \in Y_j} x_{ik}, \qquad j \in U \tag{5.5}$$

$$y_{ij} \leq |Y_j| x_{ij}, \qquad j \in U \tag{5.6}$$

$$x_{ij} \in \{0,1\}, \qquad j \in U \setminus F_i \tag{5.7}$$

$$x_{ij} = 0, \qquad j \in F_i \tag{5.8}$$

$$y_{ij} \geq 0, \qquad i \in S, j \in U \tag{5.9}$$

dove F_i è l'insieme delle storie già assegnate agli sprint $1, \ldots, i-1$ (con $F_1 = \emptyset$).

Listing 5.1: GreedyHeuristic

```
1 GreedyHeuristic
2 -------------------------
3 Set $F_1 = \emptyset$, $i = 1$
4
5 WHILE ($F_i \neq U$ and $i \leq m$):
6     Compute the optimal solution $\mathbf{x}^*$ of
         subproblem $SP_i$
7     Set $F_{i+1} = F_i \cup \{ j \in U : x^*_{ij} = 1 \}$
8     Set $i = i + 1$
```

Pseudocodice della GreedyHeuristic. Questa procedura è concettualmente greedy perché, a ogni passo, prende la "miglior scelta locale" per lo sprint corrente

(risolvendo SP_i) e non rivede le decisioni sugli sprint precedenti.

5.1.2 QuickGreedyHeuristic: versione veloce basata su knapsack

l'euristica *GreedyHeuristic*, riassunta in Figura 5.1, è estremamente rapida se eseguita una sola volta, ma il suo costo computazionale può diventare rilevante quando deve essere richiamata ripetutamente. In particolare come mostrato in [**?**], per alcune istanze di grandi dimensioni, una singola esecuzione di *GreedyHeuristic* richiede più di un secondo; di conseguenza, ripeterla per migliaia di iterazioni comporta un tempo totale dell'ordine di migliaia di secondi, imputabile unicamente alla componente greedy.

Per ridurre questo onere, proponiamo una variante accelerata dell'euristica greedy, denominata *QuickGreedyHeuristic*. L'idea di base consiste nel rilassare il sottoproblema SP_i rimuovendo i vincoli (5.3) (5.6). Il sottoproblema risultante, indicato con SP'_i, si riconduce a un problema di knapsack che può essere risolto in modo efficiente tramite programmazione dinamica. Tale rilassamento introduce tuttavia due criticità:

- in assenza dei vincoli di collegamento (5.5) (5.6), le variabili $\{y_{ij}\}$ diventano indipendenti dalle decisioni di assegnazione;
- l'eliminazione dei vincoli di dipendenza (5.3) (5.4) può determinare violazioni delle relazioni di precedenza tra le user story.

Per risolvere il primo problema, il knapsack SP'_i viene risolto ignorando le variabili $\{y_{ij}\}$, che sono poi valutate a posteriori a partire dalla soluzione $\{x_{ij}\}$ di SP'_i. In particolare, per ogni sprint i e user story j, poniamo

$$y_{ij} = \begin{cases} \sum_{k \in Y_j} x_{ik} & \text{se } x_{ij} = 1, \\ 0 & \text{altrimenti.} \end{cases}$$

In questo modo, le variabili y_{ij} riflettono il numero di user story "di supporto" selezionate nello sprint i a fronte della scelta di j.

Per quanto riguarda le dipendenze, la soluzione del knapsack rilassato SP'_i può violare i vincoli (5.3) (5.4). Fissato lo sprint corrente i e una user story j che viola almeno un vincolo di dipendenza, *QuickGreedyHeuristic* adotta una tra quattro strategie di recupero della fattibilità:

1. **Esclusione di** j. Si proibisce l'uso di j nello sprint i fissando $x_{ij} = 0$ e si riottimizza il knapsack SP'_i.

2. **Inserimento mirato di una dipendenza**. Si seleziona una user story $j' \in D_j^{OR}$ (oppure $j' \in D_j^{AND}$, a seconda del vincolo violato), non presente nella soluzione corrente di SP'_i, che massimizza il rapporto

$$\frac{u_{j'} r_{j'}^{cr}}{p_{j'} r_{j'}^{un}},$$

   si fissa $x_{ij'} = 1$ e si riottimizza SP'_i.

3. **Inserimento completo per vincoli AND**. Solo nel caso di dipendenze di tipo AND, si fissano in soluzione tutte le user story $j' \in D_j^{AND}$ non selezionate (ponendo $x_{ij'} = 1$) e si riottimizza SP'_i; per le dipendenze di tipo OR si applica invece la strategia (ii).

