Ottimizzazione nella Gestione dei Progetti

Alessandro Agnetis[1], Carlo Mannino[2], Marco Pranzo[1]

May 16, 2006

[1]Dipartimento di Ingegneria dell'Informazione -Università di Siena
[2]Dipartimento di Informatica e Sistemistica -Università di Roma "La Sapienza"

Premessa

Queste dispense nascono dalla sincronizzazione di due esperienze didattiche: specificamente, le lezioni del corso *Ottimizzazione nella gestione dei progetti* per gli studenti del terzo anno della facoltà di ingegneria dell'Università di Roma "La Sapienza" e quelle del corso di *Gestione dei Progetti* per gli studenti del terzo anno di Ingegneria Gestionale di Siena, A.A. 2004–2005. Questi due corsi, della durata di 50 ore circa, si propongono di fornire un'introduzione generale alle problematiche connesse con la gestione dei progetti, dedicando poi un particolare approfondimento agli aspetti quantitativi e metodologici.

La dispensa si compone di due parti. Nella prima parte (§1), introduttiva, viene delineato l'ambito della gestione per progetti, illustrandone gli aspetti concettuali e caratteristici. Oltre a una definizione il più possibile generale di "progetto", si descriveranno gli elementi costitutivi e le implicazioni organizzative di questo concetto. Questo consentirà di analizzare la natura delle diverse competenze del project manager, e di precisare meglio il ruolo che nell'ambito del project management rivestono gli aspetti metodologici. Daremo poi uno sguardo ai diversi processi che hanno luogo durante lo svolgimento di un progetto.

La seconda parte (§2, 3 e 4) tratterà invece di aspetti metodologici (nel senso che questo termine ha per studi di tipo ingegneristico). Vedremo con un certo dettaglio alcuni dei modelli matematici che costituiscono gli strumenti fondamentali per la pianificazione del progetto e per il controllo della sua evoluzione. Tali modelli riguarderanno una varietà di scenari, in base al tipo di relazioni di precedenza che intercorrono tra le attività costitutive il progetto, alle risorse eventualmente necessarie alla loro esecuzione, alla struttura dei costi.

Capitolo 1

Gestione dei progetti

Il project management, che negli ultimi anni sta sempre più affermandosi come disciplina autonoma, racchiude in sé una molteplicità di concetti, strumenti, metodi, nozioni ed esperienze i cui confini sono abbastanza sfumati. In effetti, è oggi molto sentita, da parte di molti operatori del settore, la necessità di pervenire a una sorta di "standardizzazione" terminologica, per poter discriminare con accettabile precisione cosa rientri o non rientri nella sfera di interesse del project management[1]. Quindi, è senz'altro vero che tra le metodologie del project management ricadono i concetti e i modelli matematici necessari per la gestione temporale delle attività di un progetto (che costituiscono la base storica della disciplina), ma vi appartengono anche tutta una serie di concetti e metodi, di carattere più qualitativo, che investono altri aspetti, quali l'organizzazione aziendale, la gestione dei rischi, la comunicazione interpersonale, oltre che, in ultima analisi, aspetti specificamente psicologici/culturali. In altre parole, mentre le competenze richieste per un profilo di tipo tecnico (si pensi ad esempio a un progettista di siti web) sono abbastanza facilmente caratterizzabili in termini di conoscenze e abilità in uno specifico campo tecnologico, quelle richieste a un project manager sono racchiuse in un bagaglio di informazioni ed esperienze ampio ed eterogeneo, e sfuggono a una definizione che sia allo stesso tempo sintetica ed esaustiva. Questo del resto corrisponde a una tendenza recente ma ormai abbastanza generalizzata nell'ambito del management, ossia quella di valorizzare sempre di più le conoscenze di tipo interdisciplinare, trasversale, oltre che specifici skill di tipo comportamentale/interpersonale. Esempi di questi tipi di competenze sono la capacità di comprendere le implicazioni economico/organizzative di una scelta di progetto, la capacità di impostare in modo razionale un determinato problema decisionale con informazione incompleta, di dialogare con soggetti aventi una diversa cultura aziendale, la capacità di motivare le altre persone e di saper collaborare con esse etc. Nel momento in cui si vuole considerare il project manager come una figura professionale a sé (al pari del progettista elettronico, del sistemista informatico, dell'account manager, del responsabile del marketing ...), il suo bagaglio culturale va identificato con un insieme di concetti e nozioni, talora molto eterogeneo, e che si sono in buona misura sviluppati autonomamente

[1] Il PMI (Project Management Institute) è un'associazione che riunisce oggi circa 200.000 professionisti del project management (prevalentemente negli USA), e pubblica periodicamente una sorta di manuale di riferimento relativo alla terminologia, ai metodi e alle pratiche nell'universo del project management, chiamato, forse un po' immodestamente, *Project Management Body Of Knowledge* (PMBOK).

e indipendentemente, ma che assumono una rilevanza nuova e particolare, rivisitati nel quadro unificatore della gestione dei progetti.

1.1 Caratteristiche di un progetto

Il termine *progetto* può evocare molte cose diverse; vale perciò la pena spendere alcune parole per precisare cosa sia da intendersi con questo termine. Una definizione (abbastanza condivisa) di progetto fornita dal PMI è la seguente: "Un progetto è un'iniziativa temporanea intrapresa per creare un prodotto o un servizio con caratteristiche di unicità". Dunque, un team di progetto si costituisce per realizzare una specifica missione, in una certa misura mai realizzata prima, e questo mandato ha una (più o meno esplicita) scadenza temporale. Questi due aspetti (temporaneità e unicità) sono effettivamente caratteristici di un progetto, rispetto ad altre attività aziendali che potremmo definire maggiormente "routinarie".

Per quanto riguarda la *temporaneità*, va fatto notare che la scadenza entro cui il progetto deve essere completato può essere una specifica assolutamente inviolabile (si pensi ai progetti relativi alla conversione dei sistemi contabili delle banche col passaggio all'Euro, o ai progetti nati nelle società di software per il millennium bug), oppure essere molto più "soft", fino a diventare quasi solo una data indicativa. In questi ultimi casi è essenziale che i partecipanti al progetto abbiano una percezione comune della effettiva urgenza o necessità di rispettare tale data. Comunque sia, è chiaro a tutti che il progetto ha un obiettivo ben preciso, e che una volta realizzato, termina.

L'*unicità* è una questione meno immediata. In effetti, quasi tutte le attività hanno in sé una quota di novità; non per tutte però ha senso la classificazione in termini di progetto. La costruzione del tunnel sotto la Manica è senz'altro un buon esempio di progetto (ancorché eccezionalmente grande, rispetto alla media), in quanto ha portato alla realizzazione di qualcosa di unico e nuovo. Invece, realizzare la versione per Unix di un programma per la contabilità che già gira sotto Windows, pur portando in senso stretto a un oggetto nuovo, può rientrare tra le attività di routine per le quali esistono già procedure standardizzate e in cui gli elementi di incertezza sono molto limitati – in questo senso, non si può parlare di progetto.

Alcuni (Damiani et al 2004) fanno notare che un aspetto altamente caratteristico del lavorare per progetti è la cosiddetta *elaborazione progressiva*, ossia il fatto che, pur essendo il progetto unico e a termine, sia la missione (*scope*) che la tempistica sono in realtà oggetto di un continuo processo di revisione. In parole (forse troppo) povere, potremmo dire che spesso nei progetti "si sa quando si parte, ma non si sa quando, e dove, si arriva". Infatti, l'attività di pianificazione (a cui sono dedicate le metodologie che vedremo in queste dispense) non andrà mai intesa come un processo che si esaurisce prima dell'esecuzione del progetto, ma al contrario come un processo che, al pari di altri (vedi §1.6), accompagna, benché talora "in background", tutto lo svolgimento del progetto: eventi imprevisti, fatti nuovi, variazioni nelle specifiche del committente, possono portare a rivedere il piano operativo, eventualmente modificando la struttura, la tempistica o i costi delle attività ancora da eseguire. Del resto, si ritrova qui un discorso applicabile a quasi tutti i modelli matematici, che sono validi nella misura in cui e fin tanto che le

condizioni che erano alla base della sua costruzione rimangono valide. È importante a questo proposito utilizzare modelli che siano in grado di "riconfigurarsi" per far fronte alle mutate condizioni ambientali.

Una caratteristica fondamentale di un progetto è, infine, il fatto che la missione complessiva che il progetto è chiamato a realizzare sia suddivisibile in *fasi*, e queste, in ultima analisi, in molteplici *attività* elementari, che saranno poi l'oggetto centrale dei nostri modelli. Ovviamente, cosa sia da considerarsi "elementare" o no dipende molto dal contesto; inoltre, due project manager diversi possono avere una percezione diversa su questo punto. Comunque, tutto il *ciclo di vita* del progetto è sostanzialmente volto a definire, pianificare, eseguire e controllare (in una sola parola: gestire) lo svolgimento di tutte le attività. Dal modo però in cui le attività del progetto sono organizzate e gestite dipende in modo cruciale la sua riuscita. Come fatto rilevare da Damiani et al (2004), ancora oggi l'approccio prevalente nell'impostare un progetto è quello di seguire la tipica sequenza temporale: "definire le attività—pianificarle temporalmente—eseguirle". Per questo motivo, molti identificano il corpo metodologico del project management esclusivamente con le metodologie matematiche che riguardano la pianificazione – e che anche noi vedremo in un certo dettaglio in questo corso. È però importante avere sempre presente che proprio l'impossibilità di prevedere tutti gli eventi che possono avere impatto sulla riuscita del progetto, costringe in genere a mantenere un approccio più flessibile e aperto a eventuali modifiche e revisioni della pianificazione del progetto.

1.2 Progetti e strutture organizzative

Il project management ha come campo d'azione l'insieme dei processi che è necessario attivare e gestire per portare avanti un progetto. In effetti, più che di gestione *dei* progetti, come è prassi comune, sarebbe invece più corretto parlare di gestione *per* progetti, indicando con questo termine una modalità di organizzazione dei processi aziendali che si contrappone ad altre, più consolidate e tradizionali. Non vogliamo affrontare qui nei dettagli tutte le complesse problematiche dell'organizzazione aziendale, ma solo evidenziare gli aspetti innovativi e peculiari della gestione per progetti.

Organizzazione funzionale

In linea di massima, la classica organizzazione *funzionale* è gerarchica, ossia esistono precise ed esplicite relazioni di supremazia e subordinazione tra i dipendenti, nell'ambito di reparti o settori relativamente disgiunti: progettazione, produzione, ricerca e sviluppo, marketing etc. La forte strutturazione delle funzioni e delle responsabilità che ne deriva, dà luogo a un ambiente estremamente "stabile" e in cui i ruoli e il "campo di azione" delle varie persone sono facilmente identificabili. Un'organizzazione di questo tipo costituisce la risposta più efficace a un mercato relativamente stazionario, in termini tanto di tipologia che di volume di beni prodotti. La sua rigidità è compensata dall'efficienza con cui i processi aziendali possono essere condotti.

I principi che hanno ispirato l'organizzazione funzionale sono oggi messi in discussione dalle sfide poste dal "rapidissimo tasso di innovazione delle tecnologie", che porta con sé "processi continui di ri-segmentazione dei mercati su scala regionale e globale. Questi

processi sono più evidenti in settori come la microelettronica, le biotecnologie, la comunicazione digitale, ma riguardano tendenzialmente tutte le imprese che operano in ambienti caratterizzati da elevata incertezza e volatilità dei mercati" (Gagliardi 2000). Inoltre, l'azienda si trova tipicamente di fronte clienti o committenti che, molto più che nel passato, esprimono esigenze sempre più personalizzate e che quindi richiedono soluzioni "su misura". Un'organizzazione strettamente gerarchica, con le sue inerzie e le sue rigidità, non è sempre quella più indicata a muoversi nei tempi giusti in questi nuovi scenari. La necessità di garantire rapide risposte alle richieste del cliente e di continui cambiamenti/aggiornamenti nella natura del servizio fornito, fanno sì che i livelli più bassi ed esecutivi dell'organizzazione siano investiti di maggiori responsabilità rispetto al passato.

Proviamo a esemplificare. La classica organizzazione funzionale è una gerarchia in cui ogni impiegato ha un unico superiore, chiaramente determinato (il "capo"). Il personale è inquadrato per funzioni, e un numero tipicamente limitato di persone è impegnato in progetti. Comunque, i componenti di un'unità funzionale partecipano al progetto nella misura in cui questo riguarda la loro funzione, indipendentemente dalle altre, e questo rende poco fluido il passaggio delle informazioni. Si consideri il progetto di sviluppo di un nuovo prodotto (es. un cellulare). Il progetto vedrà coinvolto personale della funzione Progettazione. Se a un certo punto, per fare una scelta di progetto (es. usare una batteria all'idrogeno), occorre avere informazioni relative al processo produttivo (es. con le nostre macchine riusciamo ad assemblare cellulari con batterie all'idrogeno?), il project manager passerà la domanda al capo-funzione, che si consulterà con il capo della funzione Produzione, mentre la risposta seguirà il percorso contrario. Come si può vedere, l'organizzazione funzionale impone ai flussi decisionali di attravere tutta la piramide organizzativa, quando invece in molti casi occorre dotarsi di strumenti di risposta rapida ed efficace (Damiani et al 2004).

Organizzazioni orientate ai progetti

Ecco allora che prendono forma, all'interno dell'azienda, strutture organizzative (i project team, appunto) che nascono con lo scopo di compiere una missione specifica (esempio: sviluppare e testare un nuovo ammorbidente), e che vedono la partecipazione di diversi soggetti, aventi tipicamente competenze e funzioni diverse nell'ambito dell'azienda (esempio: il settore ricerca, il settore marketing, il settore produzione). Le persone che compongono il team di progetto continuano a far parte delle rispettive aree e funzioni aziendali, ma vengono "prestati" al progetto, almeno per una percentuale di tempo, per tutto l'arco temporale in cui il progetto si sviluppa. Dunque la gestione per progetti può aver luogo anche in un'azienda organizzata in modo tradizionale; è chiaro però che già alcuni aspetti fondanti del progetto, come la costituzione del gruppo di lavoro e l'assegnazione delle responsabilità, possono dar luogo a potenziali conflitti con gli interessi di diverse aree aziendali (specificamente, delle aree a cui appartengono i componenti del team). Inoltre, spesso il project manager *non* fa parte dell'azienda ma è un libero professionista chiamato appositamente per dirigere il progetto, e dunque non ha alcun "potere" immediato (nel senso di rapporto gerarchico) sugli altri componenti il gruppo. Dunque, mentre in un contesto gerarchico molte decisioni vengono imposte da un principio di autorità, nell'ambito del progetto è molto più rilevante l'*autorevolezza* del project manager e la motivazione

dei partecipanti al team.

L'estremo opposto rispetto all'organizzazione funzionale è l'organizzazione *orientata ai progetti*: in essa, gran parte delle risorse dell'organizzazione sono impegnate su progetti, e i project manager hanno ampia autonomia e autorità sui membri del proprio team. Anche in queste organizzazioni esistono le funzioni (o dipartimenti), ma i suoi componenti in genere dipendono dai rispettivi project manager, o svolgono funzioni di supporto comuni ai vari progetti.

Riassumendo, le strutture organizzative possono essere descritte secondo uno spettro continuo che ai due estremi ha l'organizzazione funzionale e quella orientata ai progetti. Tra questi due estremi vi sono ovviamente molte soluzioni intermedie. Si hanno cioè le organizzazioni *a matrice*. Nelle matrici *deboli*, esiste una figura esterna ai dipartimenti, che però ha scarsa autorità e dunque ha più il ruolo di un supervisore/coordinatore che di un project manager vero e proprio. Nelle matrici *forti*, viceversa, accanto ai vari dipartimenti esiste una vera e propria funzione di project management, con un responsabile che coordina i vari project manager.

In realtà, in molti casi l'organizzazione risultante è un misto delle diverse strutture. Ad esempio, anche un'organizzazione funzionale può decidere di creare un team ad hoc per gestire un progetto particolarmente delicato, e a questo team può essere assegnato personale in modo permanente (per la durata del progetto) dalle varie aree aziendali, e può operare in generale con procedure (e.g. contabili) autonome rispetto al resto dell'azienda.

Dunque, in sintesi, la gestione per progetti nasce principalmente per un'esigenza di flessibilità. Si può dire che, in una certa misura, si sta verificando, per le forme organizzative aziendali, un'evoluzione parallela a quella avvenuta (qualche decennio prima) relativamente alla struttura fisica degli impianti (e.g. manifatturieri): in quel caso, da layout orientati al prodotto (catene di montaggio, transfer-lines etc), concepiti per produrre in modo efficiente un volume elevato di una gamma molto limitata di prodotti, si è passati in molti casi a layout orientati al processo (flexible manufacturing systems, centri di lavorazione, job shop etc), in cui diverse isole di lavorazione, eventualmente riconfigurabili, erogano determinate operazioni: questo tipo di layout è giustificato appunto quando l'impianto deve produrre prodotti diversi, eventualmente in quantità limitata, e ciascun prodotto richiede lavorazioni specifiche e individuali.

Stili e culture

Infine, sarebbe errato credere che la struttura organizzativa determini in modo completo la configurazione di un progetto. Se si confrontano progetti, anche aventi lo stesso scopo, ma che hanno luogo in aziende diverse, lo svolgimento del progetto è inevitabilmente influenzato non solo dalla struttura, ma anche dalle procedure, dalla cultura, e dagli scopi delle organizzazioni che danno vita al progetto stesso. Diverse aziende o organizzazioni condividono valori, procedure e concezioni diverse, che si riflettono anche sul proprio modo interno di operare, e dunque sullo "stile" che un project manager e il suo team dovranno adottare andando a operare in un contesto piuttosto che in un altro. Ad esempio, un team che proponga un'ipotesi tecnica con margini di rischio elevati avrà più successo in un ambiente imprenditoriale aggressivo. Un project manager portato a stabilire rapporti informali e "di parità" coi membri del team potrà incontrare difficoltà in un ambiente

fortemente gerarchico (e viceversa, un project manager con uno stile autoritario potrebbe avere problemi in un ambiente meno rigido). Uno stesso tipo di progetto (esempio, trattamento di un terreno agricolo per destinarlo alla piantagione di ananas) può essere portato avanti in modo molto diverso a seconda della cultura e dei valori dell'organizzazione in cui il progetto ha luogo, e non solo per motivi dimensionali (esempio, una multinazionale vs. una cooperativa di piccoli produttori).

1.3 Le aree di competenza del project management

Diversi autori hanno intrapreso lo sforzo di definire, e così in qualche modo circoscrivere, l'universo di competenze che un (buon) project manager dovrebbe avere. La schematizzazione effettuata da Damiani et al (2004) risponde bene a questa esigenza di sistematizzazione, in quanto raggruppa le competenze di interesse per il PM in quattro categorie fortemente distinte e caratterizzate: *applicative*, *organizzative*, *relazionale-sociali*, *metodologiche*.

Le competenze di tipo applicativo sono forse quelle più ovvie, e su cui meno si può dire a livello generale. Per gestire il progetto di sviluppo di un nuovo aeromobile saranno necessarie competenze tecniche ben diverse da quelle relative al progetto di realizzare, ad esempio, un sito web per un mobilificio. Si noti soltanto che le differenze in termini di skill non riguardano solo il settore aziendale, in questo esempio molto diverso, ma possono riguardare anche diverse funzioni aziendali all'interno dello stesso settore: ossia, un conto è dirigere un progetto di sviluppo di un nuovo ammorbidente, un altro è dirigere un progetto di campagna pubblicitaria per quello stesso prodotto. In effetti, il problema se il project manager debba essere uno "specialista" o un "generalista" è ampiamente dibattuto dai teorici dell'organizzazione aziendale. Senza entrare nel merito della disputa, ci limitiamo a osservare che nella pratica, il percorso professionale degli individui si sviluppa a partire da ruoli più tecnici (e quindi specialistici) a ruoli più gestionali (e quindi generalisti). (Questo del resto avviene tipicamente anche nelle altre aree del management.)

Per competenze di tipo organizzativo si intende la capacità di interagire con la struttura organizzativa dell'azienda e dell'ambiente in cui il progetto si svolge, comprendendone le implicazioni e i potenziali problemi che questa potrebbe porre al progetto. Questo fatto è importante in quanto, come si ricordava, i componenti del project team non sono in genere subordinati in modo permanente al project manager, e dunque sarà lui o lei in qualche misura a doversi adattare alle pratiche e alle "usanze" della realtà in cui va a operare. Se ad esempio nell'ambito di un progetto di ristrutturazione di una linea di produzione manuale nasce l'esigenza di avere dei dati relativi al processo attualmente in uso, il project manager dovrà valutare se richiedere di misurare i tempi con cui gli operai compiono determinate operazioni o se invece tentare altre strade, considerando che l'intervento di cui sopra potrebbe essere visto dal reparto produzione come una fastidiosa "invasione di campo", innescando così dinamiche pericolose all'interno dell'azienda a cui il progetto si rivolge.

Le competenze di tipo relazionale-sociale sono strettamente legate a quelle di tipo organizzativo, e fanno riferimento al fatto che il progetto è svolto, in definitiva, da un gruppo di persone. In quest'ambito rientrano diversi temi, quali la gestione dei gruppi,

la motivazione, la comunicazione (verbale e non), il comportamento organizzativo, le tecniche di negoziazione etc. Si tratta delle competenze più "soft" (nel senso di non matematiche), e riguardano aspetti che vanno ben al di là dell'ambito specifico del project management.

Infine, vi sono le competenze di tipo metodologico, che costituiscono l'oggetto principale del nostro corso. Si tratta della capacità di modellare situazioni decisionali e di utilizzare strumenti teorici per pianificare e controllare l'evoluzione del progetto nel migliore dei modi (rispetto agli obiettivi del progetto stesso). Porremo particolare enfasi sulla fase di pianificazione, da cui dipende in modo fondamentale l'organizzazione delle attività e l'assegnazione dei compiti all'interno del project team.

Parlando di aspetti metodologici, occorre però ancora una volta fare attenzione a non essere troppo restrittivi: *metodologico* non è sempre sinonimo di *matematico* (almeno, non quando si parla di project management...). Benché gli strumenti di tipo quantitativo siano spesso identificati con modelli matematici (analitici o di ottimizzazione), vi sono anche qui aspetti più qualitativi che non di meno hanno estrema rilevanza. Si consideri ad esempio un progetto il cui scopo è la valutazione dello stato di una centrale elettrica, ai fini di una sua eventuale dismissione. In questa valutazione, è importante avere indicazioni sulla verosimiglianza con cui determinati eventi (es. un incidente nella centrale) potranno accadere. Per una stima attendibile di tali probabilità, occorreranno senz'altro competenze di tipo tecnico, ma per poter utilizzare tali competenze correttamente all'interno del problema di decisione, occorre anche la capacità di quantificare correttamente tali informazioni, che sono necessariamente incerte. In particolare, questo vuol dire saper riconoscere se il parere di un esperto è influenzato o meno da fattori di disturbo (chiamati *bias cognitivi* e *motivazionali*). Ad esempio, se l'esperto è coinvolto in qualche modo nell'eventuale output del progetto, occorre valutare attentamente se un suo parere risente o meno di questa circostanza: se ad esempio l'esperto è un dipendente della centrale, e la sua dismissione implicherebbe che l'esperto rimane senza lavoro, c'è da aspettarsi che – magari inconsciamente – l'esperto tenda a sminuire la probabilità di eventi catastrofici.

1.4 Il ciclo di vita del progetto

Dal momento che i progetti, come si è detto, mirano alla realizzazione di qualcosa di nuovo e unico, presentano un elevato livello di incertezza. Per riuscire a controllare in modo efficace l'evoluzione di un progetto, in genere si usa dividere un progetto in fasi, che complessivamente delineano il ciclo di vita del progetto.

Ogni fase del progetto ha la missione di produrre uno o più oggetti (*deliverables*), che sono prodotti tangibili e verificabili, come ad esempio uno studio di fattibilità, un progetto dettagliato, o un prototipo funzionante di un dato prodotto. La conclusione di una fase del progetto è il momento, in genere, in cui si fa il punto del progetto, sulla base dei deliverables prodotti e sui dati (di tempo/costo) sull'andamento del progetto fino a quel momento.

La struttura del ciclo di vita di un progetto può essere altamente variabile; tuttavia alcune caratteristiche sono abbastanza comuni. In particolare, all'inizio la possibilità

di influenzare il prodotto finale del progetto è massima, e decresce progressivamente, all'avvicinarsi della conclusione, in quanto i costi di riconfigurazione e di correzione di eventuali errori sono sempre più alti (in altre parole, il progetto va via via "irrigidendosi" allorché una determinata ipotesi tecnica viene portata avanti in modo sempre più irreversibile).