4. **Rinforzo progressivo dei profitti**. Per ogni user story j si introduce un coefficiente κ_j che moltiplica il profitto in ogni knapsack SP'_i. Per ogni $j' \in D_j^{OR} \setminus F_i$ (oppure $j' \in D_j^{AND} \setminus F_i$, a seconda del vincolo violato) con $x_{ij'} = 0$ nella soluzione corrente, si aumenta $\kappa_{j'}$ secondo una delle due regole:

   (a) $\kappa_{j'} = \rho\, \kappa_{j'}$,

   (b) $\kappa_{j'} = \kappa_{j'}^2$.

   Successivamente, si riavvia l'euristica greedy dal primo sprint, ponendo $i = 1$ e $F_1 = \emptyset$.

La Figura 5.2 riassume il funzionamento dell'algoritmo *QuickGreedyHeuristic*. Per ciascuna strategia $s = 1, 2, 3, 4$, l'algoritmo costruisce iterativamente una soluzione, aggiornando lo sprint corrente i, l'insieme delle user story già assegnate

agli sprint precedenti (F_i) e il valore complessivo z' della soluzione ottenuta. Alla fine, viene mantenuta la migliore soluzione trovata $(\mathbf{x}^*, z^*)$.

In termini di proprietà, la strategia (i) garantisce sempre la convergenza a una soluzione fattibile, poiché ogni violazione viene gestita tramite esclusione della user story responsabile. Al contrario, le strategie (ii) e (iii), in particolare la terza, possono generare istanze di knapsack infeasibii (ad esempio quando la capacità di sprint è insufficiente per ospitare tutte le dipendenze forzate). In tali casi, se SP'_i risulta infeasibie, l'algoritmo interrompe il trattamento dello sprint i e passa all'iterazione successiva della procedura greedy. La strategia (iv) può richiedere un numero elevato di iterazioni prima di raggiungere la fattibilità; per questo motivo, viene imposto un numero massimo di iterazioni *MaxIter*, oltre il quale il tentativo con la strategia (iv) viene interrotto.

Nella fase sperimentale, si è fissato $MaxIter = 1000$. La strategia (iv) è stata applicata una volta utilizzando la regola (b), inizializzando i coefficienti a $\kappa_j = 1.025$ per ogni $j \in U$, e due volte utilizzando la regola (a), con $\kappa_j = 1$ per ogni $j \in U$ e valori di ρ pari a 2 e 5. La scelta di κ_j e ρ riflette un compromesso tra tempo di raggiungimento della fattibilità e qualità della soluzione: se i profitti crescono troppo rapidamente, si ottiene spesso una soluzione fattibile in poche iterazioni, ma potenzialmente di bassa qualità, poiché le user story "rinforzate" vengono anticipate ai primi sprint senza considerare adeguatamente la loro utilità reale; al contrario, se i profitti aumentano troppo lentamente, il numero di iterazioni necessario per ottenere una soluzione fattibile può diventare eccessivo.

5.1.3 ExchangeHeuristic: miglioramento locale per scambi

Sia *GreedyHeuristic* sia *QuickGreedyHeuristic* possono terminare senza individuare una soluzione fattibile, poiché tutti gli sprint risultano esaminati (cioè $i > m$) ma non tutte le user story sono assegnate agli sprint disponibili (ovvero $F_i \neq U$). Inoltre, anche quando *QuickGreedyHeuristic* produce una soluzione fattibile, tale soluzione può in genere essere ulteriormente migliorata: le strategie di recupero della fattibilità rispetto ai vincoli di dipendenza, infatti, possono condurre a configurazioni non localmente ottimali.

In [**?**] viene proposta *ExchangeHeuristic* descritta in Figura 5.3. Data una

soluzione $\mathbf{x}'$ da migliorare, la procedura tenta sistematicamente di scambiare user story tra coppie di sprint, ripetendo il processo fino a quando avviene almeno uno scambio migliorativo. In particolare, *ExchangeHeuristic* considera tre tipi di mosse:

1-0: spostamento di una user story $j \in U$ eseguita nello sprint i' verso uno sprint i;

1-1: scambio di una user story $j \in U$ eseguita nello sprint i con una user story $j' \in U$ eseguita nello sprint i';

2-1: scambio di due user story $j, j' \in U$ eseguite nello sprint i con una user story $j'' \in U$ eseguita nello sprint i'.

Uno scambio viene effettivamente applicato solo se è contemporaneamente *fattibile* e *profittevole*. La fattibilità richiede che, dopo lo scambio, i vincoli di capacità degli sprint coinvolti rimangano soddisfatti e che non si generino violazioni delle dipendenze tra user story; la profittabilità richiede invece che il valore complessivo della funzione obiettivo aumenti.

L'algoritmo *ExchangeHeuristic* può essere esteso includendo mosse di scambio più complesse (ad esempio scambi k-ℓ con $k, \ell > 2$). Tuttavia, l'incremento di complessità computazionale associato a tali estensioni tende a non essere compensato da miglioramenti significativi della qualità della soluzione, motivo per cui in questo lavoro ci si limita alle mosse 1-0, 1-1 e 2-1 descritte sopra.

5.2 A Lagrangian Heuristic

La letteratura propone numerose euristiche basate su metodi di decomposizione e rilassamento lagrangiano; introduzioni esaustive a questi temi sono disponibili, tra gli altri, in [**?**, **?**, **?**]. In [**?**] si descrive una procedura euristica che sfrutta il rilassamento lagrangiano del modello, combinato con un algoritmo del sottogradiente e con le euristiche greedy introdotte in precedenza.

Il rilassamento lagrangiano si ottiene dualizzando i vincoli (4.3)–(4.7) mediante i vettori di penalità $\{\lambda_j\}$, $\{\lambda_{ij}^{OR}\}$, $\{\lambda_{ij}^{AND}\}$, $\{\lambda_{ij}^{Y1}\}$ e $\{\lambda_{ij}^{Y2}\}$. Le penalità λ_j, $j \in U$, sono non vincolate, mentre le restanti sono non positive, così da interpretare le

violazioni dei vincoli corrispondenti come costi aggiuntivi nel problema rilassato. Il problema lagrangiano risultante è:

$$(LR) \qquad z_{LR}(\boldsymbol{\lambda}) = \max \sum_{k=1}^{m} \sum_{i=1}^{k} \sum_{j=1}^{n} \left(u'_{ij}(\boldsymbol{\lambda}) x_{ij} + u''_{ij}(\boldsymbol{\lambda}) y_{ij}\right) + \sum_{j=1}^{n} \lambda_j \tag{5.10}$$

$$\text{s.t.} \ \sum_{j=1}^{n} p_j r_j^{un} x_{ij} \leq p_i^{\max}, \qquad i \in S \tag{5.11}$$

$$x_{ij} \in \{0,1\}, \qquad i \in S, j \in U \tag{5.12}$$

$$0 \leq y_{ij} \leq |Y_j|, \qquad i \in S, j \in U \tag{5.13}$$

dove le *utilità penalizzate* $u'_{ij}(\boldsymbol{\lambda})$ e $u''_{ij}(\boldsymbol{\lambda})$ sono definite da

$$\begin{aligned} u'_{ij}(\boldsymbol{\lambda}) &= u_j r_j^{cr} - \lambda_j + \left(\lambda_{ij}^{OR} - \sum_{k=i}^{m} \sum_{j' \in \bar{D}_j^{OR}} \lambda_{kj'}^{OR} \right) + \\ &\quad + \left(|D_j^{AND}| \lambda_{ij}^{AND} - \sum_{k=i}^{m} \sum_{j' \in \bar{D}_j^{AND}} \lambda_{kj'}^{AND} \right) - \lambda_{ij}^{Y1} - |Y_j| \lambda_{ij}^{Y2} \\ u''_{ij}(\boldsymbol{\lambda}) &= u_j a_j + \lambda_{ij}^{Y1} + \lambda_{ij}^{Y2}, \end{aligned} \tag{5.14}$$

con $\bar{D}_j^{OR} = \{j' \in U : j \in D_{j'}^{OR}\}$ e $\bar{D}_j^{AND} = \{j' \in U : j \in D_{j'}^{AND}\}$.

Grazie alla struttura dei vincoli di capacità, il problema LR si decompone in $2m$ sottoproblemi indipendenti, due per ogni sprint $i \in S$:

$$(LR_i^1) \qquad z_{LR_i^1}(\boldsymbol{\lambda}) = \max \sum_{j=1}^{n} u'_{ij}(\boldsymbol{\lambda}) x_{ij} \tag{5.15}$$

$$\text{s.t.} \ \sum_{j=1}^{n} p_j r_j^{un} x_{ij} \leq p_i^{\max} \tag{5.16}$$

$$x_{ij} \in \{0,1\}, \qquad j \in U \tag{5.17}$$

e

$$(LR_i^2) \qquad z_{LR_i^2}(\boldsymbol{\lambda}) = \max \sum_{j=1}^{n} u''_{ij}(\boldsymbol{\lambda}) y_{ij} \tag{5.18}$$

$$\text{s.t. } 0 \leq y_{ij} \leq |Y_j|,\ j \in U \tag{5.19}$$

Il sottoproblema LR_i^1 è un problema di knapsack, mentre LR_i^2 è risolvibile per ispezione, assegnando $y_{ij} = |Y_j|$ se $u''_{ij}(\boldsymbol{\lambda}) > 0$ e $y_{ij} = 0$ altrimenti. Il valore ottimo del rilassamento lagrangiano è quindi