Si noti che il concetto di ciclo di vita di un progetto non va confuso con quello di ciclo di vita di un prodotto: ad esempio, il progetto che ha come scopo lo sviluppo di un nuovo PC portatile è solo una fase (la prima) del ciclo di vita del prodotto.

Non esistono regole fisse per specificare il numero, la durata e il livello di dettaglio delle varie fasi. Diciamo che in linea di massima il numero tipico è 4-5, ma si può arrivare anche a molte di più o, nei casi più semplici, a fasi uniche. In genere, comunque, il passaggio da una fase a una successiva è abbastanza marcato, per cui gli strumenti di pianificazione che vedremo vanno pensati sostanzialmente per ciascuna fase, o almeno per quella o quelle "centrali" nello sviluppo del progetto. Per illustrare questa varietà, vediamo alcuni esempi di cicli di vita di progetti.

Morris (1988) illustra un tipico progetto nel campo dell'edilizia, diviso in quattro fasi:

- Fattibilità. Formulazione dell'ipotesi tecnica, progetto, studio di fattibilità. Al termine di questa fase si valuta se procedere o meno.

- Pianificazione. Definizione di massima di tempi, costi, condizioni di contratto, scadenze, descrizione delle attività. In questa fase si costruisce la WBS del progetto. Al termine di questa fase sono stipulati i contratti coi subfornitori.

- Costruzione. È la fase centrale e operativa del progetto, che termina con la sostanziale conclusione dell'impianto.

- Startup. Collaudo e finitura. La fase si conclude con la consegna dell'impianto.

Murphy (1989) illustra invece il ciclo di vita di un tipico progetto di introduzione di un nuovo farmaco.

- Ricerca e screening. Ricerca di base e applicata per scoprire un farmaco adatto a trattare una determinata patologia.

- Sviluppo pre-clinico. Individuato un principio, in questa fase si conducono test di laboratorio, per determinare sicurezza e efficacia del farmaco, e parallelamente si avviano le procedure per richiedere la sperimentazione clinica del farmaco, se lo sviluppo pre-clinico va a buon fine.

- Sperimentazione. È la fase centrale del progetto, che consiste di una serie di analisi, principalmente cliniche (tossicità, metabolismo, efficacia etc) ma anche legate al processo produttivo (costi di produzione, stabilità del principio attivo etc) e termina con la definizione finale della formula.

- Postsubmission. Superati tutti i tipi di test, per ottenere l'approvazione ministeriale occorrono ancora diverse attività di supporto per ottenere l'autorizzazione al commercio del nuovo farmaco.

Va osservato che in questo caso, l'intero ciclo di vita può facilmente arrivare a coprire oltre 10 anni.

Una struttura diversa del ciclo di vita è quello a spirale, tipico dei progetti di sviluppo software: in questi progetti, una stessa successione di fasi viene ripetuta più volte. Ad esempio, Muench et al (1994) descrivono un tipico progetto di sviluppo software in cui si reiterano più volte quattro passi fondamentali, applicandoli a versioni successive dello stesso prodotto: 1) identificare le specifiche, 2) progetto del programma, 3) implementazione, 4) test/valutazione.

1.5 I processi del project management

Il ciclo di vita di un progetto descrive una suddivisione delle attività in poche fasi, che hanno luogo in successione temporale. Il concetto di processo indica invece quelle azioni di coordinamento e pianificazione che hanno luogo durante tutto lo svolgimento del progetto, e che costituiscono, in definitiva, i compiti del team di progetto.

Possiamo distinguere[2]:

- Processi di pianificazione
- Processi esecutivi
- Processi di controllo

Questo corso verterà principalmente sui processi di pianificazione, che per certi versi costituiscono il nucleo fondamentale dell'attività del project manager e del team di progetto, e che vedremo nel §1.6.

I processi esecutivi sono quelli consistenti propriamente nell'esecuzione delle attività previste dal progetto, e dunque la loro natura dipende in modo essenziale dal tipo di progetto che si sta considerando. Fanno parte dei processi esecutivi però anche tutti quelli relativi al supporto che il team dà all'esecuzione materiale del progetto. I più importanti sono: valutazione della qualità dell'output del progetto, man mano che esso viene prodotto; adozione di tecniche per migliorare lo "spirito di gruppo" (e.g. stabilire regole per risolvere i conflitti interpersonali, partecipare a seminari motivazionali...); circolazione delle informazioni all'interno del team; tutte le attività connesse all'acquisizione di beni o servizi (procurement) dall'esterno quando se ne presenta la necessità.

I processi di controllo hanno lo scopo di monitorare l'andamento del progetto e segnalare tempestivamente eventuali "derive" rispetto agli obiettivi, ai tempi, ai costi, al livello qualitativo previsti. Quando una discrepanza significativa viene rilevata, vanno messe in atto procedure opportune. Ad esempio, se – caso tipico – è ormai evidente che non si riuscirà a terminare un'attività nei tempi previsti, può essere necessario rivedere lo schedule delle attività future, e dunque ri-eseguire determinati processi di pianificazione. Un cambiamento può essere anche imposto da una variazione negli obiettivi del progetto, e dunque interessare più attività.

[2]Il PMBOK ne considera esplicitamente anche altri due, di inizializzazione e di chiusura

1.5.1 Il triplo vincolo

Da un punto di vista concettuale, il compito del project manager potrebbe riassumersi sinteticamente dicendo che consiste nell'abilità di integrare e mediare tra tutti i vari processi. Infatti, essendo comunque le risorse (in senso lato) a disposizione del project manager limitate, dedicare troppa attenzione a un singolo processo potrebbe distogliere da altri. Un'immagine che viene spesso usata dai project manager per illustrare questo concetto è il *triplo vincolo*, vale a dire la necessità di far fronte alle specifiche in termini di tempo, costi e qualità del risultato. Il compito del project manager è anche quello di rendersi conto quale, di questi tre, è quello determinante (*driver*), che si dovrà curare particolarmente. Ad esempio, nei progetti legati all'adeguamento dei sistemi informativi per il millennium bug, l'aspetto temporale era assolutamente prioritario (e spesso lo è, anche in casi meno estremi). Nello sviluppo di un nuovo farmaco, è invece prioritario il rispetto di requisiti che rientrano tra gli aspetti qualitativi.

1.6 I processi di pianificazione

Abbiamo già accennato al fatto che il processo di pianificazione non è qualcosa che ha luogo esclusivamente prima che il progetto abbia inizio, bensì è un processo che viene continuamente richiamato ogni qual volta si verifica una discrepanza tra quanto si era, appunto, pianificato, e quanto si va realizzando. In effetti, si usa distinguere tra *piano preliminare* e *piano esecutivo*. Il piano preliminare viene preparato prima che il progetto abbia inizio, e comprende un insieme di informazioni quantitative fondamentali, quali la definizione di tutte le attività, il fabbisogno di risorse, l'indicazione delle responsabilità, la schedulazione delle attività stesse, la stima dei loro costi. Il piano esecutivo può essere visto come un'istanziazione del piano preliminare. Ad esempio, se nel piano preliminare si è indicato che per una certa attività occorrono 2 mesi/persona di un ricercatore, nel piano esecutivo si indica, con nome e cognome, il ricercatore assegnato a quella attività per due mesi (o i due ricercatori assegnati per un mese). Analogamente, le date, che nel piano preliminare sono relative all'inizio del progetto, divengono invece assolute.

I processi di pianificazione sono numerosi. Nelle prossime sezioni analizziamo più da vicino i più rilevanti, vale a dire la pianificazione dell'ambito del progetto (*scope planning*), la pianificazione della gestione dei rischi, la pianificazione dei costi, la pianificazione temporale delle attività.

1.6.1 Pianificazione dell'ambito del progetto (scope planning)

La pianificazione dell'*ambito* del progetto consiste nel definire in dettaglio tutto ciò che deve e non deve far parte della missione del progetto. Tale descrizione implica sostanzialmente di segmentare il progetto in un numero (tipicamente elevato) di attività elementari. Cosa sia da intendersi per *elementare* non è in genere scontato, e varia da situazione a situazione. In genere, si considera elementare un'attività che, ai fini del progetto, non richieda di essere specificata in modo più dettagliato. Ad esempio, si consideri un progetto consistente in un'indagine di mercato. Una delle attività iniziali può essere, ad esempio, "1.Elaborazione del questionario". A seconda della composizione del project

team e degli skill presenti, questo livello di specificazione può essere appropriato. Se ad esempio vi è un esperto di marketing, questi sarà in grado di eseguire tutta l'attività, e di poterne altresì stimare tempi e risorse necessarie. Altrimenti, può essere necessario dettagliare ulteriormente questa attività, suddividendola ad esempio in "1.1 Definizione della lunghezza, dei contenuti e degli scopi del questionario", "1.2 Scrittura del questionario", "1.3 Scrittura della lettera di accompagnamento", ed eventualmente ciascuna di queste attività può avere esecutori e responsabili diversi.

Per passare dall'oggetto finale di un progetto alla specifica in termini di attività si utilizza un procedimento top down il cui risultato è la cosiddetta *Work Breakdown Structure* (*WBS*).

Si parte dal livello più alto suddividendo il task complessivo in alcune macro-attività; ciascuna di queste viene esplosa in un certo numero di task componenti, l'esplosione di tali componenti in rispettivi sotto-componenti... e così via fino al livello di dettaglio desiderato. A ogni livello si riduce il valore monetario e la complessità delle attività, fino a raggiungere una dimensione gestibile per la pianificazione e il controllo. Le foglie di questo albero di decomposizione sono le attività elementari, che rappresenteranno l'input fondamentale per il processo di pianificazione temporale. Spesso i nodi al penultimo livello corrispondono a work packages, aventi ciascuno un responsabile per l'esecuzione di tutte le attività che lo compongono.

In pratica, la WBS è per i progetti qualcosa di simile a quello che la Bill Of Materials (BOM) è per i prodotti – in effetti, nei progetti di innovazione manifatturiera, non di rado c'è una forte sovrapposizione tra le due strutture, allorché diverse fasi del progetto corrispondono allo sviluppo di diversi componenti di un nuovo prodotto. Non dimentichiamo però che gli elementi della BOM rappresentano oggetti fisici, quelli della WBS delle attività.

Chiaramente, il numero di livelli di una WBS dipende molto dal tipo di progetto e da chi lo esegue. Nel campo dei grandi progetti, la WBS può avere facilmente dieci o più livelli (nel senso di distanza dal nodo radice). È importante però sottolineare che per poter dire di essere giunti a identificare una attività elementare, è necessario poter associare a tale attività:

- Una persona o un gruppo di persone responsabili dell'esecuzione dell'attività
- Una stima della durata, o un range di possibili durate
- Una stima dei costi, eventualmente in relazione alle durate
- Un oggetto preciso che deve costituire l'output dell'attività. Tale oggetto (che può essere un oggetto fisico, ma anche una analisi economica, un algoritmo, un modello, un report, una decisione etc.) prende il nome di *deliverable*, ed è indispensabile per poter certificare l'avvenuta esecuzione dell'attività,
- Le relazioni che legano l'attività alle altre, in termini di dipendenza/precedenza
- Il fabbisogno di risorse richiesto (§3.1).

(Le ultime due voci caratterizzano l'attività, ma non sono tipicamente espresse nella WBS, bensì dal processo di pianificazione temporale e di pianificazione dei costi rispettivamente.) Segmentare un progetto è un'operazione di grande complessità e, come si può capire, di grande importanza. In pratica, preparare la WBS vuol dire enunciare a un elevato livello di dettaglio come il progetto dovrebbe evolvere. La WBS assume un'importante valenza di riferimento, non solo perché specifica quello che *va* fatto, ma anche perché quello che è escluso dalla WBS *non va* fatto. Ciò non impedisce, ovviamente, di aggiustare il tiro mentre il progetto è in corso; tuttavia lo sforzo iniziale di preparazione della WBS serve in genere a chiarirsi le idee, a discutere tutti gli aspetti realizzativi della soluzione adottata e a raggiungere un accordo unanime sullo *scope* del progetto.[3] Nel seguito, supporremo sempre che le attività non siano interrompibili. (L'interruzione di una attività è da considerarsi un evento eccezionale, che richiede, in genere, l'adozione di opportune procedure di gestione dei rischi.)

1.6.2 Pianificazione della gestione dei rischi

Col termine *risk management* si intende tutto quello che riguarda le azioni intraprese per far fronte a possibili imprevisti nell'andamento del progetto, con azioni commisurate all'importanza dei rischi e del progetto stesso. Quello che si intende col termine *rischio* è qualsiasi evento imprevisto che può avere un impatto sullo svolgimento del progetto. Ad esempio, un rischio può essere causato dal fatto che un determinato permesso di costruzione tardi ad arrivare e dunque metta in pericolo il rispetto delle scadenze. O ancora, un'altra causa di rischio è il fatto di avere risorse umane limitate; se una persona si ammala il progetto rischia di arenarsi. Ci sono diverse categorie di rischi:

- Rischi tecnici. Es.: fare affidamento su una tecnologia relativamente poco conosciuta, cambiamento rispetto a standard consolidati in corso d'opera, prefissarsi obiettivi irrealistici.

- Rischi gestionali. Sono i rischi connessi dall'adozione di procedure di management non ottimizzate. Ad esempio, l'adozione di schedule troppo vincolati o costosi, o che non tengono in conto la limitatezza delle risorse disponibili... possono derivare dalla non adozione (o dall'adozione approssimativa o errata) di modelli e algoritmi di ottimizzazione nelle fasi cruciali di pianificazione temporale

- Rischi organizzativi. Sono i rischi che derivano dall'ambiente organizzativo in cui si trova a operare il progetto. Ad esempio, la possibilità di improvvise interruzioni nel finanziamento del progetto, conflitti con altri progetti aventi componenti del project team in comune...

[3] Si noti che stiamo qui implicitamente assumendo, come sempre faremo, che la strada da seguire per soddisfare le specifiche (quella che in gergo si chiama *ipotesi tecnica*) sia stata già scelta. Alcuni teorici del project management sostengono che la fase ancora più preliminare di scelta dell'ipotesi tecnica debba essa stessa far parte del progetto, come una sorta di attività zero. Da un punto di vista formale cambia poco, ma da quello pratico vuol dire trattare alla stregua delle altre un'attività che, tipicamente, ha un elevato livello di incertezza sia sui tempi che sui costi.

- Rischi esterni. Sono legati a fattori che non sono sotto il controllo del project team: cambiamenti nella legislazione, cambiamenti nella proprietà dell'azienda committente... [4]

In genere si identificano due processi distinti di pianificazione, vale a dire una *pianificazione della gestione dei rischi*, e una *pianificazione della risposta ai fattori di rischio*. Il primo processo consiste nel definire come saranno identificati i fattori di rischio, in che modo si intende monitorare lo svolgimento del progetto dal punto di vista dei rischi, come distribuire ruoli e responsabilità, quali risorse finanziarie saranno allocate alla gestione dei rischi. Questa fase di pianificazione è distinta da quella in cui si analizzano nello specifico i singoli rischi, e che costituisce invece l'oggetto di un altro processo, quello del piano di risposta ai rischi. In astratto, tutta questa suddivisione può sembrare a prima vista eccessiva, ma risponde all'esigenza, particolarmente importante in progetti di un certo rilievo, di procedere in modo il più possibile strutturato.

La gestione dei rischi riveste notevole importanza nelle attività del project manager. A parte, come si è detto, i rischi legati a una conduzione scadente del project team, per gli altri tipi di rischio le metodologie di risposta sono tipicamente non quantitative, e non saranno coperte in questa dispensa.

1.6.3 Pianificazione temporale delle attività

La pianificazione temporale costituisce in un certo senso il cuore del project management, e per questo motivo vi dedicheremo ampio spazio. L'input alla pianificazione temporale è rappresentato principalmente dalla WBS. Tale pianificazione si compone di diversi sottoprocessi, i più rilevanti dei quali sono:

- Determinazione delle relazioni di interdipendenza tra attività (vincoli di precedenza)
- Stima delle durate delle attività
- Pianificazione della tempistica (*schedule*) del progetto.

Siccome vedremo in seguito con notevole dettaglio i modelli per il calcolo dello schedule del progetto, vogliamo qui dedicare alcune considerazioni ai primi due punti.

Vincoli di precedenza

L'esistenza di rapporti di precedenza tra attività deriva spesso da considerazioni di carattere tecnologico abbastanza semplici (e.g. per costruire i muri portanti bisogna prima aver scavato le fondamenta). In altri casi, tale dipendenza non è ovvia. Soprattutto nei progetti in cui si interviene su una realtà preesistente (ad esempio, nella ristrutturazione di una linea di produzione), è possibile assumere un'erronea dipendenza temporale tra due attività che invece possono benissimo essere condotte in parallelo, solo perché tradizionalmente queste attività sono state effettuate in sequenza. A sua volta, quest'ultima circostanza può essere motivata da una contingente scarsità di risorse: ad

[4] Il PMI esclude espressamente da questa categoria le "cause di forza maggiore", come disastri naturali o guerre civili, contro i quali invero anche il project team più volenteroso può fare ben poco.

esempio, nell'assiematura manuale di un radiatore per autovetture, uno stesso operaio provvede dapprima a assemblare due semi-scatole che dovranno contenere il radiatore e poi inserisce i cursori nella console del radiatore, ma non vi è motivo per cui queste attività debbano essere svolte in sequenza: con due operai, possono essere svolte in parallelo.

Comunque, determinare correttamente i rapporti di precedenza tra le varie attività del progetto e le risorse richieste per il loro conseguimento è un compito che tipicamente può risultare lungo ma non eccessivamente complesso.

Stima delle durate

Riuscire a prevedere in modo accurato la durata di ogni attività di un progetto è un compito, in generale, alquanto difficile. La difficoltà maggiore è dovuta alle incertezze e agli imprevisti che si avranno durante l'esecuzione del progetto.

La stima della durate delle attività è generalmente basata sull'esperienza di un supervisore dell'attività. Qui riemergono i problemi di assessment dell'informazione cui si accennava in precedenza: il supervisore non è necessariamente la persona più obiettiva per giudicare la propria attività, anche se probabilmente è la persona con più esperienza. Inoltre, poiché i progetti vengono pianificati prima della loro esecuzione, esistono diversi elementi di incertezza e rischio che devono essere stimati. La conseguenza è che i supervisori, per precauzione, tendono a riferirsi alle loro peggiori esperienze e i tempi di esecuzione vengono in definitiva sovrastimati. (La sovrastima aumenta all'aumentare dei livelli di management coinvolti nel processo di stima.)

A seconda dei casi, le durate delle attività possono essere considerate un'informazione nota, o essere variabili aleatorie. I modelli deterministici (come quelli che vedremo in questo corso) sono i più semplici, e si basano su ipotesi apparantemente restrittive. Tuttavia, la loro rigidità è spesso ripagata dalla possibilità di effettuare analisi che da un punto di vista qualitativo si rivelano essere significative anche a fronte di una (piccola) perturbazione nei dati.

Nei modelli deterministici, si associa un solo valore a ciascuna attività. Se la varianza non è elevata, si può ragionevolmente assumere come durata il valor medio: se un'attività può durare ad esempio fra i 9 e gli 11 giorni, si assume che duri 10 giorni.

Anche se ci occuperemo prevalentemente di modelli deterministici, vale la pena accennare al *PERT* (Project Evaluation and Review Technique), che del resto rimane l'unica via percorribile quando l'aleatorietà nelle durate delle attività è elevata. In effetti, si può giustamente osservare che una durata media di 10 giorni può corrispondere a tempi che variano da 9 a 11 giorni, oppure da 3 a 17 giorni: un modello deterministico non è in grado di tener conto di queste differenze. Il PERT assume che le durate delle attività siano variabili aleatorie tra loro indipendenti. In particolare, se d_i è la variabile aleatoria che esprime la durata dell'attività i, la funzione densità di probabilità $F(d_i)$ si suppone approssimata da una distribuzione *beta*. Indicando con a_i e b_i un valore minimo e massimo entro i quali è compresa con certezza la durata dell'attività, e con m_i la durata più probabile, il valore atteso della durata è circa $E(d_i) = (a_i + 4m_i + b_i)/6$ e la varianza è circa $Var(d_i) = (b_i - a_i)^2/36$. La scelta della funzione di distribuzione beta è legata alle sue proprietà, che permettono di calcolare facilmente la media e la varianza: in pratica, basta richiedere al supervisore esperto i valori a_i, b_i e m_i.

Purtroppo, non sempre le assunzioni teoriche richieste da tale funzione sono soddisfatte in pratica, e questo è uno dei motivi per cui in effetti oggi il PERT appare abbastanza superato. Inoltre, il PERT nasce come strumento per stimare la durata complessiva di un progetto, che è senz'altro un aspetto importante, in un progetto complesso, ma non il solo. I modelli deterministici sono in grado di incorporare più elementi del problema (e.g. la presenza di risorse limitate), pur rimanendo computazionalmente trattabili.

Come si è accennato, in generale la durata di un'attività può dipendere dalla quantità di risorse impegnate: un'attività che richiede complessivamente 48 ore/uomo per essere eseguita, può essere svolta in 8 ore da 6 persone, o in 4 ore da 12 persone. Alcuni modelli tengono esplicitamente in conto questo fatto; nei modelli più semplici, invece, occorre risolvere più volte uno stesso modello, utilizzando di volta in volta una diversa modalità di esecuzione (e quindi una diversa coppia durata/risorse). Naturalmente, un caso particolarmente semplice è quello in cui la disponibilità di risorse è illimitata, e la durata di un'attività non dipende dunque dalle risorse assorbite.

1.6.4 Pianificazione dei costi

Un aspetto cruciale del ruolo del project manager è quello di pianificare e tenere sotto controllo i costi del progetto. A tali costi concorrono diverse voci, che vedremo sommariamente. Torneremo sul processo di cost management in maggior dettaglio nel §3.1.

La voce di costo forse più ovvia relativamente a un progetto è costituita dai *costi di realizzazione* associati alle singole attività; costi che possono essere sia interni all'azienda committente che esterni (subcontraenti). I costi di realizzazione sono spesso la voce principale di costo, e sono quelli considerati nei modelli decisionali che vedremo nel §3.1. Tuttavia, accanto a questi vanno ricordati:

- Costi iniziali: connessi all'analisi delle richieste del committente e alla preparazione del piano preliminare.
- Costi di gestione e controllo: relativi a definire il piano esecutivo, rivedere i requisiti, valutare i deliverable, organizzare riunioni e parteciparvi, scrivere report sull'andamento del progetto...
- Costi di coordinamento: dal momento che il tempo delle persone costa, tutto il tempo dedicato alle varie attività di coordinamento (e.g. riunioni periodiche, missioni etc) va computato nei costi complessivi del progetto.

Tornando ai costi di realizzazione delle attività, va osservato che in parte questi saranno legati all'utilizzo di risorse che le attività richiedono. Se chiediamo a un fornitore esterno lo sviluppo di un codice software per eseguire una determinata funzione nell'ambito di un software complesso, tale fornitore si farà pagare. Tuttavia, un'altra componente dei costi è legata alla modalità con cui l'attività sarà eseguita: in primo luogo, in quanto tempo. Un compito tipico del project manager è infatti quello di determinare quali attività occorre accelerare, e in che misura, per rientrare ad esempio nelle specifiche temporali del progetto. Accelerare un'attività ha un certo costo (crashing cost), dalla cui struttura e proprietà dipende la complessità di alcuni importanti problemi gestionali (§3.2).

Anche se noi ci occuperemo solo degli aspetti di pianificazione dei costi, va sottolineato che *monitoraggio* e *controllo* dei costi sono elementi fondamentali nella gestione dei progetti. È infatti necessario verificare continuamente che i costi del progetto seguano le stime effettuate e che in definitiva il budget sia sempre sotto controllo. Uno strumento operativo utilizzato per monitorare e tenere sotto controllo i costi di realizzazione è la cosiddetta *baseline curve*, che diagramma i costi attesi fino al tempo t (Figura 1.1). Questa curva rappresenta l'output del processo di cost budgeting (§3.1), e deriva dal calcolo dei costi delle risorse allocate alle varie attività. Questa curva è anche meglio conosciuta come *curva a S*, dalla sua caratteristica forma, dovuta al presupposto empirico per cui dopo un terzo del tempo previsto per il progetto si dovrebbe essere speso circa un quarto del budget, e che analogamente nell'ultimo terzo di tempo si spende circa un altro quarto. Variazioni significative rispetto a questa curva possono evidenziare errori nella stima dei costi o dei tempi, e rendono quindi necessaria una ripianificazione del progetto in corso d'opera.