$$z_{LR}(\boldsymbol{\lambda}) = \sum_{i=1}^{m} (m - i + 1)\big(z_{LR_i^1}(\boldsymbol{\lambda}) + z_{LR_i^2}(\boldsymbol{\lambda})\big) + \sum_{j=1}^{n} \lambda_j, \tag{5.20}$$

che fornisce un upper bound valido per il problema originale P [web:52][web:55]. La ricerca del vettore di penalità $\boldsymbol{\lambda}^*$ che minimizza $z_{LR}(\boldsymbol{\lambda})$ porta al *dual lagrangiano* $z_{LR}(\boldsymbol{\lambda}^*) = \min_{\boldsymbol{\lambda}}\{z_{LR}(\boldsymbol{\lambda})\}$, affrontato qui tramite un algoritmo del sottogradiente [web:55][web:59].

Sia $(\boldsymbol{x}, \boldsymbol{y})$ la soluzione, di valore $z_{LR}(\boldsymbol{\lambda})$, ottenuta risolvendo LR a una generica iterazione del sottogradiente. I moltiplicatori lagrangiani vengono aggiornati secondo

$$\begin{aligned}
\lambda_j &= \lambda_j + \alpha g_j, && j \in U \\
\lambda_{ij}^{OR} &= \max\{0, \lambda_{ij}^{OR} + \alpha g_{ij}^{OR}\}, && i \in S, j \in U^{OR} \\
\lambda_{ij}^{AND} &= \max\{0, \lambda_{ij}^{AND} + \alpha g_{ij}^{AND}\}, && i \in S, j \in U^{AND} \\
\lambda_{ij}^{Y1} &= \max\{0, \lambda_{ij}^{Y1} + \alpha g_{ij}^{Y1}\}, && i \in S, j \in U \\
\lambda_{ij}^{Y2} &= \max\{0, \lambda_{ij}^{Y2} + \alpha g_{ij}^{Y2}\}, && i \in S, j \in U
\end{aligned} \tag{5.21}$$

dove α è la lunghezza del passo lungo la direzione di ricerca data dal sottogradiente

$\boldsymbol{g}$, le cui componenti sono

$$\begin{aligned}
g_j &= \sum_{i=1}^{m} x_{ij} - 1, && j \in U \\
g_{ij}^{OR} &= \sum_{k=1}^{i} \sum_{z \in D_j^{OR}} x_{kz} - x_{ij}, && i \in S, j \in U^{OR} \\
g_{ij}^{AND} &= \sum_{k=1}^{i} \sum_{z \in D_j^{AND}} x_{kz} - x_{ij}|D_j^{AND}|, && i \in S, j \in U^{AND} \\
g_{ij}^{Y1} &= \sum_{k \in Y_j} x_{ik} - y_{ij}, && i \in S, j \in U \\
g_{ij}^{Y2} &= |Y_j| x_{ij} - y_{ij}, && i \in S, j \in U
\end{aligned} \tag{5.22}$$

La regola per il passo adottata negli esperimenti è

$$\alpha = \beta \frac{0.1\, z_{LR}(\boldsymbol{\lambda})}{\|\boldsymbol{g}\|_2^2},$$

dove β è inizializzato a un valore dipendente dal problema (nel nostro caso, $\beta = 3$) e viene ridotto moltiplicandolo per 0.85 se, dopo un numero prefissato di iterazioni (10), il valore $z_{LR}(\boldsymbol{\lambda})$ non migliora [web:55][web:59]. Il numero massimo di iterazioni è fissato a 5000 e il metodo viene interrotto anticipatamente se, in un intervallo di 50 iterazioni, il valore di $z_{LR}(\boldsymbol{\lambda})$ non si riduce di almeno lo 0.01%.

La procedura complessiva, denominata *LagrangianHeuristic*, è riassunta in Figura 5.4. A ogni iterazione del sottogradiente si risolve LR per un dato vettore di penalità $\boldsymbol{\lambda}$, ottenendo il valore $z_{LR}(\boldsymbol{\lambda})$ e aggiornando il miglior upper bound z_{LR}^*. Successivamente, si calcola una soluzione euristica $\boldsymbol{x'}$ tramite *QuickGreedyHeuristic*, utilizzando le utilità penalizzate definite in (5.14), e la si migliora mediante *ExchangeHeuristic*; il valore z' della soluzione così ottenuta sostituisce il miglior valore corrente z^* se $z' > z^*$. Inoltre, se la condizione $\gamma z^* \leq z'$ è soddisfatta, si genera una seconda soluzione euristica $\boldsymbol{x''}$ con la più costosa *GreedyHeuristic*, sempre basata sulle utilità penalizzate, e il corrispondente valore z'' viene confrontato con z^* per un eventuale aggiornamento.

Il parametro γ controlla la frequenza di invocazione di *GreedyHeuristic*: po-

nendo $\gamma = 1$ si esegue questa procedura solo quando $\boldsymbol{x}'$ rappresenta la migliore soluzione trovata finora, mentre per $\gamma < 1$ *GreedyHeuristic* viene attivata anche quando z' è "sufficientemente vicino" a z^*, cioè entro una distanza percentuale $100(1-\gamma)$. Al termine di *LagrangianHeuristic* si dispone di una soluzione fattibile di valore z^* e di un upper bound z^*_{LR} derivante dal rilassamento lagrangiano, che consente di stimare la distanza massima della soluzione euristica dall'ottimo del problema originale.

Listing 5.2: QuickGreedyHeuristic

```
Set z_star = -infinity

FOR each Strategy s = 1, 2, 3, 4:
    Set z_temp = 0
    Set F_1 = empty
    Set i = 1
    Set Iter = 0

    WHILE (F_i != U AND i <= m):

        REPEAT:
            Compute optimal solution x_temp of knapsack
               SP_i_prime

            IF (x_temp violates some dependency):
                Apply strategy s
                IF (s == 4):
                    Set z_temp = 0
                    Set F_1 = empty
                    Set i = 1
                    Set Iter = Iter + 1
        UNTIL (x_temp feasible OR SP_i_prime infeasible OR
           Iter > MaxIter)

        IF (SP_i_prime infeasible OR Iter > MaxIter):
            Set z_temp = -infinity
            Set i = m + 1
        ELSE:
            z_temp = z_temp + value(SP_i)
            F_{i+1} = F_i union { j in U with x_temp[i,j] =
               1 }
            i = i + 1

    IF (z_star < z_temp):
        z_star = z_temp
        x_star = x_temp

END FOR
```

Listing 5.3: ExchangeHeuristic

```
Algorithm ExchangeHeuristic(x'):
Repeat
    // Apply 1-0 exchanges
    For sprint i = 1..m-1:
        For sprint i' = i+1..m:
            For each story j with x'[i',j] = 1:
                If moving j from i' to i is feasible and
                   profitable:
                    x'[i',j] = 0
                    x'[i,j]   = 1

    // Apply 1-1 exchanges
    For sprint i = 1..m-1:
        For sprint i' = i+1..m:
            For each pair j, j' with x'[i,j]=1 and x'[i',j
               ']=1:
                If exchanging j and j' is feasible and
                   profitable:
                    x'[i,j]   = 0
                    x'[i',j'] = 0
                    x'[i,j']  = 1
                    x'[i',j]  = 1

    // Apply 2-1 exchanges
    For sprint i = 1..m:
        For sprint i' = 1..m:
            For each triple j, j', j'' with x'[i,j]=x'[i,j
               ']=x'[i',j'']=1:
                If exchanging j with {j',j''} is feasible
                   and profitable:
                    x'[i,j]   = 0
                    x'[i,j']  = 0
                    x'[i',j'']= 0
                    x'[i',j]   = 1
                    x'[i',j']  = 1
                    x'[i,j'']  = 1

Until no exchanges occur.
```

Listing 5.4: Algorithm LagrangianHeuristic

```
LagrangianHeuristic:

Set lambda = 0
Set z_best = -INF
Set z_LR_best = +INF

REPEAT

    Compute z_LR(lambda) by solving the Lagrangian problem
       LR

    IF z_LR(lambda) < z_LR_best THEN
        z_LR_best = z_LR(lambda)
    ENDIF

    Compute a heuristic solution x_prime with
       QuickGreedyHeuristic
        using penalized utilities
    Improve x_prime with ExchangeHeuristic
    Let z_prime = value of x_prime

    IF gamma * z_best <= z_prime THEN
        Compute a heuristic solution x_second with
           GreedyHeuristic
            using penalized utilities
        Let z_second = value of x_second

        IF z_best < z_second THEN
            z_best = z_second
            x_best = x_second
        ENDIF
    ENDIF

    IF z_best < z_prime THEN
        z_best = z_prime
        x_best = x_prime
    ENDIF

    Update penalties lambda

UNTIL subgradient stopping conditions are satisfied
```

Chapter 6

Risultati ottenuti

6.1 dati utilizzati

Gli algoritmi considerati in questo lavoro sono stati implementati in C++ all'interno di Microsoft Visual Studio 2024.

Per la valutazione sperimentale sono stati utilizzati sia progetti reali sia progetti sintetici generati artificialmente.

I progetti reali provengono da contesti applicativi differenti e sono stati realizzati da aziende italiane che adottano metodologie Agile da diversi anni. Di seguito se ne riassumono le principali caratteristiche.