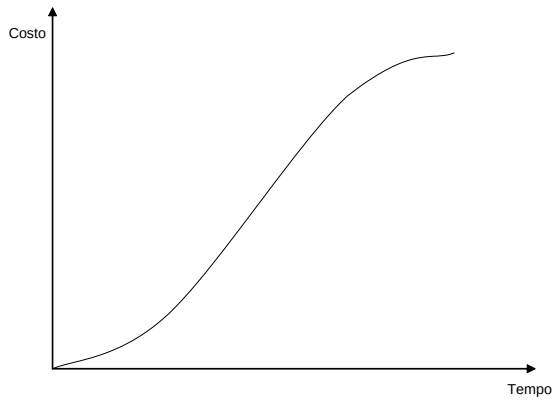

Figura 1.1: Tipica "curva a S" di un progetto.

In alcuni casi esiste un'altra possibilità, vale a dire sovrapporre parzialmente, al prezzo di un investimento maggiore di risorse, due attività che in teoria dovrebbero essere sequenziali (*fast tracking*). Tale modalità, oltre a comportare una lievitazione dei costi, si riflette anche, tipicamente, in un aumento del livello di rischio, dal momento che in genere implica che due diverse attività dovranno condividere alcune risorse. Nei modelli che vedremo non è contemplata questa possibilità.

1.6.5 Processi secondari di pianificazione

Oltre ai processi di pianificazione visti, ve ne sono molti altri "di supporto", che non esamineremo, ma tra i quali è opportuno almeno ricordare: identificazione degli standard di qualità e di come conseguirli; identificazione dei rischi e loro analisi qualitativa e quantitativa (in relazione agli obiettivi del progetto); acquisizione di risorse umane; assegnazione di ruoli e responsabilità per ciascuna attività del progetto; pianificazione della comunicazione (determinare le informazioni che i diversi attori del team dovranno ricevere per portare avanti la propria missione).

Capitolo 2

Analisi Temporale

In questo capitolo introduciamo alcuni strumenti per analizzare la rappresentazione di un progetto e trarne informazioni riguardanti sia il progetto complessivo che le singole attività. In particolare, ci occuperemo del problema di determinare in quanto tempo il progetto terminerà, e stabilire per ogni sua attività quando può o deve cominciare o terminare senza causare ritardi complessivi nella durata del progetto. Conoscendo queste informazioni, inoltre, è possibile stabilire per ogni attività quanto essa possa essere ritardata senza rallentare l'intero progetto.

Inizialmente, analizzeremo la situazione più semplice, ovvero supporremo che non vi sia nessun vincolo legato alla disponibilità di risorse, e i vincoli di precedenza tra attività sono di tipo finish-start, senza time-lag. Solo nel §2.5 ci occuperemo di vincoli di precedenza più generali e della presenza di time-lag.

La nostra attenzione si concentrerà sul calcolo del minimo tempo di completamento del progetto (anche noto come *makespan*) e sull'identificazione delle attività critiche e di altre grandezze d'interesse.

2.1 Rappresentazione di un progetto: reti di attività

Le reti di attività sono grafi che modellano le relazioni di precedenza che intercorrono tra le attività di un progetto. Storicamente, sono state introdotte negli anni '50 nell'ambito del PERT (tant'è che ancora oggi, impropriamente, molti utilizzano il termine PERT per indicare in realtà l'insieme dei vincoli di precedenza).

Diremo che l'attività x precede l'attività y (indicato con $x \prec y$) se y non può iniziare prima che x sia terminata. Relazioni di questo tipo prendono il nome di *finish-start*, e sono quelle più comuni. Come vedremo meglio nel §2.5, è possibile considerare anche relazioni di precedenza di tipo diverso.

Una situazione più generale si ha se y non può iniziare prima che sia trascorso un certo tempo $FS(x, y)$ dalla fine di x. La quantità $FS(x, y)^{\min}$ prende il nome di *time-lag*. Ovviamente, se $FS(x, y)^{\min} = 0$, siamo nella situazione precedente.

Ovviamente, la relazione di precedenza finish-start tra attività è transitiva, per cui verranno indicate solo le precedenze non implicate da altre (se $x \prec y$ e $y \prec z$, non scriveremo che $x \prec z$, in quanto implicato).

ESEMPIO 1 *Per la produzione di un nuovo prodotto sono state individuate le attività indicate nella seguente tabella. Per ciascuna attività sono specificati il tempo di esecuzione (deterministico) e le relazioni di precedenza.*

Attività	*Descrizione*	*Tempo (settimane)*	*Preceduta da*
1	*Disegno del prodotto*	*6*	-
2	*Disegno della confezione*	*2*	-
3	*Ordine e ricezione dei materiali per prodotto*	*3*	*1*
4	*Ordine e ricezione dei materiali per confezione*	*2*	*2*
5	*Realizzazione prototipo del prodotto*	*4*	*3*
6	*Realizzazione della confezione*	*1*	*4*
7	*Produzione lotto campione*	*1*	*5,6*
8	*Test di mercato del prodotto*	*6*	*7*
9	*Revisione del prodotto*	*3*	*8*
10	*Revisione della confezione*	*1*	*8*
11	*Presentazione dei risultati al CdA*	*1*	*9,10*

□

Una rete di attività è dunque un grafo orientato $G = (V, A)$. Esistono due possibili rappresentazioni di una rete di attività

- *Activity-on-Arc*: gli archi rappresentano le attività mentre i nodi esprimono i vincoli di precedenza. In particolare, i nodi corrispondono a eventi, quali il completamento di alcune attività (quelle corrispondenti ad archi entranti nel nodo) e l'inizio di altre (quelle uscenti): l'attività (i, j) non potrà iniziare prima che siano terminate le attività entranti nel nodo i, mentre nessuna attività uscente dal nodo j può iniziare prima che sia terminata (i, j) (e le altre attività entranti in j).
- *Activity-on-Node*: i nodi rappresentano le attività, gli archi rappresentano vincoli di precedenza: un arco da i a j indica che l'attività j non può iniziare prima che i sia terminata.

Sia che si utilizzi una rappresentazione AoA che AoN, una proprietà fondamentale della rete è che sia aciclica. Infatti, la presenza di un ciclo implicherebbe l'esistenza di una successione di attività x, y, z tale che $x \prec y$, $y \prec z$, e $z \prec x$, violando la transitività delle relazioni di precedenza.

2.1.1 Activity on Arc (AoA)

In questa rappresentazione, l'arco (u, v) rappresenta un'attività, che può iniziare dopo che tutte le attività entranti in u sono terminate.

Nella costruzione di una rete di attività AoA si seguono in genere alcune regole:

1. Ogni rete AoA ha un solo nodo iniziale (che rappresenta l'inizio di tutte le attività) e un solo nodo finale (la fine di tutte le attività). Il nodo iniziale non ha predecessori (*nodo sorgente*) e il nodo finale non ha successori (*nodo pozzo*). Tutti gli altri nodi hanno almeno un predecessore e almeno un successore.

2. Fra una coppia di eventi (nodi) c'è al più un'attività (un arco), per cui G si tratta di un *grafo semplice* (e non un multigrafo).

A queste andrebbe aggiunto il fatto che in genere i nodi vengono numerati in modo che gli archi vadano sempre da nodi con indice più piccolo a nodi con indice più grande. Su questo torneremo nel §2.2.2.

Per poter costruire una rete AoA associata a un insieme di attività e relative precedenze, rispettando le regole suddette, talvolta è necessario aggiungere attività o eventi fittizi (*dummy*); questo può accadere in vari casi:

1. In generale, non esiste un'unica attività iniziale o finale. Se ad esempio $a \prec c$ e $b \prec c$, sia a che b non hanno predecessori; basta allora introdurre un nodo e degli archi fittizi (Figura 2.1(a)). Per modellare correttamente un progetto si possono allora aggiungere un nodo fittizio (il nodo 1) per l'evento inizio progetto, e archi fittizi dal nodo 1 ai nodi che rappresentano l'inizio delle prime attività (sorgenti). In modo del tutto analogo, nel caso esistano più nodi-pozzo, basterà collegarli a un unico nodo che rappresenta la fine progetto.

2. Archi paralleli. Se alcune attività possono essere eseguite in parallelo, e hanno lo stesso insieme di predecessori e successori, sembrerebbe "naturale" collegarle tutte agli stessi nodi. Poiché però il grafo deve rimanere semplice, per mantenere la corrispondenza tra coppie di nodi e attività, bisogna introdurre attività ed eventi fittizi, come illustrato per le tre attività parallele (a, b, c) in Figura 2.1(b).

3. Rappresentazione dei vincoli di precedenza. Vi sono casi in cui la struttura dei vincoli di precedenza richiede una certa attenzione. Ad esempio, si considerino quattro attività a, b, c, d, tali che $a \prec c$, $b \prec c$ e $b \prec d$. La rappresentazione in Figura 2.2(a) è sbagliata, in quanto forzerebbe anche la relazione $a \prec d$. La rappresentazione corretta, chef a uso di un'attività fittizia, è mostrata in Figura 2.2(b).

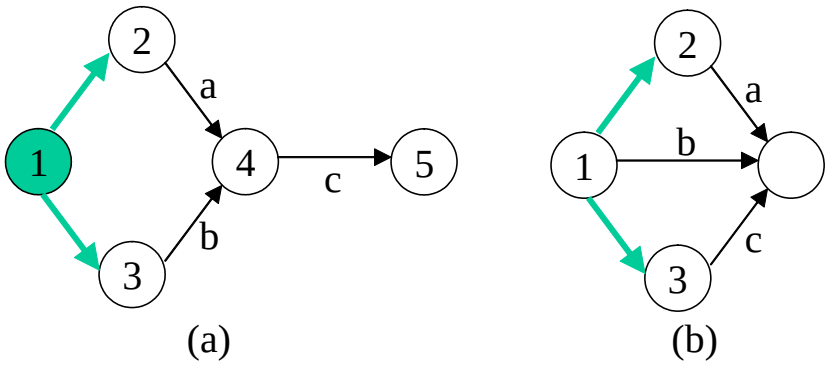

Figura 2.1: Attività fittizie e parallele in una rete AoA.

Per csotruire reti AoA corrette, si può allora procedere nel seguente modo, schematizzato in Figura 2.3: partendo dalle attività "slegate", si "incollano" via via per i nodi

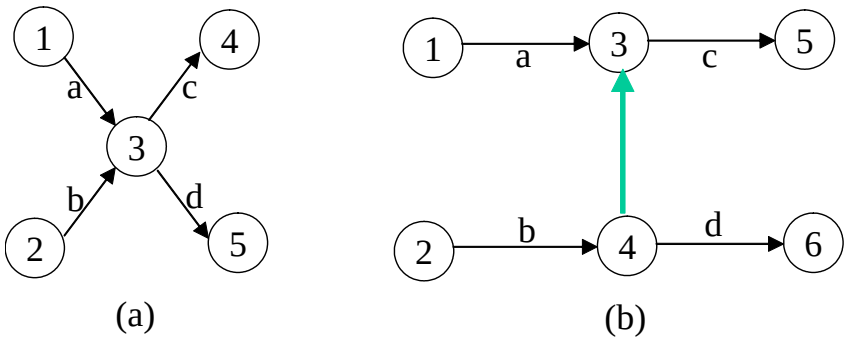

Figura 2.2: Vincoli di precedenza in una rete AoA.

Input: Elenco attività e relazioni di precedenza
Output: Rete AoA (non minimale)
begin
Associa a ogni attività un arco disgiunto
Aggiungi archi per rappresentare le relazioni di precedenza
Aggiungi i nodi di inizio e fine progetto e relativi archi fittizi
Contrai archi fittizi se ciò non crea precedenze inesistenti
end

Figura 2.3: Algoritmo per la costruzione di reti AoA

estremi, facendo attenzione a non introdurre precedenze che non fanno parte delle specifiche. Il seguente esempio illustra questo algoritmo.

ESEMPIO 2 *In Figura 2.4(a) è mostrato un esempio di contrazione di una rete AoA composta da 4 attività e 7 archi fittizi. Si noti che il quarto step dell'algoritmo in Figura 2.3 permette di ridurre la dimensione della rete eliminando un arco fittizio (Figura 2.4(b)).*

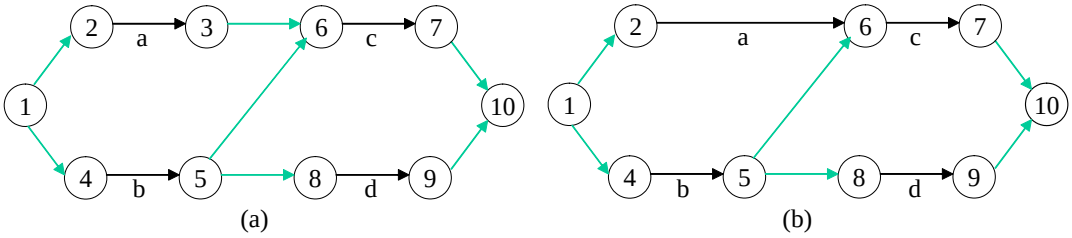

Figura 2.4: Contrazione degli archi fittizi in una rete AoA.

□

ESEMPIO 3 *Facendo riferimento all'Esempio 1, la sua rappresentazione in termini di rete AoA è mostrata in Figura 2.5.*

□

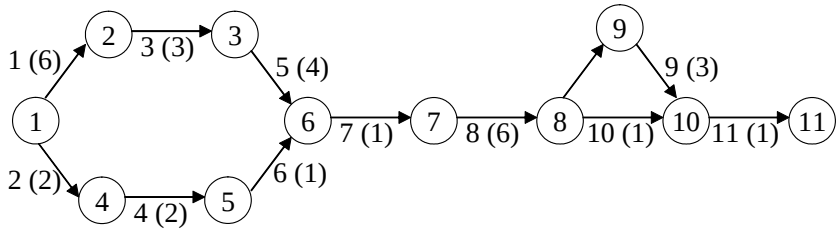

Figura 2.5: Rete AoA

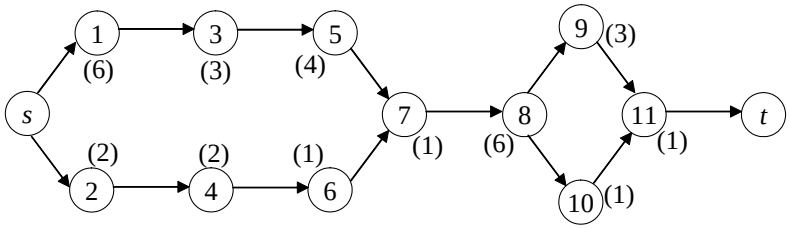

Figura 2.6: Rete AoN.

2.1.2 Activity on Node (AoN)

Nella rappresentazione Activity on Node (AoN), i nodi rappresentano le attività del progetto, mentre gli archi rappresentano le relazioni di precedenza tra le attività. In questa rappresentazione, gli unici nodi fittizi che può essere necessario aggiungere sono quelli di inizio e fine progetto.

Concettualmente, la rappresentazione AoN è forse più semplice di quella AoA, soprattutto perché non c'è bisogno di introdurre attività fittizie. Inoltre, come vedremo (§2.5), utilizzando una rete AoN si rappresentano più facilmente, rispetto alla AoA, altri tipi di relazioni tra attività, diverse da quella finish-start; per questo noi faremo prevalentemente uso di questa rappresentazione. C'è comunque da osservare che la rappresentazione AoA è spesso più compatta, e questo può essere utile soprattutto ai fini della comunicazione interna del project team.

Esempio 4 *La rappresentazione AoN dell'Esempio 1 è mostrata in Figura 2.6.* □

2.2 Durata di un progetto

Consideriamo un progetto rappresentato mediante una rete AoN $G = (V, A)$, in cui V rappresenta le attività del progetto, e A i vincoli di precedenza. È dato inoltre un vettore delle durate $d \in \mathbb{R}^{|V|}$, in cui d_i è la durata (nota e deterministica) dell'attività i.

Un piano temporale (*schedule*) delle attività è un'assegnazione di istanti di inizio alle attività. Tale assegnazione specifica completamente l'allocazione temporale delle attività, dal momento che queste ultime non sono interrompibili. Dunque, uno schedule è un vettore $s = [s_1, \ldots s_n]^T$, in cui s_i rappresenta l'*istante iniziale* dell'attività i.

Chiaramente, affinché un vettore s possa corrispondere a uno schedule ammissibile, andranno soddisfatti tutti i vincoli del problema (precedenze, risorse, ecc.). Ad esempio, se $d_a = 3$, $s_a = 1$ e deve essere $a \prec b$, chiaramente non può essere $s_b = 3$, in quanto b inizierebbe prima della fine di a.

Per comodità descrittiva introduciamo anche la variabile f_i che rappresenta l'*istante finale* dell'attività $i \in V$. Valendo l'ipotesi di durata deterministica, si avrà che $f_i = s_i + d_i$, per ogni $i \in V$. Poiché stiamo considerando soltanto relazioni di precedenza finish-start senza time lag tra due attività i e j, si ha il vincolo lineare $f_i \leq s_j$ per ogni $(i, j) \in A$, ovvero, eliminando le f_i, si ha $s_i + d_i \leq s_j$ per ogni $(i, j) \in A$.

La prima domanda che ci poniamo è dunque la seguente: qual è la durata (minima) del progetto? Siccome nella sua rappresentazione AoN vi sarà senz'altro un'attività di fine progetto, possiamo equivalentemente dire che vogliamo determinare il minimo valore della differenza tra l'istante di inizio dell'attività conclusiva e dell'attività iniziale (siano tali attività fittizie o reali), ovvero $s_n - s_1$. Questo obiettivo è anche noto in letteratura come *makespan*.

2.2.1 Formulazione come programmazione lineare

Dalla discussione precedente, si ha che, avendo per ora trascurato la presenza di risorse limitate o di un budget limitato, l'unico fattore che vincola l'esecuzione di ciascuna attività è il completamento delle attività che la precedono. Quindi, per formulare il problema di minimizzare il makespan del progetto basterà introdurre un vincolo per ciascuna attività. Dato dunque il grafo aciclico $G(V, A)$, in cui indichiamo con 1 il nodo iniziale e n il nodo finale, si ha:

$$\begin{aligned} \min\, & s_n - s_1 \\ & s_i + d_i \leq s_j \quad \forall (i,j) \in A \\ & s_i \geq 0 \quad \forall i \in V \end{aligned} \tag{2.1}$$

Si noti che sono presenti anche vincoli di non negatività sugli istanti iniziali delle attività, ovvero $s_i \geq 0$.

Esempio 5 *Per il progetto mostrato in Figura 2.7, il problema (2.1) si formula come segue.*

$$\begin{aligned} \min\, & s_8 - s_1 \\ & s_1 + 0 \leq s_2 \\ & s_1 + 0 \leq s_3 \\ & s_1 + 0 \leq s_4 \\ & s_2 + 2 \leq s_5 \\ & s_3 + 1 \leq s_6 \\ & s_4 + 5 \leq s_6 \\ & s_4 + 5 \leq s_7 \end{aligned}$$

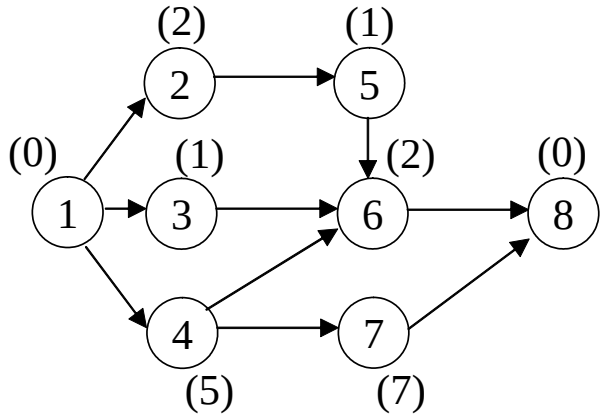

Figura 2.7: Esempio di progetto con rete AoN.

$$\begin{aligned} s_5 + 1 &\leq s_6 \\ s_6 + 2 &\leq s_8 \\ s_7 + 7 &\leq s_8 \\ s &\geq 0 \end{aligned}$$

□

Benché il problema del makespan possa essere affrontato come un problema di PL, la sua struttura è più particolare, come ora vediamo. Se riscriviamo i vincoli del problema (2.1) in forma di $\geq$, e rimuoviamo i vincoli di non negatività, si ottiene la seguente formulazione:

$$\begin{aligned} \min\, & s_n - s_1 \\ & s_j - s_i \geq d_i \quad \forall (i,j) \in A \end{aligned} \tag{2.2}$$

Il problema (2.2) è equivalente al problema originario. Infatti, pur avendo rimosso i vincoli di non negatività, possiamo sempre ottenere, da una soluzione del problema (2.2), una soluzione equivalente per (2.1): in particolare, se s^* è una soluzione ottima di (2.2) e $-k$ è il valore della sua componente più negativa ($k \geq 0$), ponendo $s'_i = s^*_i + k \;\; \forall i \in V$ si ha che s' soddisfa tutti i vincoli di (2.1) e le due funzioni obiettivo hanno lo stesso valore. Dunque, possiamo risolvere (2.2) anziché (2.1).

Scriviamo il duale del problema (2.2). Si osservi che i vincoli primali sono associati agli archi $(i,j) \in A$ del grafo G. Il termine noto d_i può essere quindi interpretato come lunghezza l_{ij} dell'arco (i,j). Per ogni $(i,j) \in A$, associamo al vincolo (i,j) la variabile (non negativa) duale x_{ij}. I vincoli duali sono associati ai nodi e sono di uguaglianza perché le variabili primali in (2.2) non sono vincolate. Indicando con $\delta^-(i)$ e $\delta^+(i)$ l'insieme degli archi rispettivamente entranti in i e uscenti da i, il problema duale risultante è:

$$\max \sum_{ij \in A} l_{ij} x_{ij} \tag{2.3}$$

$$\begin{aligned}
\sum_{(h,i)\in\delta^-(i)} x_{hi} - \sum_{(i,j)\in\delta^+(i)} x_{ij} &= 0 \quad i \in V - \{1,n\} \\
- \sum_{(1,j)\in\delta^+(1)} x_{1j} &= -1 \\
\sum_{(h,n)\in\delta^-(n)} x_{hn} &= 1 \\
x_{ij} &\geq 0
\end{aligned}$$

A questo punto è possibile riconoscere che la (2.3) è la ben nota formulazione del problema di determinare un cammino di lunghezza *massima* dal nodo 1 al nodo n in $G(V, A)$[1]. Poichè il grafo G è aciclico, possiamo essere certi che il problema è ben posto (non possono esistere cicli di peso positivo, il che, come vedremo più in là, potrebbe complicare le cose). Dalla dualità forte, deriva che all'ottimo le funzioni obiettivo dei due problemi hanno lo stesso valore: dunque, il makespan del progetto coincide con la lunghezza del cammino massimo sul grafo G, ove gli archi uscenti da ciascun nodo i sono pesati con la durata d_i di quella attività. Questa osservazione ci consente allora di affrontare il problema del calcolo del makespan per mezzo di algoritmi più specifici.

Dapprima vedremo un algoritmo per il problema del cammino massimo su grafi privi di cicli orientati di peso positivo, con sole precedenze di tipo finish-start. Nel §2.5.2 vedremo invece un classico algoritmo (Ford-Bellman), che ha il pregio di essere applicabile anche al caso di vincoli di precedenza più generali del caso finish-start.

2.2.2 Earliest start schedule

Per illustrare un algoritmo per il calcolo del cammino massimo su grafi aciclici, conviene introdurre ulteriori definizioni, che sono interessanti di per sé. Chiamiamo *earliest start schedule* uno schedule ammissibile $ES = (EST_1, \ldots, EST_n)$ avente la proprietà che, per ogni altro schedule ammissibile $s = (s_1, \ldots, s_n)$, si ha che $EST_1 \leq s_1, \ldots, EST_n \leq s_n$. In altre parole, in ES ogni attività inizia ad essere eseguita il prima possibile. Ciascun valore EST_i prende il nome di *earliest start time* (dell'attività i).

Inoltre, è conveniente adottare una particolare numerazione dei nodi del grafo G, detta *numerazione topologica*. Questa è tale che, se esiste l'arco (i, j) in G, allora il numero associato a i è inferiore a quello associato a j. Ossia, identificando il nome dell'attività con la sua numerazione, possiamo scrivere $i < j$. Vale allora il seguente teorema, di cui vediamo anche la dimostrazione, in quanto costruttiva.

Teorema 1 *Un grafo orientato G ammette una numerazione topologica dei nodi se e solo se G è aciclico.*

Dimostrazione (Solo se.) Per assurdo, supponiamo che vi sia un ciclo orientato $(v_1, v_2, \ldots, v_k)$. Poiché i nodi di G sono numerati topologicamente, deve essere $v_1 < v_2$. Similmente $v_1 < v_2 < v_3 < \ldots < v_k < v_1$ e quindi $v_1 < v_1$, assurdo.

(Se.) La dimostrazione della sufficienza è divisa in due parti.

[1]Si ricorda che le variabili duali sono vincolate soltanto a essere non negative, e non intere, in quanto la totale unimodularità della matrice dei coefficienti garantisce che tutte le soluzioni di base sono intere.

Input: G
Output: Ordinamento topologico
Inizializzazione: $G_1 = G$
begin
  for $j = 1, \ldots n$
    Trova una sorgente $v_j \in G_j$
    Assegna a v_j il valore j
    Poni $G_{j+1} = G_j - \{v_j\}$ (rimuovi v_j e tutti gli archi incidenti)
  end
end

Figura 2.8: Algoritmo per l'ordinamento topologico.

1. Se G è un grafo orientato aciclico, allora esiste almeno un nodo s privo di archi entranti (sorgente). Infatti, supponiamo al contrario che ogni nodo abbia almeno un arco entrante. Allora possiamo scegliere un nodo qualunque v, scegliere un arco entrante (u, v), passare al nodo u, scegliere un arco entrante (w, u), passare al nodo w, eccetera. Essendoci solo n nodi distinti, in al massimo n passi dovremo incontrare un nodo precedentemente già incontrato e quindi formare un ciclo.

2. La seconda parte della dimostrazione della sufficienza è costruttiva, e fornisce un algoritmo per la numerazione topologica. Si scelga una sorgente v_1 di $G = G_1$. Consideriamo ora il sottografo G_2 indotto dai nodi $V - \{v_1\}$ (i.e. ottenuto da G rimuovendo il nodo s_1 e gli archi incidenti in tale nodo). G_2 è ovviamente ancora aciclico, e quindi contiene almeno una sorgente v_2. L'argomento precedente può essere riapplicato a G_2 e così via, ottenendo una sequenza di grafi $(G_1, G_2, \ldots, G_n)$ e di nodi distinti $(v_1, v_2, \ldots, v_n)$ con la proprietà che v_i è una sorgente nel grafo G_i, per $i = 1, \ldots, n$. Mostriamo che la numerazione $(v_1 = 1, v_2 = 2, \ldots, v_n = n)$ è topologica. Infatti, per $i = 1, \ldots, n$, i nodi di G_i sono tutti e soli i nodi numerati da i a n, e inoltre il nodo i è una sorgente di G_i: quindi, eventuali archi entranti in i possono solo provenire da nodi j con $j < i$.

□

Dalla precedente dimostrazione, una numerazione topologica dei nodi di G può quindi essere calcolata in $O(|A|)$ con l'algoritmo riassunto in Figura 2.8.

Esempio 6 *In Figura 2.9 viene illustrato lo svolgimento dell'algoritmo di numerazione topologica 2.8 con riferimento all'Esempio 5.* □

Vediamo ora l'algoritmo per il calcolo del makespan. L'algoritmo procede determinando l'earliest start time dei vari nodi, seguendo l'ordinamento indotto da una numerazione topologica. Per fare questo, introduciamo l'*earliest finish time* EFT_i dell'attività i, definito come $EST_i + d_i$. Ricordando il significato di EST_i, si ha che EFT_i è il *tempo minimo di completamento dell'attività* i.

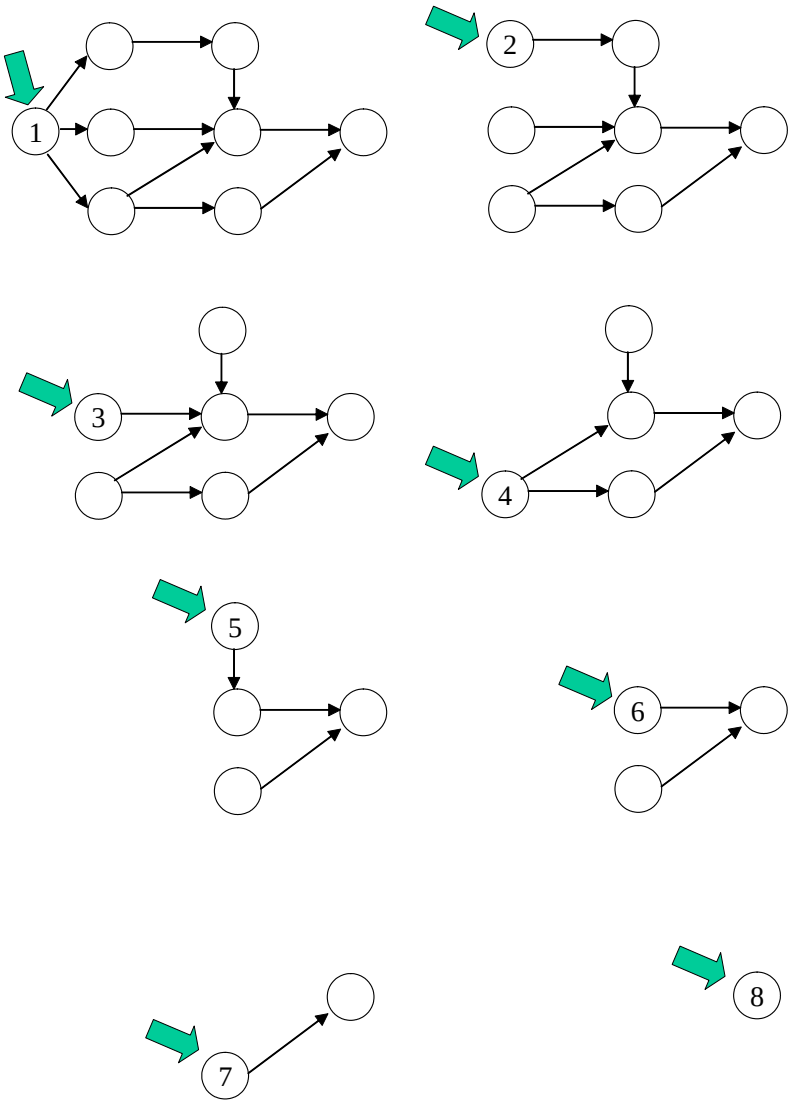

Figura 2.9: Esempio di svolgimento dell'algoritmo di ordinamento topologico per il grafo in Figura 2.7

.

Input: G
Output: EST, EFT e $prec$
Inizializzazione: $EST_1 = 0$, $EFT_1 = 0$, $prec_1 = 1$.
begin
  Ordina topologicamente i nodi di $G = (V, A)$
  for $j = 2 \dots n$
    $EST_j = \max_{i<j}\{EFT_i\}$
    $prec_j = \arg\max_{i<j}\{EFT_i\}$
    $EFT_j = d_j + EST_j$
  end
end

Figura 2.10: Algoritmo per il calcolo del makespan.

Il calcolo di EST_j (e quindi di EFT_j) è immediato, considerando che un'attività j può cominciare quando tutte le attività che la precedono sono terminate, ovvero si può scrivere

$$EST_j = \max\{EFT_i | i \prec j\} \tag{2.4}$$

In definitiva, si ha l'algoritmo ricorsivo per il calcolo del makespan illustrato in Figura 2.10, in cui $prec_j$ indica l'attività per cui è ottenuto il massimo nella (2.4).

Utilizzando le informazioni contenute nel vettore *prec* si può ricostruire, in particolare, il cammino massimo dal nodo 1 al nodo n. Tale cammino, di lunghezza pari al makespan del progetto, prende il nome di *cammino critico*, e in generale non è unico. Le attività appartenenti a un cammino critico sono dette *attività critiche*. Il cammino critico è importante perché un ritardo che si generi su una qualsiasi attività critica provoca un aumento nella durata del progetto stesso. Da un altro punto di vista, come vedremo nel §3.3, si ha che si può ridurre il makespan del progetto solo riducendo i tempi di esecuzione di attività critiche lungo tutti i cammini critici.

Esempio 7 *Con riferimento ai dati dell'Esempio 5, la tabella seguente mostra il valore degli earliest start time e finish time, nonché i predecessori sull'albero dei cammini massimi per ogni attività del progetto.*

j	EST_j	EFT_j	$prec_j$
1	0	0	-
2	0	2	1
3	0	1	1
4	0	5	1
5	2	3	2
6	5	7	4
7	5	12	4
8	12	12	7

□

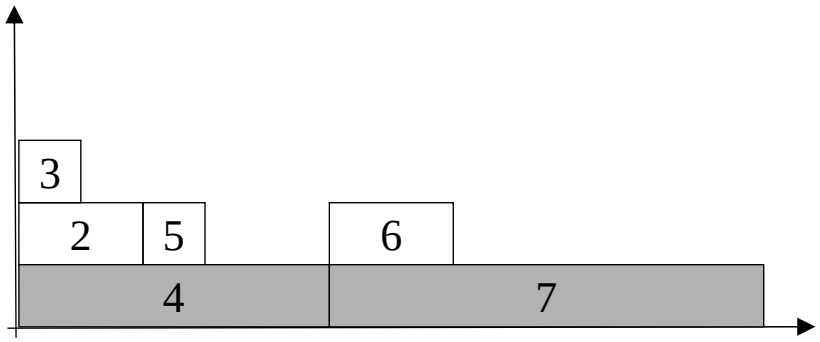

Figura 2.11: Diagramma di Gantt dell'earliest start schedule dell'Esempio 5.

2.3 Latest start time

Dato uno schedule s ammissibile, è possibile darne una rappresentazione grafica mediante un *diagramma di Gantt*, griglia bidimensionale che mostra la collocazione temporale delle varie attività, ciascuna rappresentata con una barra orizzontale di lunghezza pari alla sua durata. La Figura 2.11 mostra il diagramma di Gantt dell'earliest start schedule relativo all'Esempio 5, in cui sono evidenziate le attività critiche.

Poniamoci ora in un'ottica leggermente diversa, che corrisponde a molte situazioni reali. Ossia, supponiamo di non essere tanto interessati a terminare il progetto prima possibile, bensì a terminarlo entro una scadenza temporale (*deadline*) T. Ci poniamo allora la seguente domanda: quando possono cominciare *al più tardi* le singole attività senza che la deadline T sia violata? Indichiamo con *Latest Start Time* (LST_j) l'ultimo istante di tempo in cui l'attività j può cominciare senza che il progetto finisca in ritardo, e con $LFT_j = LST_j + d_j$ il *latest finish time*, e cioè l'ultimo istante di tempo in cui l'attività j può terminare senza che il progetto termini in ritardo. Inoltre il *latest start schedule* LS è lo schedule in cui tutte le attività del progetto iniziano all'istante LST_j. Il calcolo dei valori LST_j può avvenire in maniera simmetrica a quanto visto per gli EST_j. Basta infatti osservare che un'attività j può terminare al più tardi all'ultimo istante utile per l'inizio delle attività successive, dunque:

$$LFT_j = \min\{LST_i | i \succ j\} \tag{2.5}$$

e $LST_j = LFT_j - d_j$. Chiaramente, se il progetto deve terminare all'istante T, la (2.5) va inizializzata ponendo $LFT_n = T$.

Indichiamo con $succ_j$ l'attività i per cui si ha il minimo nella (2.5). In definitiva, l'algoritmo ricorsivo in Figura 2.12 calcola il latest start schedule. Il grafo avente come archi l'insieme delle coppie $(j, succ_j)$ è evidentemente un albero-pozzo con radice nel nodo n.

Nel caso particolare in cui T sia proprio pari al makespan del progetto, calcolato tramite la (2.4), su tale albero, il cammino che va da 1 a n è un cammino critico.

Esempio 8 *Con riferimento ai dati dell'Esempio 5, la tabella seguente mostra latest start e finish time, nonché il vettore* succ.

Input: G
Output: LST, LFT e $succ$
Inizializzazione: $LST_n = T$, $LFT_n = T$, $succ_n = T$
begin
  for $j = n - 1 \dots 1$
    $LFT_j = \min_{i>j}\{LST_i\}$
    $succ_j = \arg\min_{i>j}\{LST_i\}$
    $LST_j = LFT_j - d_j$
  end
end

Figura 2.12: Algoritmo per il calcolo di LST_j e LFT_j per ogni attività.

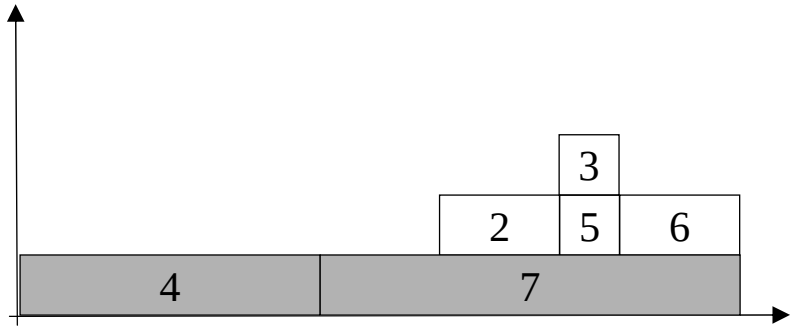

Figura 2.13: Diagramma di Gantt del Latest Start Schedule per l'esempio 5.

j	LFT_j	LST_j	$succ_j$
8	*12*	*12*	-
7	*12*	*5*	*8*
6	*12*	*10*	*8*
5	*10*	*9*	*6*
4	*5*	*0*	*7*
3	*10*	*9*	*6*
3	*9*	*7*	*5*
1	*0*	*0*	*4*

Il diagramma di Gantt rappresentante il latest start schedule è mostrato in Figura 2.13.

□

Valutare la complessità dell'algoritmo per il calcolo dell'Earliest Start Schedule è semplice (considerazioni del tutto analoghe valgono per il Latest Start Schedule): a ogni iterazione, si confronta il valore di EFT_i per tutti i predecessori immediati del nodo j. A ognuno di tali predecessori corrisponde l'arco $(i, j) \in A$. Nell'ambito delle $n-1$ iterazioni,

ogni arco è dunque scandito una e una sola volta, e la complessità è quindi $O(|A|)$ (lineare nel numero di archi).

2.4 Tempi di slittamento

I concetti di percorso critico e di earliest/latest start/finish schedule evidenziano il fatto che, nell'ottica di portare a termine un progetto nei tempi previsti, non tutte le attività hanno la stessa importanza. Cerchiamo di rendere queste considerazioni più precise definendo opportune grandezze associate a una rete di attività. Nel seguito, supporremo che la deadline del progetto sia pari al makespan.

Il *total float* TF_j dell'attività j è la quantità di tempo di cui può essere ritardata l'attività j, rispetto al proprio EST_j, senza che la deadline del progetto venga violata. In base alla discussione del paragrafo precedente, si ha che $TF_j = LST_j - EST_j = LFT_j - EFT_j$. Il total float (anche indicato su alcuni testi semplicemente come *tempo di slittamento*) misura il margine di libertà che si ha nello schedulare un'attività. Se (e solo se) il total float di una attività è 0, allora questa attività è critica, e dunque ogni attività su un cammino critico ha total float 0.

Il margine misurato dal total float è effettivamente disponibile solo se nessuno dei predecessori di j termina dopo EST_j e nessuno dei successori inizia prima di LFT_j, circostanze che invece possono benissimo verificarsi. In definitiva, la eventuale decisione di schedulare j in modo che termini all'istante LFT_j può, in generale, costringere altre attività a iniziare dopo il rispettivo earliest start time. Per questo motivo, sono state introdotte altre due misure (simmetriche) che esprimono il margine di libertà nella schedulazione di un'attività che non può essere compromesso da decisioni relative ad altre attività.

Si è visto che il total float è pari a $LFT_j - EFT_j$. Se però qualche successore di j è schedulato in modo da terminare prima del proprio latest finish time, questo può impedire a j di terminare a LFT_j. Di certo, comunque, nessuno potrà obbligare j a terminare prima di $\min_{i \succ j}\{EST_i\}$. Diciamo allora che il *free float* FF_j dell'attività j è la quantità di tempo di cui la fine dell'attività j può essere ritardata indipendentemente dalla schedulazione dei successori, ed è pari a $FF_j = \min_{i \succ j}\{EST_i\} - EFT_j$. In altre parole, che l'attività j inizi a EST_j o a $EST_j + FF_j$ non ha alcun impatto sulle attività successive.

Simmetricamente, se qualche predecessore di j inizia dopo il proprio earliest start time, anche j potrà essere obbligato a iniziare dopo EST_j; in nessun caso tuttavia sarà costretto a iniziare dopo $\max_{i \prec j}\{LFT_i\}$. Il *safety float* SF_j dell'attività j misura di quanto l'inizio dell'attività j può essere ritardato indipendentemente dalla schedulazione dei predecessori, ed è pari a $SF_j = LST_j - \max_{i \prec j}\{LFT_i\}$.

Poniamo al lettore, per esercizio, queste due domande: Se un'attività è critica, può avere FF_j o SF_j diversi da zero? E può un'attività avere *sia* FF_j *che* SF_j pari a 0 e non essere critica?

Esempio 9 *Con riferimento all'Esempio 5, possiamo calcolare tutti i valori earliest/latest start/finish time e i tre tempi di slittamento, come mostrato nella seguente tabella. Si noti che tutte e sole le attività critiche hanno total float 0.*

j	d_j	EST_j	EFT_j	LFT_j	LST_j	TF_j	FF_j	SF_j	
1	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	
2	*2*	*0*	*2*	*9*	*7*	*7*	*0*	*7*	
3	*1*	*0*	*1*	*10*	*9*	*9*	*4*	*9*	
4	*5*	*0*	*5*	*5*	*0*	*0*	*0*	*0*	*critica*
5	*1*	*2*	*3*	*2*	*10*	*7*	*2*	*0*	
6	*2*	*5*	*7*	*4*	*12*	*5*	*5*	*0*	
7	*7*	*5*	*12*	*12*	*5*	*0*	*0*	*0*	*critica*
8	*0*	*12*	*12*	*12*	*12*	*0*	*0*	*0*	*critica*

□

2.5 Relazioni di precedenza generalizzate

Oltre alle relazioni di precedenza di tipo finish-start, descritte fino ad ora, possono sussisterne altre. Come vedremo, queste altre *relazioni di precedenza generalizzate* possono essere espresse con *vincoli lineari* sulle variabili s e f.

Questi vincoli generalizzano quanto visto finora per due motivi: anzitutto, queste relazioni non legano solo l'istante finale di un'attività con quello iniziale di un'attività successiva, ma anche, tra loro, gli istanti iniziali, quelli finali, o quello iniziale con quello finale di un'altra attività. Inoltre, esistono in generale dei *time-lag* che impongono una distanza minima o massima tra questi istanti. Distinguiamo quattro tipi di relazioni:

1. Relazioni di tipo *start-start*. Vincolano gli istanti di inizio di due attività: l'attività j deve iniziare dopo almeno $SS(i,j)^{\min}$ dall'inizio dell'attività i:

   $$s_i + SS(i,j)^{\min} \leq s_j$$

   (relazione *di tipo min*), oppure: l'attività j deve iniziare al massimo dopo un tempo $SS(i,j)^{\max}$ dall'inizio dell'attività i:

   $$s_j \leq s_i + SS(i,j)^{\max}$$

   (relazione *di tipo max*).

2. Relazioni di tipo *start-finish*. Vincolano l'istante di inizio di un'attività con quello di fine di un'altra: l'attività j deve finire almeno dopo un tempo $SF(i,j)^{\min}$ dall'inizio dell'attività i:

   $$s_i + SF(i,j)^{\min} \leq f_j$$

   (relazione di tipo min), oppure: l'attività j deve finire al massimo dopo un tempo $SF(i,j)^{\max}$ dall'inizio dell'attività i:

   $$f_j \leq s_i + SF(i,j)^{\max}$$

   (relazione di tipo max).

3. Relazioni di tipo *finish-start*. Sono quelle utilizzate finora, con l'aggiunta però dei time lag. Dunque, l'attività j deve iniziare almeno dopo un tempo $FS(i,j)^{\min}$ dalla fine dell'attività i:

$$f_i + FS(i,j)^{\min} \leq s_j$$

(relazione di tipo min), oppure: l'attività j deve iniziare al massimo dopo un tempo $FS(i,j)^{\max}$ dalla fine dell'attività i:

$$s_j \leq f_i + FS(i,j)^{\max}$$

(relazione di tipo max).

4. Relazioni di tipo *finish-finish*. Vincolano gli istanti finali di due attività: l'attività j deve finire almeno dopo un tempo $FF(i,j)^{\min}$ dalla fine dell'attività i:

$$f_i + FF(i,j)^{\min} \leq f_j$$

(relazione di tipo min), oppure: l'attività j deve finire al massimo dopo un tempo $FF(i,j)^{\max}$ dalla fine dell'attività i:

$$f_j \leq f_i + FF(i,j)^{\max}$$

(relazione di tipo max).

Possibili applicazioni di queste relazioni sono identificabili frequentemente, in contesti anche molto diversi. Ad esempio, nella costruzione di un appartamento si può posare il parquet solo dopo che un certo tempo è trascorso dal completamento del massetto, al fine di permettere la sua completa asciugatura (finish-start). Analogamente, la spianatura dell'asfalto deve cominciare un po' dopo che si è cominciato a stendere l'asfalto, ma non troppo dopo (altrimenti l'asfalto si raffredda). Il lancio di un nuovo prodotto può cominciare dopo un po' che la campagna pubblicitaria è partita, ma non troppo dopo (start-start). Le attività di dismissione di un vecchio impianto produttivo possono essere completate solo un po' di tempo dopo che quello nuovo ha cominciato a operare (start-finish).

Veniamo ora al problema di trovare un piano delle attività che minimizzi il tempo di completamento in presenza di relazioni di precedenza generalizzate. Analogamente a quanto già visto nel §2.2.1, anche questo problema può essere affrontato in termini di programmazione lineare; tuttavia, la presenza di lag temporali richiede una certa attenzione.

Anzitutto, possiamo riscrivere i vincoli del problema in funzione delle sole variabili s, ricordando che $f_i = s_i - d_i$. Inoltre, possiamo porli tutti nella forma normalizzata di "$\leq$", ossia

$$s_i + l_{ij} \leq s_j$$

oppure

$$s_j + l_{ji} \leq s_i$$

dove l_{ij} e l_{ji} sono opportune costanti, chiamate *lag normalizzati*. Ad esempio, il vincolo $s_i + SS(i,j)^{\min} \leq s_j$ diventa $s_i + l_{ij} \leq s_j$, ove si ponga $l_{ij} = SS(i,j)^{\min}$. Il vincolo $s_i + SF(i,j)^{\max} \geq f_j$ diviene $s_j + l_{ji} \leq s_i$, con $l_{ji} = d_j - SF(i,j)^{\max}$. Procedendo in

maniera analoga è possibile ricavare tutte le operazioni di normalizzazione delle relazioni temporali, riassunte nella Tabella 2.1. Si osservi che il vincolo finish-start senza time lag ($FS(i,j)^{\min} = 0$) si traduce come $l_{ij} = d_i$.

Anche in presenza di vincoli di precedenza generalizzati, possiamo utilmente rappresentare il progetto per mezzo di una rete AoN. Mentre finora l'arco (i,j) indicava semplicemente che l'attività j doveva seguire la i, adesso utilizziamo gli archi per rappresentare anche l'informazione associata alle relazioni normalizzate. Precisamente, abbiamo che nella rete AoN esiste l'arco (k,q) dal nodo k al nodo q se e solo se esiste una relazione normalizzata specificata da un parametro di tipo $SS(k,q)$, $SF(k,q)$, $FS(k,q)$ o $FF(k,q)$, sia di tipo min che max. Da questa rete AoN è possibile ricavare una rappresentazione normalizzata, in cui cioè l'arco è orientato da k a q se la relazione è della forma $s_k + l_{kq} \leq s_q$. Si noti come la presenza di una relazione di tipo max tra due attività i e j porta all'introduzione di un arco dal nodo j al nodo i, ovvero orientato inversamente rispetto alla relazione. L'arco è pesato con il lag normalizzato l_{kq}. La rete ottenuta prende il nome di *constraint digraph* (grafo orientato vincolato).