- PayTV Progetto di sviluppo di un data mart per una grande azienda del settore pay TV. Caratteristiche principali:
  - durata complessiva: 8 mesi;
  - numero di user stories: 44;
  - numero di sprint: 10;
  - durata media degli sprint: 17 giorni;
  - numero di dipendenze: 35 (prevalentemente di tipo AND);
  - numero di affinità: 1;
  - velocità di sviluppo: 2,43 story point/giorno, stimata empiricamente su dati storici.

- Web Progetto di sviluppo di un sito web complesso basato su un Content Management System. Caratteristiche principali:
  - numero di user stories: 104 (maggiore di PayTV);
  - numero di sprint: 4;
  - durata degli sprint: 10 giorni ciascuno;
  - durata complessiva: 40 giorni;
  - velocità di sviluppo: 6 story point/giorno;
  - dipendenze: 6 dipendenze in catena;
  - nessun vincolo di affinità.

Dal punto di vista dell'efficacia, tutti i piani ottimali generati dagli algoritmi sono stati giudicati fattibili e realistici dai project manager; nella maggior parte dei casi, le differenze rispetto ai piani originariamente proposti dai team sono state valutate come migliorative in termini di composizione degli sprint (si veda [**?**] per un'analisi dettagliata).

Per mettere maggiormente sotto stress gli algoritmi, è stato inoltre generato un insieme di progetti sintetici. Il generatore di istanze opera come segue:

1. crea le user stories assegnando in modo casuale utilità, rischio e complessità;
2. aggiunge gruppi di dipendenze, organizzati sia come catene (una story dipende da una sola story) sia come DAG (una story dipende da più stories);
3. definisce insiemi di storie affini;
4. fissa la capacità di sprint a 45 story point e la velocità di sviluppo a 3 story point/giorno (ogni sprint dura quindi 15 giorni).

6.1.1 Raggruppamento dei progetti

La Tabella 6.1 riassume le caratteristiche principali di ciascun progetto, riportando:

- il numero n di user stories;
- il numero massimo m di sprint;

- il numero n_{aff} di stories coinvolte in almeno un vincolo di affinità;
- la cardinalità degli insiemi U^{OR} e U^{AND};
- la lunghezza massima l_{max} dei gruppi di dipendenze;
- il numero massimo d_{max} di dipendenze che coinvolgono una singola story.

I progetti sono stati raggruppati in cinque cluster:

Gruppo A Contiene i progetti reali (PayTV e Web).

- Parametri derivati da casi industriali.
- Mix realistico di dimensione, dipendenze e (eventuale) presenza di affinità.

Gruppi B e C Raccolgono progetti sintetici che combinano in modo vario i parametri sopra descritti.

- Dimensioni crescenti in termini di numero di stories e sprint.
- Diversa distribuzione di dipendenze OR/AND e lunghezza delle catene/DAG.
- Presenza o assenza di affinità secondo configurazioni eterogenee.

Gruppo D Progetti sintetici in cui l'utilità è fortemente correlata alla complessità.

- Per ogni story, la complessità è sempre pari al doppio dell'utilità.
- Scenario utile per analizzare casi in cui "alto valore" implica automaticamente "alta complessità".

Gruppo E Progetti caratterizzati da user stories ad elevata complessità.

- Ogni sprint può contenere al più 5 stories.
- Configurazione concepita per studiare il comportamento degli algoritmi in presenza di capacità molto stringenti.