Dato dunque un constraint digraph $G = (V, E)$, il problema di trovare un piano delle attività ammissibile che minimizzi il makespan può essere formulato come:

$$\begin{array}{rcll} \min s_n & & & \\ s_i + l_{ij} & \leq & s_j & \forall (i,j) \in E \\ s_i & \geq & 0 & \forall i \in V \end{array} \tag{2.6}$$

dove, ricordiamo, s_n è l'inizio dell'attività n (eventualmente fittizia) di fine progetto.

Vale la pena fare alcune osservazioni relative alla costruzione del constraint digraph. Anzitutto, se in un constraint digraph ci sono archi paralleli, basta considerare quello di peso massimo: infatti, se si hanno entrambi i vincoli $s_i + 2 \leq s_j$ e $s_i + 3 \leq s_j$, il secondo domina il primo, che può essere quindi omesso. In effetti, una stessa coppia di attività può essere legata da più relazioni di precedenza generalizzate: la più "stringente" delle relazioni di tipo min (ossia quella con l_{ij} massimo) obbligherà j a iniziare dopo l'istante $s_i + l_{ij}$, mentre la più stringente di quelle di tipo max (quella con l_{ji} massimo) vincolerà j a iniziare non oltre $s_i - l_{ji}$. In generale, si può dunque individuare un intervallo $[s_i + l_{ij}, s_i - l_{ji}]$, detto *finestra temporale* (time window) di s_j relativamente a s_i.

Si noti che in generale occorrerà imporre esplicitamente il fatto che ogni attività i termini prima che l'attività n inizi: infatti, adesso la presenza di un arco da i a j non garantisce più che j inizi dopo che i è finita: in una relazione start-start abbiamo solo che j inizia (almeno $SS(i,j)^{\min}$ istanti) dopo l'inizio di i, ma se per esempio j è molto lunga, è anzi probabile che termini *dopo* la fine di i. Perciò, se un nodo i non è coinvolto esplicitamente in relazioni $FS(i,j)^{\min}$ con qualche altro nodo j, si deve aggiungere al grafo un arco (i,n) di peso $FS(i,n)^{\min} = 0$. Solo laddove invece la fine di i preceda strettamente l'inizio di qualche altra attività j, questa relazione risulta essere ridondante, e l'arco (i,n) può dunque essere omesso. Per motivi del tutto simmetrici, aggiungeremo anche archi del tipo $(1,i)$ di peso $FS(1,i)^{\min} = 0$.

Esempio 10 *La figura 2.14 illustra la rete AoN di un progetto, in cui sono state aggiunte*

Relazione	Vincolo	vincolo norm.	lag normalizzato
$SS(i,j)^{\min}$	$s_i + SS(i,j)^{\min} \leq s_j$	$\rightarrow s_i + l_{ij} \leq s_j$	$l_{ij} = SS(i,j)^{\min}$
$SS(i,j)^{\max}$	$s_i + SS(i,j)^{\max} \geq s_j$	$\rightarrow s_j + l_{ji} \leq s_i$	$l_{ji} = -SS(i,j)^{\max}$
$SF(i,j)^{\min}$	$s_i + SF(i,j)^{\min} \leq f_j$	$\rightarrow s_i + l_{ij} \leq s_j$	$l_{ij} = -d_j + SF(i,j)^{\min}$
$SF(i,j)^{\max}$	$s_i + SF(i,j)^{\max} \geq f_j$	$\rightarrow s_j + l_{ji} \leq s_i$	$l_{ji} = d_j - SF(i,j)^{\max}$
$FS(i,j)^{\min}$	$f_i + FS(i,j)^{\min} \leq s_j$	$\rightarrow s_i + l_{ij} \leq s_j$	$l_{ij} = d_i + FS(i,j)^{\min}$
$FS(i,j)^{\max}$	$f_i + FS(i,j)^{\max} \geq s_j$	$\rightarrow s_j + l_{ji} \leq s_i$	$l_{ji} = -d_i - FS(i,j)^{\max}$
$FF(i,j)^{\min}$	$f_i + FF(i,j)^{\min} \leq f_j$	$\rightarrow s_i + l_{ij} \leq s_j$	$l_{ij} = d_i - d_j + FF(i,j)^{\min}$
$FF(i,j)^{\max}$	$f_i + FF(i,j)^{\max} \geq f_j$	$\rightarrow s_j + l_{ji} \leq s_i$	$l_{ji} = d_j - d_i - FF(i,j)^{\max}$

Tabella 2.1: Normalizzazione delle relazioni di precedenza generalizzate.

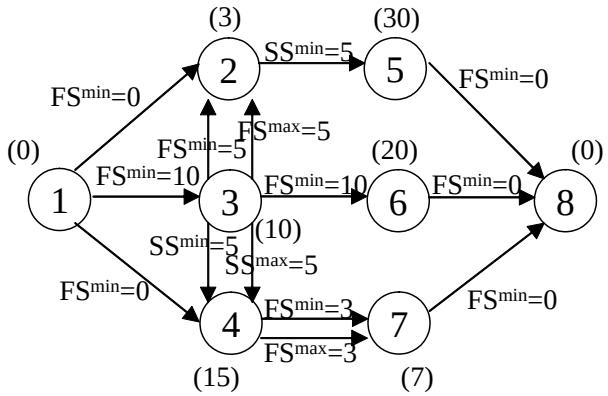

Figura 2.14: AoN con relazioni di precedenza generalizzate.

le informazioni relative alle precedenze generalizzate. Il corrispondente constraint digraph è riportato in Figura 2.15.

□

2.5.1 Schedule delle attività

Nel caso di vincoli di precedenza semplici (§2.2.1), abbiamo visto che il problema di minimizzazione del makespan può essere visto come il duale di un problema di cammino massimo. Questa osservazione consentiva di risolvere il problema in modo molto efficiente, sfruttando il fatto che il grafo del progetto è aciclico. Dobbiamo ora fare i conti col fatto che il constraint digraph può contenere cicli orientati, il che, evidentemente, complica leggermente il problema. Osserviamo anzitutto che, essendoci dei cicli, non si può più parlare di numerazione topologica dei nodi. Dunque, nel seguito adotteremo una numerazione qualsiasi, pur continuando a indicare con 1 e n i nodi di inizio e fine progetto.

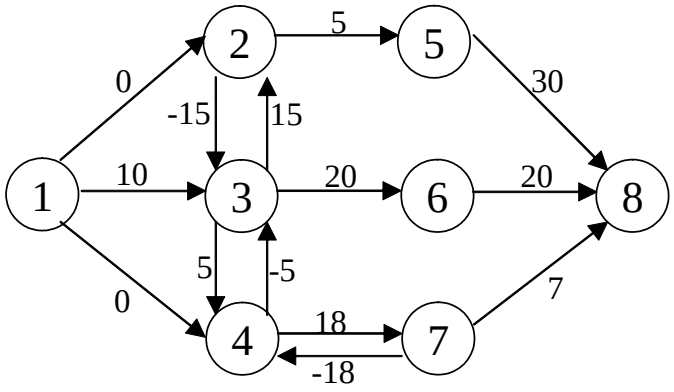

Figura 2.15: Constraint digraph associato al grafo in Figura 2.14.

Richiami sui cammini massimi

Senza dimostrazioni, vogliamo brevemente richiamare alcuni concetti che dovrebbero essere già noti dal corso di Ricerca Operativa (pur essendo stati visti con riferimento a problemi di cammino minimo). Come si è visto nel §2.2.2, determinare il cammino massimo su un grafo aciclico $G = (N, A)$ è un problema molto semplice, risolubile in $O(|A|)$, indipendentemente dal valore dei pesi degli archi. Nel caso in cui nel grafo siano presenti cicli, l'algoritmo precedente non è più utilizzabile.

Supponiamo che nel grafo siano presenti cicli, ma che *tutti gli archi* abbiano pesi negativi. In questo caso, il problema di determinare i cammini massimi da un nodo a tutti gli altri è facilmente risolubile, ad esempio utilizzando l'algoritmo di Dijkstra (il problema equivale infatti a trovare il cammino minimo su un grafo con pesi cambiati di segno). La complessità dell'algoritmo di Dijkstra è superiore a quella dell'algoritmo per grafi aciclici, ma ancora polinomiale ($O(|N|^2 + |A| \log |N|)$).

La generalizzazione successiva riguarda il caso in cui i pesi possono essere sia positivi che negativi, ma non vi sono cicli di peso strettamente positivo. In questo caso l'algoritmo di Dijkstra non si può più applicare. Questo caso è quello che ci interessa, e che vedremo come risolvere nel prossimo paragrafo.

Il caso più generale è quello in cui i pesi sono qualsiasi, ed *esiste* un ciclo di peso positivo, raggiungibile dal nodo 1 (si noti che nel caso dei nostri constraint digraphs, qualunque ciclo è senz'altro raggiungibile dal nodo 1, a causa dell'aggiunta degli archi fittizi $(1, i)$). In questo caso, il problema di cercare un cammino di peso massimo non è ben posto. Infatti, una volta "entrati" nel ciclo positivo, si avrebbe interesse a percorrerlo infinite volte, per massimizzare il peso del cammino. Certamente si può allora riformulare il problema precisando che in questo caso si vuole cercare il ciclo *semplice* (ossia, senza percorrere due volte uno stesso nodo) di peso massimo. Quest'ultimo problema, però, non è affatto semplice, e anzi rappresenta uno dei problemi NP-completi più difficili (è stret-

tamente legato al noto problema del commesso viaggiatore, TSP). Tuttavia, nell'ambito della gestione progetti, gli aspetti risolutivi di questo problema non ci interessano, in quanto come vedremo tra poco per avere significato fisico i nostri constraint digraphs non possono avere cicli positivi.

2.5.2 Algoritmo di Ford-Bellman per il cammino massimo

Quanto visto nel §2.2.2 è ancora sostanzialmente valido, ossia il problema di determinare l'earliest start time EST_i di ogni attività $i \in V$, consiste nel trovare il valore del cammino orientato di costo massimo dal nodo 1 al nodo i. Questo problema può essere risolto se e solo se il grafo non contiene cicli orientati di lunghezza strettamente positiva (in effetti, se vi sono cicli positivi il valore di EST_i non è più definibile), come illustrato nel seguente esempio.

ESEMPIO 11 *Si consideri un progetto costituito da tre attività, a, b, c, con* $d_a = 4$, $d_b = 2$, $d_c = 1$. *Si sa che b deve iniziare almeno 9 giorni dopo che a è iniziata, c almeno dopo 8 giorni che b è finita, e inoltre c deve finire al massimo 16 giorni dopo che a è iniziata. I tre vincoli generalizzati sono dunque dati da* $SS(a,b)^{\min} = 9$, $FS(b,c)^{\min} = 8$, $SF(a,c)^{\max} = 16$. *Nella rappresentazione AoN, si ha il ciclo* $\{(a,b),(b,c),(c,a)\}$, *con pesi* $l_{ab} = 9$, $l_{bc} = 10$, $l_{ca} = -15$, *dunque un ciclo positivo. Andando a scrivere le corrispondenti relazioni, si ottiene:*

$$\begin{aligned} s_a + 9 &\leq s_b \\ s_b + 10 &\leq s_c \\ s_c - 15 &\leq s_a \end{aligned} \tag{2.7}$$

Sommando le tre disequazioni si ottiene $4 \leq 0$, *ovvero un assurdo.* ∎

Quanto visto in questo esempio vale per qualunque ciclo C positivo: sommando i vincoli normalizzati relativi agli archi del ciclo si ottiene sempre la disequazione $l(C) \leq 0$, dove $l(C)$ è appunto la lunghezza del ciclo.

Dunque, se andando a scrivere in forma normalizzata le relazioni di precedenza generalizzate, nel constraint digraph nascono cicli di peso positivo, ci troviamo di fronte a una incongruenza nella definizione del progetto: si sarebbero cioè presi impegni impossibili da rispettare. D'altra parte, in situazioni più complesse di quella illustrata nell'Esempio 11, può non essere semplice accorgersi della presenza di cicli positivi. Per questo motivo, siamo interessati a trovare un algoritmo che risolva efficientemente il problema (2.6) se non vi sono cicli positivi, e che ne trovi invece uno nel caso contrario.

Un algoritmo che presenta queste caratteristiche è l'algoritmo di Ford-Bellman [1958] (Figura 2.16).[2]

[2]Si noti che sarebbe possibile impiegare anche l'algoritmo di Floyd-Warshall. Tuttavia, la complessità dell'algoritmo di Floyd-Warshall è maggiore, anche perché determina i cammini massimi tra *tutte* le coppie di nodi, mentre nell'ambito del constraint digraph siamo interessati soltanto alla determinazione del cammino massimo da 1 a n.

Input: G
Output: s e $prec$
Inizializzazione: $s_1 := 0, s_i := -\infty, prec(i) = 0 \forall i = 2, \ldots, n.$
begin
  Ordina gli archi $\{e_1, \ldots, e_m\}$
  repeat
    for $k = 1 \ldots m$
      if $e_k = (i, j)$ è tale che $s_j < s_i + l_{ij}$
        $s_j = s_i + l_{ij}$
        $prec_j = i$
      end
    end
  until
    (s non si è modificato **or** sono state eseguite n iterazioni)
end

Figura 2.16: Algoritmo di Ford-Bellman.

Schematicamente, l'algoritmo di Ford-Bellman funziona nel seguente modo. Lo scopo, lo ricordiamo, è il calcolo dell'earliest start schedule. Le variabili che vengono via via aggiornate nel corso dell'esecuzione dell'algoritmo rappresentano un vettore di istanti di inizio, *non necessariamente ammissibile*, delle varie attività, che alla fine conterranno l'earliest start time di ciascuna attività.

- Dato il constraint digraph G, inizialmente viene scelto (anche a caso) un ordinamento *degli archi* e definita una soluzione iniziale, non necessariamente ammissibile (ad esempio $s_i := -\infty$).
- A ogni iterazione, tutti gli archi di G vengono visitati, seguendo l'ordine prefissato, e viene effettuata la seguente operazione: se per un arco (i, j) si ha che $s_j < s_i + l_{ij}$ violando il corrispondente vincolo (2.6), si pone $s_j := s_i + l_{ij}$
- Se durante un'iterazione non si verifica alcun aggiornamento, l'algoritmo termina e $EST_i := s_i$ (ossia, il valore s_i corrente corrisponde all'earliest start time di i).

Come si vede in Figura 2.16, l'algoritmo si arresta quando le etichette non vengono più aggiornate per tutta una iterazione del blocco repeat, oppure, comunque, dopo n iterazioni.

Vogliamo ora dimostrare la correttezza dell'algoritmo. A questo scopo, supponiamo inizialmente che nel constraint digraph *non* vi siano cicli positivi. In tal caso, vale la proprietà di *finitezza dei cammini massimi*. Infatti, l'assenza di cicli positivi implica che tutti i cammini massimi sono composti da non più di $n - 1$ archi (un cammino al più attraverserà tutti i nodi una volta sola). Vale allora il seguente:

Teorema 2 (Principio di ottimalità) *Se $P = \{j_1, j_2, \ldots, j_{k-1}, j_k\}$ è un cammino massimo da j_1 a j_k, allora $P = \{j_1, j_2, \ldots, j_{k-1}\}$ è un cammino massimo da j_1 a j_{k-1}.*

Sia ES il vettore dei costi dei cammini massimi. Ricordiamo dalla (2.4), che EST_i è calcolabile se sono noti i valori EST_j di tutti i nodi che precedono i in G. Dunque, come nella (2.4)

$$EST_i = \max_{j \prec i}\{EST_j + l_{ji}\}$$

Vogliamo mostrare che quando l'algoritmo 2.16 termina, il vettore s coincide con ES. Nei lemmi che seguono, S_i è una variabile che viene aggiornata nel corso delle varie iterazioni.

LEMMA 1 *A ogni iterazione, si ha $s_i \leq EST_i$ per ogni i.*

Dimostrazione Per induzione. Per $k = 0$, si ha ovviamente $s_i \leq EST_i, \forall i \in V$. Per assurdo, si supponga che h sia il primo nodo per cui, a un certo punto, $s_h > EST_h$, e sia r l'ultimo nodo per cui si è posto $s_h = s_r + l_{rh}$. Per l'ipotesi induttiva, $s_r \leq EST_r$. Ma allora $s_h = s_r + l_{rh} \leq EST_r + l_{rh} \leq EST_h$ da cui segue la contraddizione. □

LEMMA 2 *Alla fine di ogni iterazione k, per ogni nodo i per cui almeno un cammino di costo massimo da 1 a i è composto da un numero di archi minore o uguale a k, si ha $s_i = EST_i$.*

Dimostrazione Per induzione. Per $k = 0$, si ha che $s_1 = 0$ perché esiste un unico cammino massimo da 1 a 1 (composto ovviamente da zero archi). Sia $P = \{1, j_1, \ldots, j_{k-1}, j_k = i\}$ un cammino massimo fino a i. Poniamo $h = j_{k-1}$. Il cammino $\{1, j_1, \ldots, j_{k-1} = h\}$ è ovviamente (principio di ottimalità) un cammino massimo fino a h composto da $k-1$ archi. Quindi, $EST_i = EST_h + l_{hi}$. Per ipotesi induttiva, alla fine dell'iterazione $k-1$ si ha, $s_h = EST_h$. Per il Lemma 1, $s_i \leq EST_i$. Ma allora quando si processa l'arco (h, i) all'iterazione k sarà $s_i \leq EST_i = EST_h + l_{hi}$ e dopo il processamento sarà $s_i = \max\{s_i, EST_h + l_{hi}\} = EST_h + l_{hi} = EST_i$. □

Fin qui abbiamo supposto che non vi fossero cicli positivi. Ma laddove ci fossero, saremmo in grado di accorgercene?

LEMMA 3 *Nel caso in cui sia presente un ciclo di peso positivo, all' n-esima iterazione l'algoritmo aggiorna almeno un valore s_i.*

Dimostrazione Si consideri un'iterazione k qualsiasi (dunque anche $k > n$), e un ciclo positivo $\{j_1, j_2, \ldots, j_h\}$. Affinché in questa iterazione non vi sia alcun aggiornamento, dev'essere $s_{j_2} \geq s_{j_1} + l_{j_1 j_2}$, $s_{j_3} \geq s_{j_2} + l_{j_2 j_3}$,..., $s_{j_1} \geq s_{j_h} + l_{j_h j_1}$. Sommando queste h relazioni, si ottiene

$$0 \geq l_{j_1 j_2} + l_{j_2 j_3} + \ldots + l_{j_h j_1}$$

che poiché il ciclo ha peso positivo è un assurdo. □

Dunque in presenza di cicli positivi l'algoritmo continuerà a trovare, tra gli archi che costituiscono il ciclo, almeno un arco da aggiornare. Come conseguenza dei precedenti lemmi e del fatto che ogni cammino massimo ha al più $n-1$ archi, si ha che dunque se non vi sono cicli positivi, l'algoritmo termina in al più $n-1$ iterazioni. Altrimenti, iniziando l'iterazione n-esima, si ha la certezza che il grafo contiene un ciclo di peso positivo e quindi si può arrestare l'esecuzione dell'algoritmo.

ESEMPIO 12 *Si supponga di voler applicare l'algoritmo di Ford-Bellman al grafo di Figura 2.15. Per prima cosa si sceglie un ordinamento degli archi, ad esempio (1,2), (1,3) (1,4), (2,3), (2,5), (3,2), (3,4), (3,6), (4,3), (4,7), (5,8), (6,8), (7,4), (7,8). Nella seguente tabella vengono riportati i valori degli s_i al termine di ogni iterazione. Si osservi come al termine della terza iterazione non sia stato modificato nessun valore degli istanti inziali delle operazioni, quindi l'algoritmo si arresta fornendo la soluzione trovata.*

Nodo	*Iter0*	*Iter1*	*Iter2*	*Iter3*
1	*0*	*0*	*0*	*0*
2	$-\infty$	*25*	*25*	*25*
3	$-\infty$	*10*	*10*	*10*
4	$-\infty$	*15*	*15*	*15*
5	$-\infty$	*5*	*30*	*30*
6	$-\infty$	*30*	*30*	*30*
7	$-\infty$	*33*	*33*	*33*
8	$-\infty$	*50*	*60*	*60*

□

Per quanto riguarda infine la complessità computazionale dell'algoritmo di Ford-Bellman, può essere facilmente calcolata come segue. Per quanto discusso, il blocco dentro repeat viene eseguito al più n volte. Ogni sua esecuzione richiede la scansione di tutti gli m archi, e dunque $O(m)$ operazioni. In definitiva risulta dimostrato il seguente teorema.

TEOREMA 3 *Dato un constraint digraph G, l'earliest start schedule può essere calcolato correttamente in $O(mn)$.*

Capitolo 3

La gestione dei costi

Il precedente capitolo ha trattato dei progetti in modo relativamente "statico", ossia in cui le durate delle attività sono fisse e note. Questo ovviamente può essere accettabile in determinate situazioni, e/o può essere una semplificazione utile a ottenere informazioni di massima sul progetto complessivo. Tuttavia, in generale occorre fare i conti col fatto che la durata stessa di un'attività può essere legata ai costi. In particolare, si può pensare di accelerare o rallentare determinate fasi del progetto, se questo è tecnicamente possibile, e se è conveniente dal punto di vista dei costi complessivi. I costi associati a una singola attività (quelli che nel §1.6.4 sono stati indicati come *costi di realizzazione*) derivano fondamentalmente dall'utilizzo di risorse richiesto, che possono essere interne o esterne all'organizzazione che esegue il progetto. In ogni caso, per i modelli che vedremo in questo capitolo le risorse sono ancora illimitate – eventualmente, potranno essere molto costose, ma sempre reperibili. Nel §4 vedremo invece come tener conto del fatto che le risorse disponibili al progetto possono essere limitate.

3.1 Il processo di cost management

I modelli che vediamo in questo capitolo si inseriscono nell'importante processo di gestione dei costi o *cost management*, che si può pensare suddiviso in 4 fasi distinte:

1. Resource planning: determinare quali risorse (personale, attrezzature, materiali) e in che quantità dovranno essere utilizzate.

2. Stima: sviluppare una stima (almeno approssimata) del costo delle risorse necessarie al progetto.

3. Budgeting: allocazione dei costi alle singole attività del progetto. I modelli che vedremo in questo capitolo si inseriscono in questa particolare fase.

4. Controllo: monitoraggio e controllo dei costi nel corso dell'esecuzione del progetto, avendo come riferimento il budget preventivato.

Va detto che in alcuni progetti (specialmente se non di grossa entità), ancorché concettualmente distinti, i primi tre processi possono vedersi come un singolo processo.

La fase di resource planning consiste nella ricognizione accurata delle risorse necessarie allo svolgimento di tutte le attività del progetto. Richiede in ingresso:

- Work Breakdown Structure, che come già detto contiene la descrizione di tutte e sole le attività in cui si dovrà articolare il progetto.
- Dati storici: informazioni riguardanti le risorse utilizzate in circostanze analoghe nel passato.
- Risorse disponibili: descrizione dettagliata delle risorse (umane e non) effettivamente a disposizione del progetto, e in quali periodi di tempo.
- Politiche dell'organizzazione: le politiche dell'organizzazione riguardo all'assunzione di personale e all'acquisto o leasing di risorse varie devono essere note e incluse nella pianificazione.
- Stime delle durate delle attività: le durate e la variabilità delle attività, incluse le eventuali diverse modalità di svolgimento delle stesse.

Gli strumenti utilizzati nel resource planning sono prevalentemente di tipo qualitativo, o al limite richiedono semplici conti. In particolare, in questo caso il giudizio degli esperti (interni o meno all'organizzazione che ha in carico il progetto) è fondamentale. Per il resource planning gli esperti possono anche, in taluni casi, fornire informazioni sulle metodologie per l'attuazione di attività in modalità alternative a quelle previste, e dunque con diverse richieste di risorse. L'output è l'indicazione precisa relativa a quali risorse e in che quantità sono richieste dalle singole attività (foglie della WBS).

Dalla descrizione del fabbisogno di risorse, è necessario passare a una quantificazione dei loro costi (cost estimating), e dunque dei costi delle singole attività (cost budgeting) del progetto. Queste due fasi sono strettamente legate e utilizzano strumenti di tipo analogo. Le informazioni necessarie in ingresso sono:

- Work Breakdown Structure
- Richiesta di risorse: calcolata nella fase precedente.
- Costi unitari: per ogni tipo di risorsa richiesta bisogna conoscere il costo unitario (costo orario del personale, costo a metro cubo di materiale grezzo, ecc.). Se tale costo unitario non è disponibile, va esso stesso stimato.
- Stima delle (possibili) durate delle attività: le stime delle durate devono comprendere tutte le diverse modalità di esecuzione.
- Dati storici: basati su progetti precedenti; da questi si può pensare di ricavare i parametri dei modelli che vedremo.
- Schedule delle attività: l'informazione sulla durata e sulla dislocazione temporale delle attività è fondamentale per l'attribuzione dei costi al periodo esatto nell'orizzonte temporale (e dunque, tra l'altro, per i fini contabili)

- Valutazioni del rischio: a seconda dell'incertezza e/o della criticità di alcuni dati, si possono adottare coefficienti correttivi più o meno "prudenti".

La fase di cost estimating fornisce una stima dei costi delle singole risorse, quella di cost budgeting la imputa alle singole attività. Per effettuare quest'ultima fase è necessario dunque correlare l'utilizzo delle risorse alla durata di un'attività. Ad esempio, supponiamo che la realizzazione fisica del prototipo di un satellite richieda 1 mese se effettuata con una unità di personale, 3 settimane con due unità, 2 settimane con quattro unità. Nei tre casi, il costo della manodopera sarà relativo a 4, 6 oppure 8 settimane di manodopera. Il costo associato dunque a questa attività dipende dal tempo in cui si desidera (o è necessario) realizzarla.

Anche se noi ci concentreremo sul problema dell'allocazione dei costi, va ricordato che l'output del processo di cost estimating comprende anche altri documenti, come ad esempio la descrizione delle metodologie di attribuzione dei costi; l'indicazione sui range di variabilità delle stime; il piano di gestione delle variazioni (descrizione di come affrontare le variazioni potenziali nei costi).

3.2 Trade-off durata/costo

Dato un progetto, ci occuperemo di studiare il trade-off tra durata delle attività e costi di svolgimento delle stesse. Come abbiamo già visto, la durata di una attività dipende anche da quante risorse vengono allocate al suo svolgimento, e quindi in ultima analisi è possibile modificare (diminuire) la durata di una attività aumentandone il suo costo. Dunque, mentre finora abbiamo considerato le durate delle attività come note e immutabili, adesso le considereremo come variabili di decisione. L'obiettivo è quello di correlare le durate delle attività (e dunque lo schedule) con il costo complessivo del progetto.

Tipicamente, nella fase di stima delle durate delle attività, si giunge a determinare una durata "nominale" delle attività. Vogliamo ora affrontare la situazione in cui tale durata può essere ridotta, accettando un aumento dei costi. Tale operazione prende il nome di *crashing*. La necessità di accorciare la durata di un'attività può derivare, tipicamente, dal fatto di dover rispettare dei vincoli di deadline che risulterebbero violati se tutte le attività avessero la durata nominale. Occorre dunque modellare e risolvere il problema di determinare la durata ottimale di ciascuna attività per soddisfare vincoli di deadline relativi all'intero progetto (o fase di esso), con costi di attività che crescono al diminuire delle durate.

Come è intuibile, la complessità di questo problema dipende sostanzialmente dalla forma della funzione che esprime il costo di accorciamento (*crashing cost*). In queste dispense, considereremo i casi di costi lineari, costi convessi e costi concavi. Per semplicità supporremo che le uniche relazioni di precedenza siano di tipo finish-start senza time lag. L'estensione al caso generale non presenta comunque difficoltà concettuali particolari.

3.3 Costi lineari

Iniziamo col considerare il caso più semplice, in cui il costo di una attività varia linearmente con la sua durata. Anzitutto, supponiamo che siano note due durate limite, che

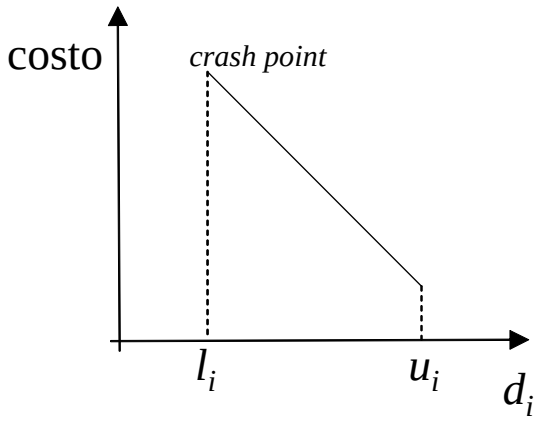

Figura 3.1: Costo di svolgimento dell'attività i (caso lineare).

definiscono l'intervallo di variazione della durata:

1. la durata massima u_i (che sarebbe la durata nominale, spesso detta semplicemente *durata*). Questa è la durata che l'attività ha allocandovi una quantità di risorse standard.

2. la durata minima l_i, detta anche *crash time*. Non è tecnicamente possibile diminuire ulteriormente la durata dell'attività, neanche allocando all'attività infinite risorse.

Il modello lineare suppone che la durata di ciascuna attività i possa assumere tutti i valori compresi tra u_i e l_i, e che i costi varino corrispondentemente (infinite *modalità d'esecuzione*).

La Figura 3.1 mostra il variare del costo della attività i in funzione della sua durata. Nel modello lineare, indichiamo con c_i il costo *marginale* di crashing, ossia quanto costa abbreviare di una unità di tempo (e.g. un giorno) l'attività i.

3.3.1 Singolo cammino critico

Consideriamo inizialmente il problema di determinare la durata del progetto supponendo che tutte le durate delle attività siano pari alla durata massima (o nominale) u_i, e supponiamo che nel grafo AoN esista un *unico* cammino critico.

Se indichiamo con T la durata (minima) del progetto, da quanto visto nel §2.2.2 è evidente che per ridurre il tempo di completamento del progetto, occorre diminuire la durata di qualche attività critica. Infatti, una diminuzione di 1 unità di tempo di una qualsiasi attività critica comporta una identica diminuzione nella durata dell'intero progetto. Chiaramente, tra tutte le attività sul cammino critico, converrà selezionare ed abbreviare la durata dell'attività con costo marginale minore. Si osservi che la diminuzione di una attività non critica non porta ad alcuna diminuzione della durata del progetto, mentre contribuirebbe all'aumento dei costi.

Input:
Output:
begin
  repeat
    Seleziona l'attività sul cammino critico che:
      — ha costo unitario minore
      — non ha ancora raggiunto il suo crash time
  until
    non è più possibile ridurre il tempo di alcuna attività critica
    or vengono individuati 2 o più cammini critici
end

Figura 3.2: Algoritmo di crashing

Dunque, inizialmente la durata complessiva di un progetto può essere diminuita, aumentando comunque il costo di esecuzione del progetto, di una unità alla volta, agendo sull'attività critica i di costo minimo. Si può andare avanti così fino a che non si verifica uno di questi due eventi: l'attività i ha raggiunto la sua durata minima l_i, oppure si ha che il cammino critico non è più unico (Figura 3.2).

Se si verifica il primo caso, semplicemente saremo costretti a scegliere un'altra delle attività del cammino critico, avente un costo marginale in generale maggiore. Invece, nel secondo caso il problema si complica, in quanto continuare a diminuire la sola attività i in generale non porterebbe più alcuna diminuzione nella durata del progetto, e farebbe soltanto aumentare i costi.

Esempio 13 *A partire dal progetto dell'Esempio 1 e rappresentato come AoN, e con le informazioni aggiuntive riportate nella seguente tabella*

j	*prec*	*durata*	*durata minima*	*costo marginale*
1	-	*6*	*4*	*70*
2	-	*2*	*1*	*50*
3	*1*	*3*	*3*	
4	*2*	*2*	*2*	
5	*3*	*4*	*2*	*40*
6	*4*	*1*	*1*	
7	*5,6*	*1*	*1*	
8	*7*	*6*	*3*	*20*
9	*8*	*3*	*2*	*40*
10	*8*	*1*	*1*	
11	*9,10*	*1*	*1*	

si ha che il tempo di completamento del progetto sarà pari a 24 settimane ed il suo costo sarà pari al costo iniziale del progetto. La figura 3.3 evidenzia il cammino critico

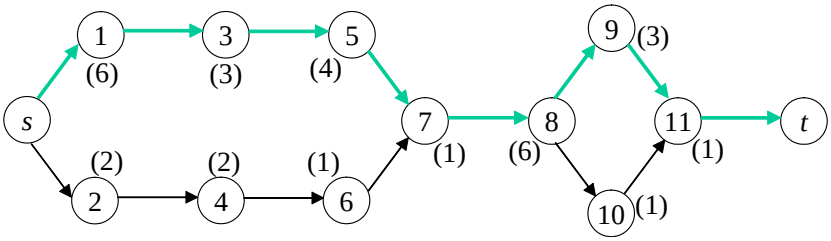

Figura 3.3: Cammino critico del progetto (iniziale).

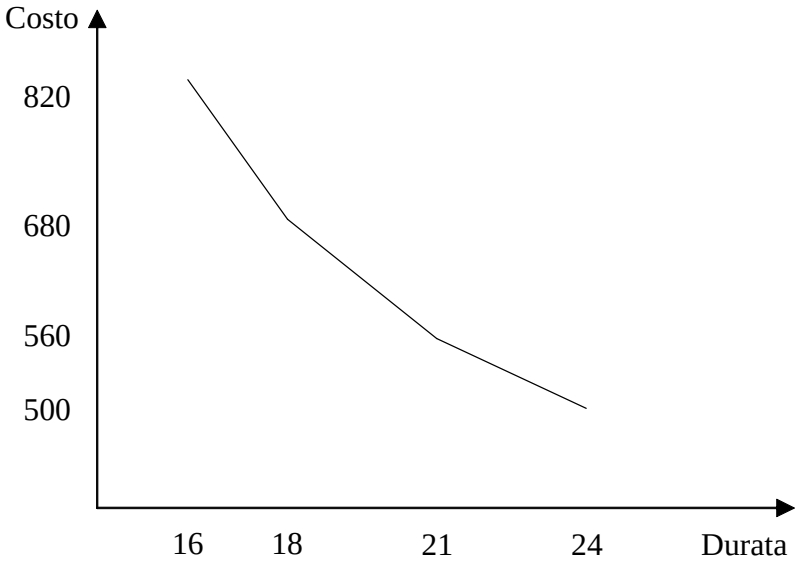

Figura 3.4: Trade-off durata/costo progetto.

del progetto in esame. Volendo diminuire il tempo di completamento, conviene diminuire la durata della attività 8 che ha costo marginale minimo (20). L'attività 8 può essere diminuita di 3 settimane, il che porta il progetto ad avere durata 21, con un incremento di costo di 60. A questo punto, non potendo più diminuire la durata dell'attività 8, selezioniamo l'attività 9 (costo marginale 40). L'attività 9 può diminuire solo di 1 unità (la durata del progetto scende a 20 settimane, il costo aggiuntivo sale a 100). L'attività avente minimo costo marginale (e che non ha raggiunto il proprio crash point) è l'attività 5, che può essere diminuita di 2 settimane (durata progetto 18, costo aggiuntivo 180). Infine, possiamo diminuire di 2 settimane la durata dell'attività 1 (durata progetto 16, incremento di costo del progetto 320). Il progetto non potrà durare meno di 16 settimane, in quanto non ci sono altre attività da diminuire sul cammino critico. L'andamento del trade-off durata/costo è mostrato in Figura 3.4 dove si evidenzia il caratteristico andamento convesso della curva di trade-off. □

3.3.2 Cammini critici multipli

Il semplice algoritmo di crashing appena visto non è applicabile al caso, più generale, in cui si presentino due o più cammini critici: come già osservato, ridurre la durata di una attività che si trova su un solo cammino critico porta a ridurre la lunghezza di quel solo cammino, lasciando invariati gli altri. Quindi, per ridurre la durata complessiva del progetto, occorre individuare un insieme di costo minimo di attività in grado di "intercettare" tutti i cammini critici.

E' possibile allora formulare il problema di determinare le durate ottime delle singole attività come un problema di programmazione lineare, in cui si cerca di minimizzare i costi di crashing, con il vincolo che il progetto termini entro una deadline T.

Sia $G = (V, A)$ la rete AoN con precedenze di tipo finish-start senza time-lag, c_i il costo marginale di riduzione della durata dell'attività $i \in V$, l_i e u_i le durate minime e massime dell'attività $i \in V$. Indichiamo con y_i la variabile di decisione rappresentante la *riduzione* di durata dell'attività $i \in V$ rispetto alla durata nominale u_i. La durata effettiva dell'attività i sarà dunque $u_i - y_i$. Inoltre, come già in §2.2.1, s_i è la variabile che indica l'istante d'inizio dell'attività $i \in V$.

I vincoli del problema sono:

$$\min z = \sum_{i=1}^{n} c_i y_i \tag{3.1}$$

$$s_j - s_i \geq u_i - y_i \quad \forall (i,j) \in A \tag{3.2}$$

$$y_i \leq u_i - l_i \quad \forall i \in V \tag{3.3}$$

$$s_n \leq T \tag{3.4}$$

$$s_i, y_i \geq 0 \quad \forall i \in V$$

La funzione obiettivo (3.1) da minimizzare è semplicemente la somma dei crashing cost. I vincoli (3.2) hanno lo stesso significato dei vincoli (2.1) e modellano i vincoli di precedenza. Infine, il vincolo (3.4) indica che il progetto deve terminare entro T (ove n è, al solito, l'attività fittizia di fine progetto).

Esempio 14 *Per il progetto dell'Esempio 13, la formulazione matematica del problema di crashing con costi lineari è la seguente (con una deadline di 20 settimane):*

$$
\begin{aligned}
\min \quad & 70y_1 + 50y_2 + 40y_5 + 20y_8 + 40y_9 && (3.5)\\
s_t &\leq 20\\
s_1 &\geq s_s\\
s_2 &\geq s_s\\
s_3 &\geq s_1 + 6 - y_1\\
s_4 &\geq s_2 + 2 - y_2\\
s_5 &\geq s_3 + 3\\
s_6 &\geq s_4 + 2\\
s_7 &\geq s_5 + 4 - y_5\\
s_7 &\geq s_6 + 1\\
s_8 &\geq s_7 + 1\\
s_9 &\geq s_8 + 6 - y_8\\
s_{10} &\geq s_8 + 6 - y_8\\
s_{11} &\geq s_9 + 3 - y_9\\
s_{11} &\geq s_{10} + 1\\
y_1 &\leq 2\\
y_2 &\leq 1\\
y_5 &\leq 2\\
y_8 &\leq 3\\
y_9 &\leq 1\\
y_i, s_i &\geq 0
\end{aligned}
$$

□

Esempio 15 *Si consideri il progetto dell'Esempio 1, rappresentato come AoN, e le informazioni aggiuntive riportate nella Tabella 3.1. Con le durate nominali, il makespan del progetto è pari a 24 settimane ed il suo costo è pari al costo iniziale del progetto. Volendo diminuire il tempo di completamento, conviene diminuire la durata della attività che ha costo marginale minimo (attività 8, costo marginale 20). L'attività 8 può essere ridotta di 3 settimane, il progetto avrà durata 21 ed un incremento di costo di 60. A questo punto, non potendo più diminuire la durata della attività 8, si seleziona l'attività 9 (costo marginale 40). L'attività 9 può diminuire solo di 1 settimana (durata progetto 20, costo aggiuntivo 100). L'attività che minimizza il costo marginale (e che non ha raggiunto il crash point) è l'attività 5, che può essere diminuita di due unità (durata progetto 18, costo aggiuntivo 180). Ora l'attività da diminuire di 5 settimane è l'attività 1 (durata progetto 13, incremento di costo del progetto 530). Fin qui, nonostante la diminuzione della durata del progetto, il cammino critico è sempre rimasto unico,* $(0, 1, 3, 5, 7, 8, 9, 11, 12)$. *A questo*

j	*prec*	durata	durata minima	costo marginale
1	-	6	1	70
2	-	2	1	50
3	1	3	1	80
4	2	2	1	90
5	3	4	2	40
6	4	1	1	-
7	5,6	1	1	-
8	7	6	3	20
9	8	3	2	40
10	8	1	1	-
11	9,10	1	1	-

Tabella 3.1: Dati dell'Esempio 15.

punto viene selezionata l'attività 3 (costo marginale 80) che può essere ridotta di due settimane. Si osservi però che dopo la riduzione della prima settimana (attività 3, durata 2) il cammino critico non è più unico, in quanto diviene critico anche $(0, 2, 4, 6, 7, 8, 9, 11, 12)$*: una ulteriore riduzione della durata della sola attività 3 non può portare alcuna diminuzione nella durata del progetto, che a questo punto dura 12 settimane, con un costo aggiuntivo pari a 610. Occorre quindi individuare un insieme di attività che portino ad una riduzione della lunghezza complessiva di* entrambi *i cammini critici. Poiché le durate delle attività 7, 8, 9 e 11, comuni ai due cammini critici, sono già al loro valore minimo, sarà necessario scegliere due attività, appartenenti alle due porzioni di cammino critico, ovvero (1,3,5) e (2,4,6). Nel cammino critico (1,3,5) l'unica attività che ancora non ha raggiunto la durata minima è l'attività 3, che può essere diminuita ancora di una settimana. Nel secondo cammino critico (2,4,6) posso scegliere di diminuire sia l'attività 2, sia l'attività 4. Decidiamo di diminuire la durata dell'attività 2 in quanto ha costo marginale minore (50). Quindi la riduzione simultanea dell'attività 2 e 3 porta alla riduzione di una settimana della durata del progetto ad un costo aggiuntivo di 50+80. A questo punto la durata del progetto è di 11 settimane, con un costo di 740. Non è possibile diminuire ulteriormente la durata del progetto, in quanto vi è un cammino critico (ossia* $(0, 1, 3, 5, 7, 8, 9, 11, 12)$*) in cui tutte le attività hanno raggiunto il crash point. La figura 3.5 evidenzia i cammini critici al termine del crashing. Il trade-off durata/costo, dal caratteristico andamento convesso, è mostrato in Figura 3.6.*

□

Va osservato che in alcuni casi l'andamento della curva di trade off tempi/costi può essere meno immediato da ricavare. Si consideri infatti il seguente esempio.

Esempio 16 *Si consideri il progetto in figura 3.7, in cui i numeri a fianco di ciascuna attività indicano la durata nominale. Si supponga inoltre che il costo marginale di riduzione sia pari a 100 per le attività 2 e 8, pari a 50 per l'attività 5, mentre per le altre attività tale costo sia superiore. Si noti che la durata nominale del progetto è pari a 46, e il*

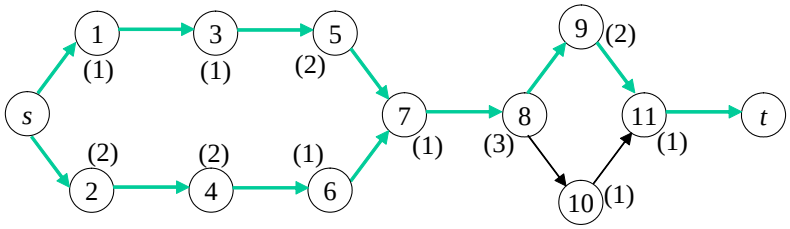

Figura 3.5: Cammini critici del progetto (finale).

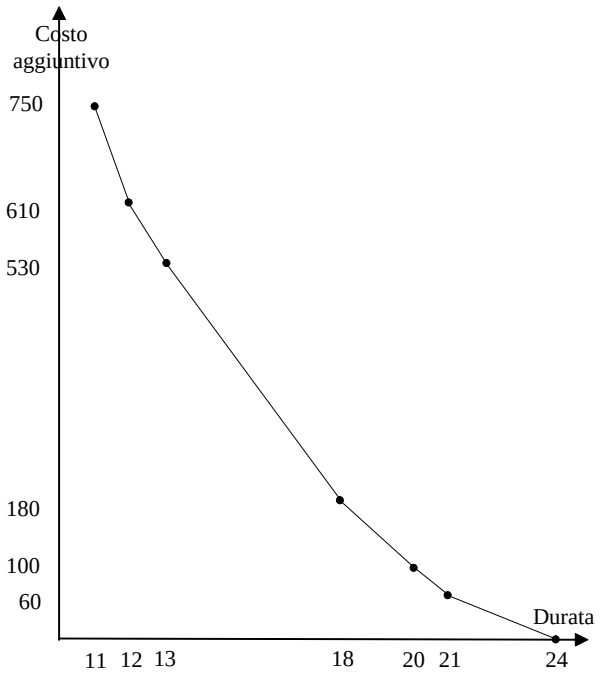

Figura 3.6: Trade-off durata/costo progetto.

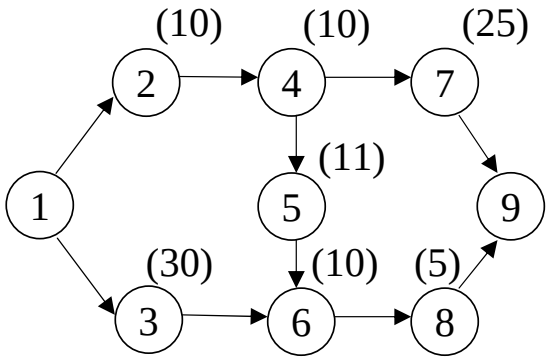

Figura 3.7: Rete AoN per l'Esempio 16.

percorso critico è $(1, 2, 4, 5, 6, 8, 9)$*. Se si vuole diminuire la durata del progetto al valore 45, conviene ovviamente ridurre di una unità di tempo l'attività 5, di costo 50. A questo punto, tutti e tre i cammini da 1 a 9 sono critici. Volendo ridurre la durata del progetto di un'altra unità, osserviamo che è possibile effettuare tale riduzione al costo di 150. Infatti, riducendo di una unità le attività 2 e 8, la lunghezza dei cammini* $(1, 2, 4, 7, 9)$ *e* $(1, 3, 6, 8, 9)$ *diviene 44, mentre quella del cammino* $(1, 2, 4, 5, 6, 8, 9)$ *subisce una duplice riduzione, portandosi a 43. Ma allora, è possibile riportare la durata dell'attività 5 al valore 11, risparmiando così 50.* □

Per avere informazioni complete relativamente al trade-off durata/costo, è necessario conoscere il valore ottimo z^* nel problema (3.1) per ogni valore della deadline T. Si noti peraltro che un lower bound T_L e un upper bound T_U per T sono facilmente ottenibili calcolando la lunghezza del percorso critico su G utilizzando, rispettivamente, le durate minime e massime di ciascuna attività.

In linea di principio, si può dunque risolvere una sequenza di problemi per tutti i valori di T in $[T_L, T_U]$, e dalle diverse soluzioni si costruisce la curva dei trade-off durata/costo. Anche se in teoria esistono, come si diceva, infinite modalità di esecuzione, va però osservato che all'ottimo, se tutti i valori u_i (come supponiamo) sono interi, la soluzione (s^*, y^*) sarà sicuramente intera. Si può infatti mostrare che la matrice dei coefficienti è totalmente unimodulare[1]. Dunque, sarebbe sufficiente risolvere il problema solo per valori interi del parametro T.

Facciamo alcune considerazioni più di dettaglio sull'andamento della curva. Come si può osservare dalla Figura 3.6, si tratta di una curva lineare a tratti, convessa. Questo andamento non deve sorprendere, in quanto costituisce un caso particolare della curva che esprime la sensibilità del valore ottimo della funzione obiettivo di un problema di PL, al variare di uno dei termini noti (risorse). Qui il parametro che varia è il tempo a disposizione T per terminare il progetto, che a tutti gli effetti è una risorsa. Man mano

[1]Vale la pena notare come in questo caso la proprietà di unimodularità rivesta un ruolo diverso rispetto al problema (2.3): mentre in quel caso è indispensabile per poter riconoscere che il problema è un problema di cammino massimo, in questo caso la soluzione fornita continuerebbe ad avere significato fisico, anche se fosse frazionaria.

che la disponibilità di questa risorsa diminuisce, il vincolo (nel nostro caso, $s_n \leq T$) risulta più stringente, e chiaramente la funzione obiettivo peggiora. La pendenza della curva di trade-off è data, com'è noto, dal valore della variabile duale associata al vincolo di deadline (prezzo-ombra). Dunque tale variabile esprime, in ciascun punto della curva, quanto costa voler terminare il progetto in una unità di tempo (e.g. settimana) in meno. Nel nostro caso, il suo valore è pari alla somma dei costi marginali delle attività critiche che è necessario abbreviare.