Table 6.1: Problem instances

Group	Proj. Name	n	m	n_{aff}	$\|U^{OR}\|$	$\|U^{AND}\|$	l_{max}	d_{max}
				A – Real				
PayTV		44	12	2	8	27	6	5
Web		104	6	5	0	4	4	1
				B – Basic (Set 1)				
25Chain-1		25	9	0	0	12	4	1
25Graph-1		25	8	0	10	1	2	2
25Affinity-1		25	9	6	5	5	2	2
50Chain-1		50	12	0	0	20	4	1
50Graph-1		50	11	0	8	10	2	2
50Affinity-1		50	12	6	8	9	2	2
75Chain-1		75	17	0	0	35	5	1
75Graph-1		75	19	0	13	20	2	2
75Affinity-1		75	17	6	17	13	2	3
100Chain-1		100	20	0	0	40	5	1
100Graph-1		100	23	0	20	14	2	3
100Affinity-1		100	22	6	16	14	3	4
				C – Basic (Set 2)				
25Chain-2		25	8	0	0	12	2	1
25Graph-2		25	8	0	3	8	3	2
25Affinity-2		25	9	6	7	4	3	2
50Chain-2		50	13	0	0	10	5	1
50Graph-2		50	13	0	13	8	4	5
50Affinity-2		50	13	6	13	9	3	3
75Chain-2		75	17	0	0	36	3	1
75Graph-2		75	18	0	14	16	2	2
75Affinity-2		75	18	6	17	13	2	2
100Chain-2		100	22	0	0	30	5	1
100Graph-2		100	22	0	12	15	5	7
100Affinity-2		100	22	6	11	14	3	2
				D – Correlated				
25Chain-3		25	15	0	0	12	4	1
25Graph-3		25	15	0	5	7	5	4
25Affinity-3		25	13	6	5	4	2	2
50Chain-3		50	22	0	0	20	4	1
50Graph-3		50	22	0	7	9	2	2
50Affinity-3		50	21	6	9	6	2	2
75Chain-3		75	31	0	0	36	6	1
75Graph-3		75	29	0	12	17	2	2
75Affinity-3		75	33	6	20	7	2	2
100Chain-3		100	38	0	0	40	8	1
100Graph-3		100	40	0	16	14	3	3
100Affinity-3		100	43	6	16	13	2	2
				B – Few				
25Chain-4		25	12	0	5	7	4	1
25Graph-4		25	13	0	5	6	5	4
25Affinity-4		25	14	6	3	7	2	2
50Chain-4		50	21	0	9	11	4	1
50Graph-4		50	21	0	2	13	2	2
50Affinity-4		50	21	6	6	8	2	2
75Chain-4		75	27	0	15	21	6	1
75Graph-4		75	29	0	12	12	2	4
75Affinity-4		75	29	6	10	12	2	2
100Chain-4		100	40	0	24	21	9	1
100Graph-4		100	40	0	8	17	2	2
100Affinity-4		100	39	6	18	10	2	2

Chapter 7

Conclusioni

Bibliography

[AA03] Oğuzhan Alagöz and Meral Azizoğlu. Rescheduling of identical parallel machines under machine eligibility constraints. *European Journal of Operational Research*, 149(3):523–532, 2003.

[ABV10] P. Avella, M. Boccia, and I. Vasilyev. A computational study of exact kanpsack separation for the generalized assignment problem. *Computational Optimization and Applications*, 45:543–555, 2010.

[AWSR03a] P. Abrahamsson, J. Warsta, M.T. Siponen, and J. Ronkainen. New directions on agile methods: a comparative analysis. In *25th International Conference on Software Engineering, 2003. Proceedings.*, pages 244–254, 2003.

[AWSR03b] Pekka Abrahamsson, Juhani Warsta, Mikko T. Siponen, and Jussi Ronkainen. New directions on agile methods: A comparative analysis. In *Proc. ICSE*, pages 244–254, 2003.

[AZDCG18] Wisam Haitham Abbood Al-Zubaidi, Hoa Khanh Dam, Morakot Choetkiertikul, and Aditya Ghose. Multi-objective iteration planning in agile development. In *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*, pages 484–493, 2018.

[B+01] K. Beck et al. Manifesto for agile software development. http://agilemanifesto.org, 2001.

[BBvB+01] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith,

Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. Manifesto for agile software development, 2001.

[BDM+99] Peter Brucker, Andreas Drexl, Rolf Möhring, Klaus Neumann, and Erwin Pesch. Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, 112(1):3–41, 1999.

[BGRT14] Marco A. Boschetti, Matteo Golfarelli, Stefano Rizzi, and Elisa Turricchia. A lagrangian heuristic for sprint planning in agile software development. *Computers Operations Research*, 43:116–128, 2014.

[BHM02] M.A. Boschetti, E. Hadjinconstantinou, and A. Mingozzi. New upper bounds for the finite two-dimensional orthogonal non-guillotine cutting stock problem. *IMA Journal of Management Mathematics*, 13:95–119, 2002.

[BM03] M.A. Boschetti and A. Mingozzi. The two-dimensional finite bin packing problem. Part I: New lower bounds for the oriented case. *4OR*, 1:27–42, 2003.

[BM10] Marco Antonio Boschetti and Lorenza Montaletti. An exact algorithm for the two-dimensional strip-packing problem. *Operations Research*, 58(6):1774–1791, November 2010.

[Coh04a] Mike Cohn. *User Stories Applied: For Agile Software Development.* Addison-Wesley Professional, 2004.

[Coh04b] Mike Cohn. *User stories applied: For agile software development.* Addison-Wesley Professional, 2004.

[Coh05] Mike Cohn. *Agile estimating and planning.* Pearson Education, 2005.

[DB98] Ronald De Boer. *Resource-constrained multi-project management.* PhD thesis, PhD thesis, University of Twente, The Netherlands, 1998.

[DCH03] Mark Denne and Jane Cleland-Huang. *Software by numbers: Low-risk, high-return development*. Prentice Hall Professional, 2003.