In definitiva quindi, per ottenere l'intera curva di trade-off, non è necessario risolvere un problema di PL per ogni valore di T, ma solo uno per ogni intervallo di valori di T all'interno dei quali la base ottima rimane la stessa. Infatti, l'analisi della sensibilità fornisce i valori estremi di ciascun ciascun intervallo, all'interno del quale la pendenza della curva di trade-off (prezzo-ombra della risorsa tempo) rimane costante.

3.3.3 Vincoli di precedenza generalizzati

L'introduzione di precedenze generalizzate non presenta difficoltà concettuali, dal punto di vista del problema (3.1), in quanto porta solo a una modifica nei vincoli (3.2). Ad esempio, una relazione start-finish di tipo max tra le attività i e j si può esprimere come

$$s_i + SF(i,j)^{\max} \geq f_j = s_j + u_j - y_j$$

Le cose sono leggermente diverse per quanto riguarda il calcolo della curva di trade-off. Infatti, nel caso precedente per individuare il range $[T_L, T_U]$ di valori possibili per T bastava risolvere due problemi di cammino massimo, utilizzando rispettivamente i tempi minimi l_i e massimi u_i di ciascuna attività. Stavolta, per via delle relazioni di precedenza generalizzate, e dunque della possibile presenza di cicli positivi nel constraint digraph, non possiamo adottare la stessa tecnica, in quanto può accadere che ponendo tutte le durate al valore minimo o massimo non esista soluzione ammissibile, mentre una soluzione può esistere per valori di durata intermedi. D'altro canto, se esiste una soluzione ammissibile per un valore $\bar{T}$, allora esiste anche per tutti i $T > \bar{T}$ (al limite, la stessa soluzione). Dunque, si può operare in questo modo. Inizialmente, si elimina il vincolo (3.4), in modo da trovare la soluzione di costo minimo senza vincoli di deadline. Se una soluzione esiste, ciò indica che il problema era ben posto, e il valore di makespan T_U così ottenuto è anche il più grande valore di deadline che si può avere interesse a considerare. È possibile a questo punto, effettuando l'analisi di sensibilità, scoprire per quale valore minimo di T tale soluzione è ancora ottima; dopo di che si ricostruisce la curva a ritroso fino al valore minimo di T, sotto il quale non vi è soluzione ammissibile, come nel caso precedente.

3.4 Costi convessi

Passiamo ora a considerare il caso, più generale, in cui il crashing cost $c_i(y_i)$ dell'attività i, in funzione dell'accorciamento y_i, è una funzione convessa. La convessità dei crashing cost rappresenta il fatto, verificato in molte situazioni, che è necessario uno sforzo incrementale crescente per ridurre ulteriormente la durata di una certa attività. È facile mostrare che,

se i costi sono convessi per ogni attività, anche la funzione di costo complessiva, pari alla somma dei costi delle singole attività, sarà convessa.

Procedendo in modo analogo a quanto fatto nel §3.3.2, possiamo formulare il problema di trovare le durate ottime delle singole attività (*min cost crash time*):

$$\begin{aligned} \min & \, c(y) \\ s_j - s_i &\geq d_i = u_i - y_i \quad \forall ij \in A \\ y_i &\leq u_i - l_i \quad \forall i \in V \\ y_i, s_i &\geq 0 \quad \forall i \in V \end{aligned} \tag{3.6}$$

dove $c(y) = c_1(y_1) + c_2(y_2) + \ldots + c_n(y_n)$. Si noti che l'insieme ammissibile è lo stesso del problema (3.1), e dunque in particolare è ancora un politopo; l'unica cosa che cambia è la funzione obiettivo. Per semplicità nel seguito ci limiteremo al caso in cui le relazioni di precedenza siano finish-start senza time lag; l'estensione al caso generale potrebbe comunque essere svolta esattamente negli stessi termini visti nel §3.3.2.

Nella letteratura scientifica esistono metodi efficienti per la minimizzazione di funzioni convesse su politopi. D'altro canto, un approccio semplificato, ma comunque in grado di fornire soluzioni più che accettabili da un punto di vista pratico, è quello in cui si utilizza, anziché la funzione obiettivo originaria, una sua approssimazione lineare a tratti. È peraltro frequente la situazione in cui i crashing cost sono essi stessi lineari a tratti, ossia quando l'informazione è del tipo: il costo *marginale* per diminuire la durata è pari a 10 per le prime x settimane, 20 dalla $x+1$ alla y, 25 per ogni ulteriore settimana.

Vediamo ora che, con questa approssimazione, il problema può essere risolto molto efficientemente. La curva dei crashing cost dell'attività i è dunque espressa da $k(i)$ tratti di retta (Figura 3.8). Avremo associati $k(i)$ costi marginali $c_{i1}, \ldots, c_{i,k(i)}$, con $c_{i1} < \ldots < c_{i,k(i)}$.

Introduciamo allora una formulazione per questo problema. Chiamiamo $u_{i0}, u_{i1}, \ldots, u_{i,k(i)}$ le ascisse degli estremi di ciascun tratto della approssimazione lineare a tratti ($u_{i0} > u_{i1} > \ldots, > u_{i,k(i)}$), e per ogni i introduciamo le variabili $y_{i1}, \ldots, y_{i,k(i)}$. La variabile y_{it} rappresenta di quante settimane è ridotta l'attività i nell'intervallo $[u_{it}, u_{i,t-1}]$ al costo marginale di c_{it}. La formulazione sarà, quindi:

$$\begin{aligned} \min & \sum_{i \in V} \sum_{t=1}^{k(i)} c_{it} y_{it} \\ s_j - s_i &\geq u_i - y_i \quad \forall (i,j) \in E \\ s_n &\leq T \\ y_i &= y_{i1} + \ldots + y_{i,k(i)} \quad \forall i \in V \\ y_{iq} &\leq u_{i,q} - u_{i,q+1} \quad \forall i \in V, q = 1, \ldots k(i) \\ y, s &\geq 0 \end{aligned} \tag{3.7}$$

Apparentemente, non tutte le soluzioni ammissibili del problema (3.7) corrispondono a soluzioni fisicamente realizzabili, in quanto i vincoli non proibiscono che $y_{iq} < u_{i,q-1} - u_{iq}$ e $y_{i,q+1} > 0$, il che va contro la definizione che abbiamo dato dei crashing cost: infatti, per

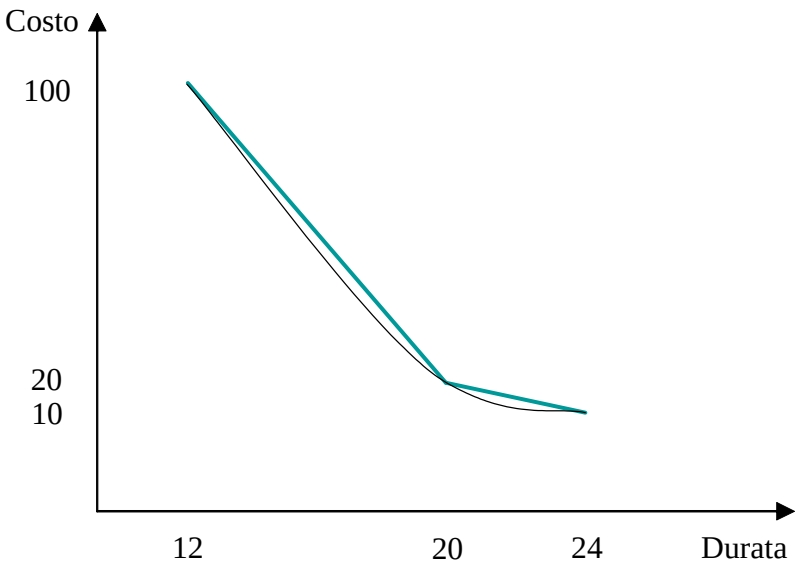

Figura 3.8: Costi convessi approssimati da una curva lineare a tratti.

u	durata	costo
u_0	24	10
u_1	20	20
u_2	12	100

poter pagare $c_{i,q+1}$ una settimana di diminuzione, occorre avere già accorciato la durata di $u_{i,q-1} - u_{iq}$ settimane al costo marginale c_{iq}. Tuttavia, la validità della formulazione (3.7) sta nel fatto che, seppure molte soluzioni ammissibili sono irrealizzabili, tutte le soluzioni ottime sono realizzabili.

TEOREMA 4 *Sia* (s^*, y^*) *ottima. Allora* $y^*_{iq} < u_{i,q-1} - u_{iq}$ *implica* $y^*_{i,q+1} = 0$.

Dimostrazione Per assurdo sia $y^*_{i,q+1} > 0$ e $y^*_{iq} < u_{i,q-1} - u_{iq}$. Poniamo $\epsilon = \min\{y^*_{i,q+1}, u_{i,q-1} - u_{iq} - y^*_{iq}\}$. Si ha $\epsilon > 0$; inoltre possiamo costruire una nuova soluzione ammissibile y' ponendo $y'_{iq} = y^*_{iq} + \epsilon$ e $y'_{i,q+1} = y^*_{i,q+1} - \epsilon$, lasciando tutte le altre componenti inalterate. La variazione di costo della nuova soluzione rispetto alla precedente è $c_{iq}y'_{iq} + c_{i,q+1}y'_{i,q+1} - c_{iq}y^*_{iq} - c_{i,q+1}y^*_{i,q+1} = c_{iq}(y'_{iq} - y^*_{iq}) + c_{i,q+1}(y'_{i,q+1} - y^*_{i,q+1})$ e cioè pari a $\epsilon(c_{iq} - c_{i,q+1}) < 0$. Quindi, la nuova soluzione costa meno della precedente, che per ipotesi era ottima, il che è una contraddizione. ∎

3.5 Costi concavi

La situazione in cui i crashing cost sono concavi corrisponde al caso in cui occorre uno sforzo via via minore per ottenere diminuzioni nella durata delle attività. Ancorché meno frequenti rispetto al caso convesso, situazioni di questo tipo capitano. Ad esempio, supponiamo che due operai effettuino una complessa operazione di montaggio, che richiede di effettuare fori, e per ogni foro l'inserimento di un componente. Un operaio da solo impiegherebbe, supponiamo, 5 giorni, e costerebbe dunque 5 giorni/persona. Se vogliamo impiegarci 4 giorni, occorre ingaggiare un secondo operaio, e avremmo in totale 8 giorni/persona. Un terzo operaio potrebbe invece operare in parallelo agli altri due e effettuare tutti gli inserimenti, e l'intera operazione potrebbe aver termine in 3 giorni: dunque, 9 giorni/persona.

Indichiamo allora ancora con $c_i(y_i)$ il crashing cost dell'operazione i in funzione della riduzione y_i. Il costo complessivo è pari alla somma di funzioni concave $c(y) = \sum c_i(y_i)$. Formalmente, il problema è ancora il seguente.

$$\begin{aligned} \min\, & c(y) & & & (3.8)\\ s_j - s_i &\geq d_i = u_i - y_i \quad \forall ij \in A\\ y_i &\leq u_i - l_i \quad \forall i \in V\\ s_n &\leq T\\ y_i, s_i &\geq 0 \quad \forall i \in V \end{aligned}$$

Stavolta si tratta di minimizzare una funzione concava su un politopo. Benché sia noto che l'ottimo giace su un vertice del politopo, non è immediato trovarlo, in generale.

Un primo approccio che si può adottare è analogo a quello visto per le funzioni convesse: ossia, approssimiamo il crashing cost di ciascuna attività con una spezzata dai costi marginali $c_{i1}, \ldots, c_{ik}$, ove stavolta $c_{i1} > \ldots > c_{ik}$ (Figura 3.9). Indichiamo inoltre con T_{iq} l'ampiezza del q-esimo intervallo della curva lineare a tratti, ossia $T_{iq} = u_{i,q-1} - u_{iq}$.

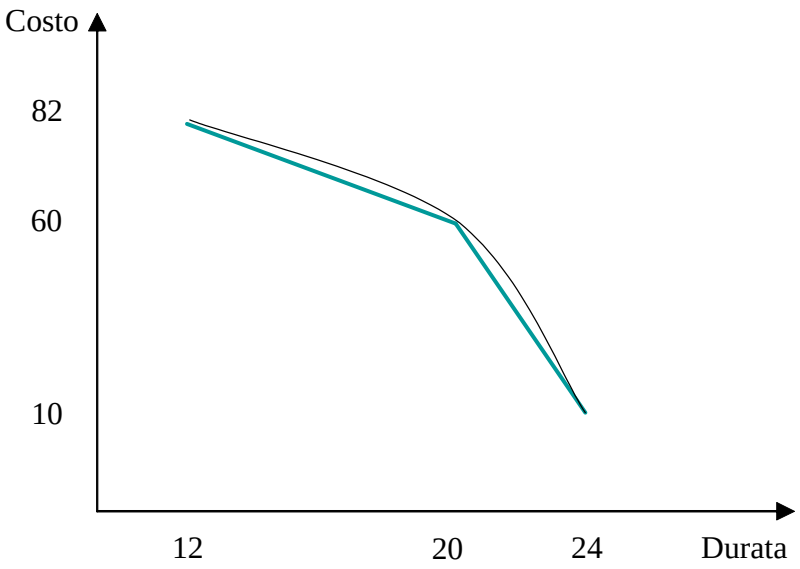

Figura 3.9: Costi concavi approssimati come costi lineari a tratti.

u	durata	costo
u_0	24	10
u_1	20	60
u_2	12	82

Introduciamo le variabili $y_{i1}, \ldots, y_{i,k(i)}$, per ogni $i \in V$. La funzione obiettivo ed i vincoli, come nel caso di costi convessi, rimangono:

$$\begin{aligned} \min \sum_{i \in V} \sum_{t=1}^{k(i)} c_{it} y_{it} & & (3.9) \\ s_j - s_i &\geq u_i - y_i \quad \forall (i,j) \in A \\ s_n &\leq T \\ y_i &= y_{i1} + \ldots + y_{i,k(i)} \quad \forall i \in V \\ y_{iq} &\leq u_{i,q-1} - u_{iq} \quad \forall i \in V, q = 1, \ldots, k(i) \\ y, s &\geq 0 \end{aligned}$$

Ma questa formulazione non è ancora sufficiente per rappresentare correttamente il problema. Infatti, nel caso di funzioni di costo concave, non vale il Teorema 4, e dunque non possiamo sperare di risolvere il problema approssimato con un problema di PL. Dobbiamo stavolta imporre esplicitamente che $y_{i,q+1} = 0$ se il valore di y_{iq} è strettamente inferiore a T_{iq}. A questo scopo, introduciamo delle variabili binarie x_{iq} tali che $x_{iq} = 0$ se $y_{iq} < T_{iq}$, mentre $x_{iq} = 1$ se $y_{i,q+1} > 0$. Il significato è che se $x_{iq} = 0$, non si è ancora saturato il tratto q-esimo della curva, e dunque non si può usare quello $q+1$-esimo (ossia, deve essere $y_{i,q+1} = 0$). Occorre quindi aggiungere due tipi di vincoli. Il primo vincolo garantisce che se $y_{iq} < T_{iq}$, $x_{iq} = 0$. Per ogni $i \in V$ e $q = 1, \ldots, k(i)$ avremo:

$$T_{iq} x_{iq} \leq y_{iq}$$

Il secondo tipo di vincoli garantisce che se $x_{iq} = 0$, allora $y_{i,q+1} = 0$. Dunque, per ogni $i \in V$ e $q = 1, \ldots, k(i) - 1$:

$$T_{i,q+1} x_{iq} \geq y_{i,q+1}$$

In definitiva si ottiene la seguente formulazione del problema con costi concavi, lineari a tratti, come problema di programmazione lineare mista intera:

$$\begin{aligned} \min \sum_{i \in V} \sum_{t=1,\ldots,k(i)} c_{it} y_{it} & & (3.10) \\ s_j - s_i &\geq u_i - y_i \quad \forall (i,j) \in A \\ s_n &\leq T \\ y_i &= y_{i1} + \ldots + y_{i,k(i)} \quad \forall i \in V \\ y_{iq} &\leq u_{i,q-1} - u_{iq} \quad \forall i \in V, q = 1, \ldots k(i) \\ T_{iq} x_{iq} &\leq y_{iq} \quad i \in V, q = 1, \ldots, k(i) \\ T_{i,q+1} x_{iq} &\geq y_{i,q+1} \quad i \in V, q = 1, \ldots, k(i) - 1 \\ y, s &\geq 0 \\ x &= \{0, 1\} \end{aligned}$$

Se l'istanza del problema (3.10) è di dimensioni contenute, si può utilizzare un solutore commerciale general purpose. Altrimenti, occorre sviluppare un algoritmo di enumerazione implicita. Tuttavia, in questo caso c'è da considerare se non convenga sviluppare

un approccio che risolva direttamente il problema concavo (3.8), anziché la sua approssimazione lineare a tratti. Quest'ultima è la strada che seguiremo. Per fare questo, è necessario richiamare alcuni concetti generali sul branch and bound.

Richiami sul branch and bound

Si consideri un generico problema di ottimizzazione (P):

$$\begin{aligned} z = \min\, & f(x) \\ & x \in S \end{aligned} \tag{3.11}$$

ove $S \subseteq R^n$ è l'insieme ammissibile (che supporremo non vuoto). Il problema consiste nell'individuare una soluzione ottima $x^* \in S$ e il suo valore $f(x^*)$ (che supporremo finito).

L'idea di fondo del branch and bound è tentare di risolvere il problema di ottimizzazione decomponendolo ricorsivamente (branching) in sottoproblemi più piccoli, fino ad arrivare a sottoproblemi di dimensioni tali per cui è possibile applicare con successo qualche metodo *efficiente* di soluzione.

Il branching consiste dunque nella decomposizione dell'insieme S in una famiglia di sottoinsiemi $(S_1 \subset S, \ldots, S_r \subset S)$ (sottoproblemi) tali che $\bigcup_{i=1}^{r} S_i = S$. Meglio ancora se i sottoproblemi partizionano esattamente lo spazio delle soluzioni ammissibili, $S_i \cap S_j = \emptyset, 1 \leq i < j \leq r$. Quindi, si risolvono gli r problemi di ottimizzazione associati:

$$\begin{aligned} z_1 = \min\, & f(x) \\ & x \in S_1 \end{aligned} \tag{3.12}$$

e così via fino al r-esimo problema

$$\begin{aligned} z_r = \min\, & f(x) \\ & x \in S_r \end{aligned} \tag{3.13}$$

Considerando il singolo i-esimo sottoproblema, sia z_i la sua soluzione ottima. Necessariamente, sarà $z_i \geq z^*$, in quanto ci stiamo limitando a un sottoinsieme S_i dell'intera regione ammissibile, e il valore della soluzione ottima per il problema originale sarà il minimo assunto dal valore delle soluzioni ottime dei vari sottoproblemi, ovvero $z^* = \min\{z_1, \ldots z_r\}$. Ogni sottoproblema può essere a sua volta decomposto in altri sottoproblemi di dimensioni minori, e la decomposizione prosegue fino a che i singoli sottoproblemi vengono risolti per via di un metodo diretto che funziona quando il problema è sufficientemente *piccolo*. La struttura che rappresenta il processo di decomposizione in sottoproblemi prende il nome di *albero di enumerazione*.

Il metodo del branch and bound ruota attorno al concetto di *rilassamento* di un problema di ottimizzazione. Dato un problema di ottimizzazione (P) come (3.11), un suo rilassamento è un problema $r(P)$:

$$\begin{aligned} z' = \min\, & g(x) \\ & x \in T \end{aligned} \tag{3.14}$$

definito in modo tale che tra le soluzioni ottime dei due problemi valga $z' \leq z$. Affinché si abbia questa relazione, in molti casi si ha che la funzione obiettivo è la stessa nei due problemi, ma $S \subseteq T$: cercando l'ottimo in un insieme più ampio, chiaramente possono trovarsi soluzioni migliori. Nell'applicazione che vedremo nel prossimo paragrafo, invece, avremo che $S = T$ mentre $f \neq g$.

Tra un problema e un suo rilassamento intercorre la relazione espressa nel seguente lemma.

LEMMA 4 *Sia x^* la soluzione ottima di $r(P)$. Se x^* è ammissibile per (P) e inoltre $f(x^*) = g(x^*)$, allora x^* è ottima anche per (P).*

Dimostrazione Poiché $r(P)$ è un rilassamento, si ha $f(x^*) = g(x^*) = z' \leq z$. Inoltre, poiché $x^* \in S$, si ha pure $z \leq f(x^*)$, da cui $z = f(x^*)$. □

Il punto centrale del branch and bound è il concetto di *eliminazione* (fathoming). Supponiamo che sia già nota una soluzione ammissibile di (P) di valore K. Si consideri un sottoproblema (P_i), e sia z'_i il valore ottimo di un rilassamento $r(P_i)$. Se $z'_i \geq K$, allora possiamo concludere che è inutile risolvere (P_i), in quanto la soluzione ottima di (P_i) sarà comunque non migliore di quella già nota, di valore K. Il problema P_i può dunque essere eliminato (e così pure il corrispondente nodo nell'albero di enumerazione).

Perché le suddette osservazioni possano essere efficaci in pratica in un algoritmo di branch and bound, è necessario che sia utilizzato un rilassamento del problema che abbia le seguenti proprietà:

1. Può essere risolto efficientemente, altrimenti il tempo di calcolo necessario a risolvere $r(P)$ rende l'intero procedimento troppo oneroso

2. Fornisce delle *buone* approssimazioni, ossia z' è abbastanza vicino a z.

Queste due esigenze possono essere in conflitto tra di loro: un rilassamento che fornisca buone approssimazioni risulta, spesso, difficile da risolvere.

L'idea di fondo del branch and bound è dunque quella di suddividere la regione ammissibile in sotto-regioni, e per ciascuna di esse risolvere un rilassamento, nella speranza che la soluzione ottima del rilassamento sia ammissibile oppure peggiore dell'ottimo corrente.

Un algoritmo specifico di branch and bound richiede poi di effettuare alcune scelte. Ad esempio, man mano che sono generati i sottoproblemi, questi vengono aggiunti a una struttura dati contenente i *problemi da risolvere*. La modalità di gestione di questa struttura dati può portare a visite in profondità, in ampiezza ecc. e da questa scelta può dipendere in modo essenziale l'efficienza dell'algoritmo. Inoltre, in molti casi è conveniente costruire euristicamente una soluzione x^H del problema (se possibile), detta *soluzione incombente*. Se, nel corso dell'algoritmo, vengono trovate altre soluzioni ammissibili migliori di questa, la soluzione incombente viene aggiornata. Al termine dell'enumerazione, il valore della soluzione incombente coincide col valore della soluzione ottima.

Ricapitolando, un algoritmo di branch and bound viene definito da:

- Rilassamenti e algoritmi di soluzione per i rilassamenti

- Regola di branching (generazione dei sottoproblemi)

- Selezione del prossimo sottoproblema da esaminare
- Algoritmi euristici di calcolo per la soluzione incombente

3.5.1 Il metodo di Falk e Horowitz

In questo paragrafo vedremo un metodo di branch and bound per risolvere il problema del crash time con costi concavi (3.8). Questo metodo è noto come *metodo di Falk e Horowitz*. Nel seguito, indichiamo con $R_i(d_i) = R_i(u_i - y_i)$ la funzione di costo (concava) associata all'attività i:

$$\begin{aligned} \min R(d) = \sum_{i=1}^{n} R_i(u_i - y_i) & \qquad (3.15) \\ s_j - s_i &\geq u_i - y_i \quad \forall i, j \in A \\ y_i &\leq u_i - l_i \quad \forall i \in V \\ s_n &\leq T \\ y_i, s_i &\geq 0 \quad \forall i \in V \end{aligned}$$

L'idea del metodo di Falk e Horowitz è di utilizzare un rilassamento in cui la regione ammissibile rimane la stessa di (3.15), ma si sostituisce la funzione obiettivo $R(d)$ con una funzione lineare $c(d)$ tale che $c(d) \leq R(d)$ per ogni vettore di durate d ammissibile. Il vantaggio è che, essendo $c(d)$ lineare, questo rilassamento è un problema di PL, e quindi risolvibile efficientemente.