[DD08a] Tore Dybå and Torgeir Dingsøyr. Empirical studies of agile software development: A systematic review. *Information & Software Technology*, 50(9-10):833–859, 2008.

[DD08b] Tore Dybå and Torgeir Dingsøyr. Empirical studies of agile software development: A systematic review. *Information and Software Technology*, 50(9):833–859, 2008.

[DH02] Erik L Demeulemeester and Willy S Herroelen. *Project scheduling: a research handbook*. Springer, 2002.

[GR04] D Greer and G Ruhe. Software release planning: an evolutionary and iterative approach. *Information and Software Technology*, 46(4):243–253, 2004.

[GRT12] Matteo Golfarelli, Stefano Rizzi, and Elisa Turricchia. Sprint planning optimization in agile data warehouse design. pages 30–41, 09 2012.

[GRT13] Matteo Golfarelli, Stefano Rizzi, and Elisa Turricchia. Multi-sprint planning and smooth replanning: An optimization model. *Journal of Systems and Software*, 86(9):2357–2370, 2013.

[GS05] Noud Gademann and Marco Schutten. Linear-programming-based heuristics for project capacity planning. *IIE Transactions (Institute of Industrial Engineers)*, 37(2):153 – 165, 2005. Cited by: 31.

[HDDR99] Willy Herroelen, Erik Demeulemeester, and Bert De Reyck. A classification scheme for project scheduling. In *Project scheduling: recent models, algorithms and applications*, pages 1–26. Springer, 1999.

[HLD02] Willy Herroelen, Roel Leus, and Erik Demeulemeester. Critical chain project scheduling: Do not oversimplify. *Project Management Journal*, 33(4):48–60, 2002.

[KGW13] Mahvish Khurum, Tony Gorschek, and Magnus Wilson. The software value map—an exhaustive collection of value aspects for the development of software intensive products. *Journal of software: Evolution and Process*, 25(7):711–741, 2013.

[KP01] R Kolisch and R Padman. An integrated survey of deterministic project scheduling. *Omega*, 29(3):249–272, 2001.

[LvdABD10] Chen Li, Marjan van den Akker, Sjaak Brinkkemper, and Guido Diepen. An integrated approach for requirement selection and scheduling in software release planning. *Requirements Engineering*, 15(4):375 – 396, 2010. Cited by: 48; All Open Access, Hybrid Gold Open Access.

[New98] Robert C Newbold. *Project management in the fast lane: applying the theory of constraints*. CRC Press, 1998.

[PP04] Yongtae Park and Gwangman Park. A new method for technology valuation in monetary value: procedure and application. *Technovation*, 24(5):387–394, 2004.

[PSW94] Adri Platje, Harald Seidel, and Sipke Wadman. Project and portfolio planning cycle: Project-based management for the multiproject challenge. *International Journal of Project Management*, 12(2):100–106, 1994.

[RFB09] Mikko Rönkkö, Christian Frühwirth, and Stefan Biffl. Integrating value and utility concepts into a value decomposition model for value-based software engineering. In *International Conference on Product-Focused Software Process Improvement*, pages 362–374. Springer, 2009.

[Sch95] K. Schwaber. SCRUM development process. In *Proc. OOPSLA*, 1995.

[Sch97] Ken Schwaber. Scrum development process. In Jeff Sutherland, Cory Casanave, Joaquin Miller, Philip Patel, and Glenn Hollowell,

editors, *Business Object Design and Implementation*, pages 117–134, London, 1997. Springer London.

[SOS93] Norman Sadeh, Shinichi Otsuka, and R Schedlback. Predictive and reactive scheduling with the microboss production scheduling and control system. In *Proceedings of the IJCAI-93 workshop on knowledge-based production planning, scheduling and control*, pages 293–306, 1993.

[SR07] Moshood Omolade Saliu and Guenther Ruhe. Bi-objective release planning for evolving software systems. In *Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ESEC-FSE '07, page 105–114, New York, NY, USA, 2007. Association for Computing Machinery.

[SW00] Hani El Sakkout and Mark Wallace. Probe backtrack search for minimal perturbation in dynamic scheduling. *Constraints*, 5(4):359–388, 2000.

[vTdP11] Gert van Valkenhoef, Tommi Tervonen, Bert de Brock, and Douwe Postmus. Quantitative release planning in extreme programming. *Information and Software Technology*, 53(11):1227–1235, 2011. AMOST 2010.

[11] Ákos Szőke. Conceptual scheduling model and optimized release scheduling for agile environments. *Information and Software Technology*, 53(6):574–591, 2011. Special Section: Best papers from the APSEC.

Acknowledgements

Optional. Max 1 page.