Per ciascuna attività $i \in V$, la funzione $R_i(d_i)$ è definita tra l_i e u_i, punti nei quali vale $R_i(l_i)$ e $R_i(u_i)$. Consideriamo allora un segmento di retta passante per i punti $(l_i, R_i(l_i))$ e $(u_i, R_i(u_i))$. La pendenza di questo segmento (in valore assoluto) è dato dunque da $c_i = (R_i(l_i) - R_i(u_i))/(u_i - l_i)$. L'equazione della retta è evidentemente $\bar{R}_i(y_i) = R_i(u_i) + c_i y_i$ Se, nel problema (3.15), sostituiamo ciascuna funzione $R_i(d_i)$ con $\bar{R}_i(d_i)$, otteniamo un problema di programmazione lineare.

$$\begin{aligned} \min \quad R(u) + c_1 y_1 + \ldots c_n y_n &= R(u) + c^T y \qquad (3.16) \\ s_j - s_i &\geq u_i - y_i \quad \forall (i, j) \in A \\ y_i &\leq u_i - l_i \quad \forall i \in V \\ s_n &\leq T \\ y, s &\geq 0 \end{aligned}$$

ove si è indicato con $R(u)$ la somma $\sum_{i=1}^{n} R_i(u_i)$, che è una costante indipendente dallo schedule.

Si ha allora il seguente risultato:

LEMMA 5 *Il problema (3.16) è un rilassamento di (3.15).*

Dimostrazione L'insieme delle soluzioni ammissibili di (3.16) coincide con l'insieme delle soluzioni ammissibili di (3.15). Inoltre, per ogni soluzione (s, y) di (3.15), per la

concavità delle R_i, si ha $R(u-y) \geq R(u) + \sum c_i y_i$. Quindi, se (s^*, y^*) è la soluzione ottima di (3.15), si ha $R(u-y^*) \geq R(u) + c^T y^*$. ∎

Dunque, si consideri il problema rilassato, e sia $(\bar{s}, \bar{y})$ la sua soluzione ottima. Si noti che per ogni attività, nei due valori estremi $y_i = 0$ e $y_i = u_i - l_i$, si ha che $R_i(u_i - y_i) = \bar{R}_i(y_i)$. Se allora, nella soluzione ottima $\bar{y}$ del problema rilassato, si ha che $\bar{y}_i = 0$ oppure $\bar{y}_i = u_i - l_i$ per ogni attività i, si ha che $R(\bar{y}) = \bar{R}(\bar{y})$ e dunque la soluzione $\bar{y}$ è ottima anche per il problema originario.

Se invece esiste almeno una attività per cui $R(\bar{y}) > \bar{R}(\bar{y})$, è necessario effettuare il branching. In particolare, il branching è ottenuto applicando il seguente schema:

- scegli un'attività i per cui $R_i(u_i - \bar{y}_i) > \bar{R}_i(\bar{y}_i) = R_i(u_i) + c_i \bar{y}_i$, ovvero R_i non è lineare e $0 < \bar{y}_i < u_i - l_i$
- crea due nuovi problemi Q_1 e Q_2 ponendo, nel primo $u_i^I := u_i - \bar{y}_i$ e nel secondo $l_i^{II} := u_i - \bar{y}_i$ (per tutte le altre attività i limiti sono gli stessi che nel problema padre)
- risolvi Q_1 e Q_2 come (3.15).

Si procede quindi ricorsivamente in questo modo. Quando, in un sottoproblema, all'ottimo tutte le attività si trovano al valore minimo o massimo di durata, quel sottoproblema viene eliminato e $\bar{y}$ può – eventualmente – sostituire la soluzione incombente.

Esempio 17 *Si supponga che, per un'attività i, $u_i = 24$, $l_i = 12$, $R_i(u_i) = 10$ e $R_i(l_i) = 82$. La funzione viene approssimata con una retta di pendenza $c_i = (82-10)/12 = 6$. Se $\bar{y} = 4$, si ha $R_i(u_i - \bar{y}_i) = 34$, mentre il valore della funzione di crashing (concava) è $R(u_i - \bar{y}) > 34$ (Figura 3.10). Si deve allora procedere con la fase di branching, ovvero generando due sottoproblemi corrispondenti ad una nuova e più accurata linearizzazione della funzione di crashing negli intervalli $[12, 20]$, e $[20, 24]$ (Figura 3.11).*

∎

Esempio 18 *Il progetto in Figura 3.12 si compone di 2 attività (la 2 e la 3 nella figura, mentre la 1 e la 4 sono le attività fittizie di inizio e fine progetto). L'attività 2 deve finire prima che la 3 cominci.*

La durata dell'attività 2 può variare tra 2 e 4 settimane, mentre la durata dell'attività 3 può variare tra 1 e 4 settimane. L'attività 3 ha un costo che varia linearmente con il tempo, secondo la legge $R_3(d_3) = -2.5d_3 + 11$, mentre l'attività 2 ha un costo (concavo) dato da $R_2(d_2) = -d_2^2 + 4d_2 + 1$. Il progetto ha una deadline pari a 7.

Introducendo le variabili s_1, s_2, s_3, ed s_4 per indicare l'istante di inizio delle attività e y_2, y_3 per indicare la riduzione della durata delle attività 2 e 3 rispetto alle durate massime consentite, scriviamo il problema di programmazione lineare corrispondente al primo problema di PL nel metodo di Falk e Horowitz (nodo radice dell'albero di enumerazione).

Per quanto riguarda l'attività 2, approssimiamo la curva $R_2(d_2) = -d_2^2 + 4d_2 + 1$ con la retta $R_2'(d_2) = 9 - 2d_2$, ovvero $\bar{R}_2(y_2) = 1 + 2y_2$. Per l'attività 3, essendo il costo lineare (caso particolare di funzione concava), basta solo riscriverlo in funzione dell'accorciamento y_3, ottenendo $\bar{R}_3(y_3) = 1 + 2.5y_3$. La prima formulazione di Falk e Horowitz è quindi

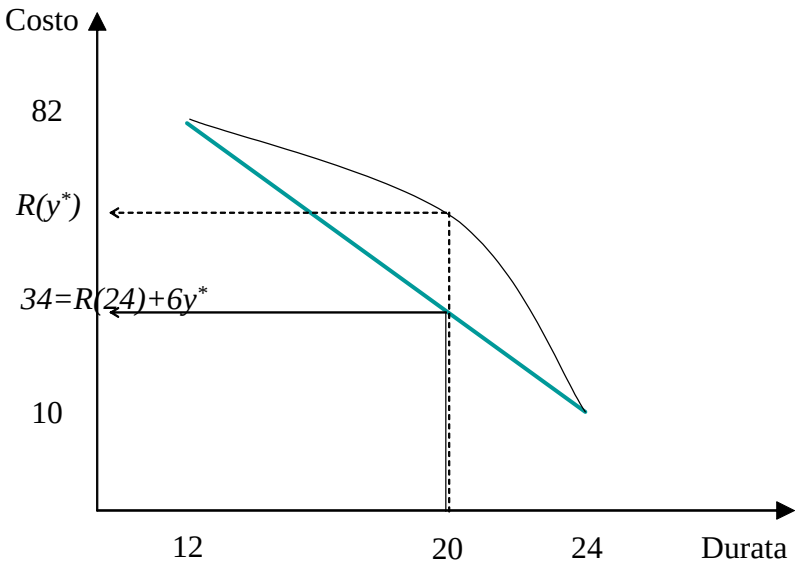

Figura 3.10: Linearizzazione di Falk e Horowitz (Soluzione PL).

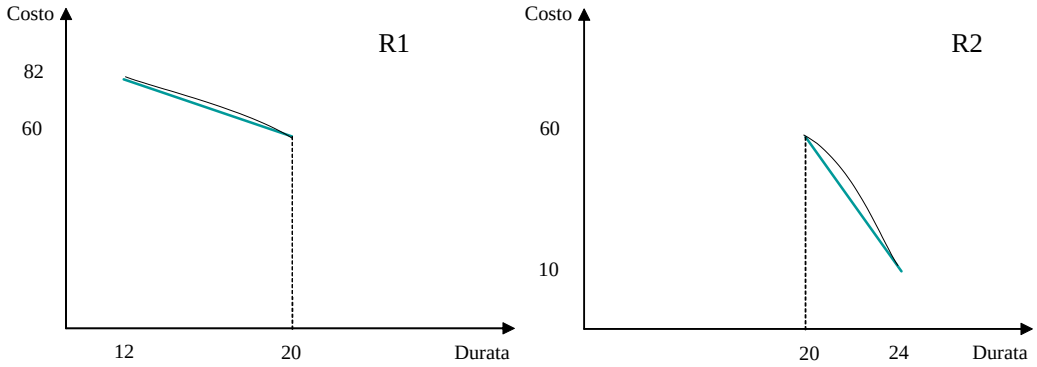

Figura 3.11: Linearizzazione di Falk e Horowitz (branching).

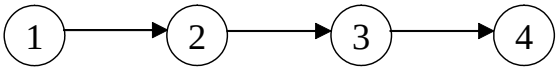

Figura 3.12: Progetto nell'Esempio 18.

$$
\begin{aligned}
min\ \ \bar{R}(y) = 2 + 2y_2 + 2.5y_3 & \qquad (3.17)\\
s_2 - s_1 &\geq 0\\
s_3 - s_2 &\geq 4 - y_2\\
s_4 - s_3 &\geq 4 - y_3\\
s_4 &\leq 7\\
y_2 &\leq 2\\
y_3 &\leq 3\\
y, s &\geq 0
\end{aligned}
$$

La soluzione ottima del problema (3.17) risulta essere: $\bar{s}_1 = 0$, $\bar{s}_2 = 0$, $\bar{s}_3 = 3$, $\bar{s}_4 = 7$, $\bar{y}_2 = 1$, $\bar{y}_3 = 0$. *Il valore della soluzione ottima del problema rilassato è* $\bar{R}_2(\bar{y}_2) + \bar{R}_3(\bar{y}_3) = 4$, *mentre in corrispondenza alla soluzione* $\bar{y}$ *la funzione concava originaria vale* $R(\bar{d}) = R_2(\bar{d}_2) + R_3(\bar{d}_3) = R_2(3) + R_3(4) = (-9 + 12 + 1) + (-10 + 11) = 5$. *L'attività 3 ha crashing cost lineari, e dunque non occorrerà mai effettuare il branching rispetto ad essa. (Peraltro, nella soluzione ottima di (3.17) non subisce alcun accorciamento.) Invece, l'attività 2 ha un accorciamento pari a 1, strettamente compreso tra 0 e* $u_2 - l_2 = 2$. *La funzione di costo originale vale* $R_2(\bar{d}_2) = 4$, *mentre* $\bar{R}_2(\bar{y}_2) = \bar{R}_2(1) = 3$. *Quindi, il valore del rilassamento è diverso dal valore della funzione di costo originale e dobbiamo passare alla fase di branching. Si noti che, siccome la regione ammissibile del problema rilassato (3.17) è uguale a quella del problema originario (3.15), disponiamo già di una soluzione incombente, di valore 5.*

Scriviamo allora i due problemi di programmazione lineare associati al primo branching. Necessariamente, dovremo scegliere l'attività 2. Otteniamo $u_2 - \bar{y}_2 = 4 - 1 = 3$, *che sarà posto come valore massimo per l'attività 2 nel problema* Q_1, *e come valore minimo per la stessa attività nel problema* Q_2. *Generiamo e risolviamo per primo* Q_1, *corrispondente a* $2 \leq d_2 \leq 3$ *(Figura 3.13). Si ha il seguente problema (sottolineate le parti che sono variate rispetto a (3.17)):*

$$
\begin{aligned}
\min\ \ \bar{R}^1(y) = \underline{5} + \underline{y_2} + 2.5y_3 & \qquad (3.18)\\
s_2 - s_1 &\geq 0\\
s_3 - s_2 &\geq \underline{3} - y_2\\
s_4 - s_3 &\geq 4 - y_3\\
s_4 &\leq 7\\
y_2 &\leq \underline{1}\\
y_3 &\leq 3\\
y, s &\geq 0
\end{aligned}
$$

Si noti come sia diminuito il coefficiente di y_2 *nella funzione obiettivo. La soluzione ottima del rilassamento è* $\bar{y}^1_2 = 0$ *e* $\bar{y}^1_3 = 0$, *di valore* $\bar{R}^1(\bar{y}^1) = 5$. *Poiché il valore dell'ottimo del rilassamento e il valore ottimo del problema coincidono, non occorre fare*

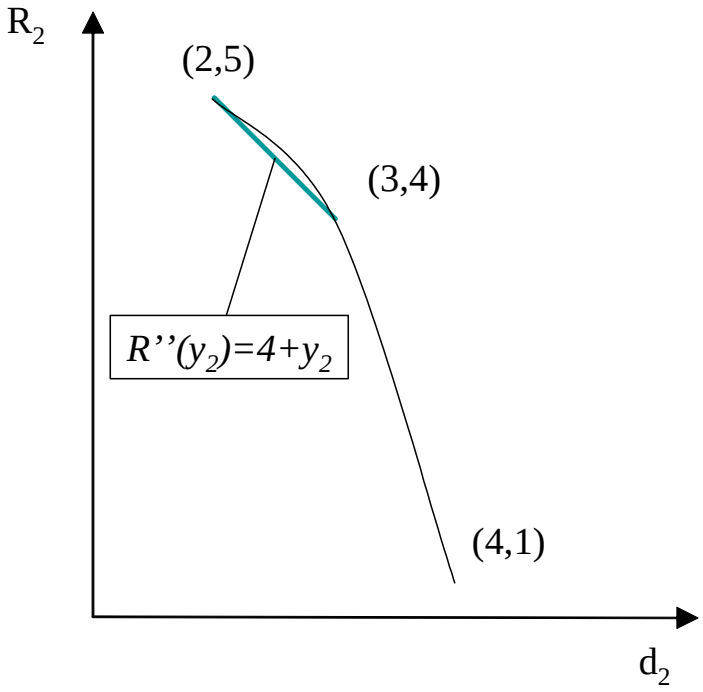

Figura 3.13: Sottoproblema Q_1 nel primo branching per l'Esempio 18.

ulteriori branching. Peraltro, disponendo già di una soluzione incombente di valore 5, il nodo dell'albero di enumerazione sarebbe stato comunque eliminato.

Generiamo e risolviamo ora Q_2 (3.19). Si ha il seguente problema (sottolineate le parti che sono variate rispetto a (3.17)):

$$\begin{aligned} \min \quad \bar{R}^2(y) = \underline{2} + \underline{3y_2} + 2.5y_3 & \qquad (3.19) \\ s_2 - s_1 &\geq 0 \\ s_3 - s_2 &\geq 4 - y_2 \\ s_4 - s_3 &\geq 4 - y_3 \\ s_4 &\leq 7 \\ y_2 &\leq \underline{1} \\ y_3 &\leq 3 \\ y, s &\geq 0 \end{aligned}$$

La soluzione ottima del rilassamento è $\bar{y}_2^2 = 0$ e $\bar{y}_3^2 = 1$, di valore $\bar{R}^2(\bar{y}^2) = 4.5$. Come nel caso precedente, anche ora si ha che il valore della funzione obiettivo del rilassamento e di quella del problema originario coincidono (in effetti d_2 è all'estremo superiore), e quindi non occorre fare ulteriori branching. Inoltre, la nuova soluzione ha valore migliore della soluzione incombente, che verrà quindi aggiornata. Essendo vuota la lista dei problemi, l'attuale soluzione incombente è la soluzione ottima. In definitiva, le durate ottime sono dunque $d_2^* = 4 - \bar{y}_2^2 = 4$ e $d_3^* = 4 - \bar{y}_3^2 = 3$.

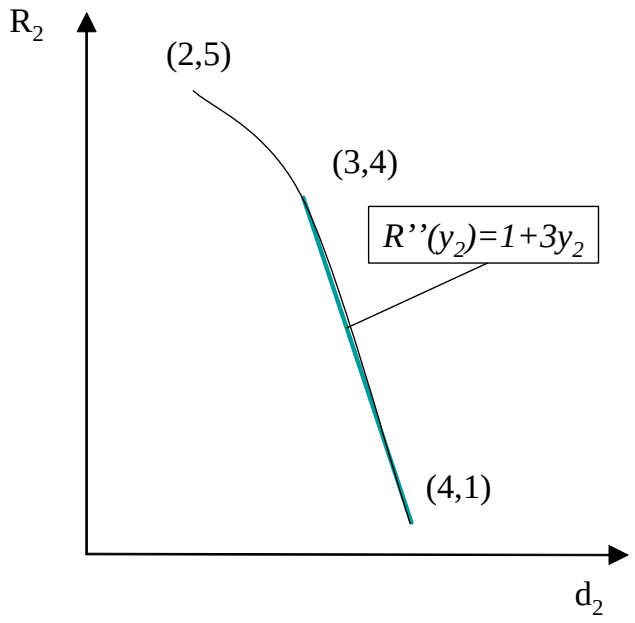

Figura 3.14: Sottoproblema Q_2 nel primo branching per l'Esempio 18.

□

Capitolo 4

Project Scheduling con vincoli sulle risorse

Ogni attività, per essere eseguita, richiede risorse. Nei processi di cost management e time management, queste risorse sono tenute in considerazione, in quanto dal loro valore e dalla modalità con cui sono utilizzate, dipende in generale il costo dell'attività, nonché la sua durata. Tra le attività del project manager troviamo quindi l'identificazione e classificazione delle risorse necessarie al compimento del progetto, e la stima della richiesta di risorse da parte di ogni attività.

Fin qui, si è supposto che le risorse erano comunque disponibili quando richieste. Nel seguito del testo, invece, vogliamo modellare esplicitamente il ruolo che alcune risorse giocano nel determinare lo svolgimento temporale del progetto. Ad esempio, nella costruzione di un edificio, per alcune attività è necessario impiegare una grossa gru. Se si dispone di una sola gru, queste attività non potranno essere svolte contemporaneamente, ed è palesemente antieconomico pensare di acquisire una gru solo per quell'attività.

4.1 Tipi di risorse

Alcuni autori hanno classificato le risorse in tre categorie di base: risorse *rinnovabili*, le risorse *non rinnovabili* e le risorse *doppiamente vincolate*. Per le risorse *rinnovabili*, la quantità disponibile è utilizzabile in ogni periodo di tempo dell'orizzonte temporale. Esempi di risorse rinnovabili sono la manodopera, i macchinari, la strumentazione, solo per citarne alcuni. Tipicamente, le risorse rinnovabili presentano vincoli di disponibilità in ogni periodo di tempo, ma non sulla quantità utilizzata durante l'intero orizzonte. Ad esempio, può esistere un vincolo giornaliero su quanti tecnici specializzati possono essere utilizzati ogni giorno, ma gli stessi tecnici possono essere impiegati anche durante l'arco di tutto il progetto. I modelli che vedremo in questo capitolo fanno riferimento a risorse rinnovabili. In generale, la quantità disponibile di una certa risorsa rinnovabile può variare da periodo a periodo, come pure può essere variabile la quantità richiesta nel tempo da ciascuna attività. Per motivi di semplicità, nei modelli che vedremo, tratteremo solo il caso in cui vi sia *un solo* tipo di risorsa (rinnovabile), la disponibilità di questa risorsa è costante e pari ad a unità, e ciascuna attività i richiede r_i unità di risorsa durante tutta

la sua esecuzione.

Anche se non li tratteremo, è bene ricordare che esistono anche altri tipi di risorse. Per le *risorse non rinnovabili*, la disponibilità è determinata su tutto l'orizzonte temporale e non su ogni singolo periodo del progetto. Tipico esempio di risorsa non rinnovabile sono le materie prime disponibili in magazzino: una volta consumate, non sono più riutilizzabili. Infine, una terza categoria di risorse è quella delle *risorse doppiamente vincolate*, ovvero la cui disponibilità risulta essere limitata sia in ogni periodo che per l'intero progetto. L'esempio più significativo di risorsa doppiamente vincolata è il capitale: in ogni periodo le uscite di cassa non possono superare un dato ammontare, e inoltre le uscite complessive non possono superare il budget. Un altro esempio, rilevante in certi tipi di progetti, è l'energia elettrica.

Oltre a questa classificazione, altre classificazioni delle risorse sono comunemente adoperate, come ad esempio suddividere le risorse in *continue* e *discrete*, oppure in risorse *interrompibili* o *non interrompibili*. Nelle risorse continue, la quantità utilizzata può variare con continuità (es. elettricità, energia...), mentre nelle risorse discrete la quantità è discretizzata (es. manodopera, giorni...). Vengono considerate risorse interrompibili quelle risorse per cui tutte le attività che le richiedono sono *interrompibili*, ovvero la risorsa richiesta può essere assegnata ad altra attività (e l'attività corrente verrà completata in seguito). Ad esempio, il micro-processore di un computer è una risorsa interrompibile. Nel seguito, supporremo sempre le risorse non interrompibili.

4.2 Introduzione al RCPSP

Il problema di calcolare il makespan di un progetto in presenza di vincoli sulle risorse, assumendo che ogni attività ne richiede una certa quantità per essere eseguita, prende il nome in letteratura di *Resource Constrained Project Scheduling Problem (RCPSP)*. In particolare, il modello del RCPSP assume le seguenti ipotesi:

- vincoli di precedenza tipo finish-start senza time lag, espressi da un grafo AoN $G = (V, A)$
- *una sola* risorsa, rinnovabile, disponibile in ogni istante in quantità a;
- attività non segmentabili temporalmente (non interrompibili)
- lo svolgimento dall'attività i richiede la quantità $r_i \geq 0$ di risorsa durante tutto il suo svolgimento

Certamente, è possibile concepire modelli più generali di questo, ad esempio con più risorse, o con vincoli di precedenza generalizzati. Tuttavia, al di là di una maggiore complessità teorica, gli aspetti strutturali sono già catturati in modo abbastanza significativo dal modello che vedremo in questo capitolo.

Del resto, già in questa forma il problema RCPSP è un problema difficile. Per convincersene, basta osservare che RCPSP generalizza un noto problema NP-completo, noto col nome di Partition. Infatti, consideriamo un'istanza di Partition, ossia n oggetti di peso (intero) $a_1, a_2, \ldots, a_n$. Il problema di Partition consiste nel chiedersi se è possibile

dividere gli n oggetti in due insieme aventi esattamente lo stesso peso, $\sum a_i/2$. Possiamo allora associare all'istanza di PARTITION un'istanza di RCPSP definita come segue: per ciascun oggetto definiamo una attività, avente durata unitaria, e che richiede una quantità a_i di risorsa. Sono disponibili b unità di risorsa. È allora evidente che è possibile partizionare gli n oggetti in due insiemi di uguale peso se e solo se è possibile terminare il progetto in due unità di tempo. Dunque, se sapessimo risolvere RCPSP, sapremmo risolvere PARTITION, il che dimostra che RCPSP è almeno tanto difficile quanto lo è PARTITION.

Un utile strumento per rappresentare uno schedule nel caso di progetti con risorsa limitata è un diagramma di Gantt bidimensionale, in cui ogni attività i viene rappresentata come un rettangolo di lunghezza pari alla sua durata d_i, e di altezza pari alle unità di risorsa richieste r_i. Il diagramma mostra visivamente quale è il makespan del progetto e quante sono le risorse richieste per il suo svolgimento per ogni istante di tempo. Un'avvertenza: si rappresentano le attività con rettangoli per chiarezza visiva, ma non c'è alcun motivo concettuale per cui i rettangoli non possano essere "spezzati": ad esempio, in Figura 4.2, un'altra attività con $r_i = 2$ può essere schedulata tra $t = 6$ e $t = 9$; solo, occorrerà rappresentarla con due rettangoli.

La presenza di risorse limitate può costringere ad allungare i tempi di completamento, rispetto al caso in cui qualunque insieme di attività non legate da vincoli di precedenza può essere svolto in parallelo.

ESEMPIO 19 *Sia dato il progetto rappresentato in Figura 4.1 e nella seguente tabella:*

i	d_i	r_i	EST_i
1	0	0	0
2	2	2	0
3	7	3	0
4	3	4	2
5	4	4	0
6	8	3	5
7	6	2	4
8	4	3	7
9	2	4	11
10	0	0	13

Una possibile soluzione è rappresentata, tramite il diagramma di Gantt, in Figura 4.2. Questa soluzione richiede l'uso di 11 unità di risorsa e verrà completata dopo 13 unità temporali. Nel caso in cui il vincolo sulla risorsa valesse 8 unità, allora sarebbe necessario individuare una nuova soluzione. Una possibile soluzione è mostrata in Figura 4.3 con tempo di completamento pari a 15.

□

Nei modelli che vedremo nel seguito, si fa riferimento a una suddivisione dell'orizzonte temporale in *periodi temporali* o *slot*. Per $t = 1, \ldots, T$, lo slot t va dall'istante $t - 1$ all'istante t. Tali slot corrispondono a una suddivisione abbastanza fine da ritenere che,

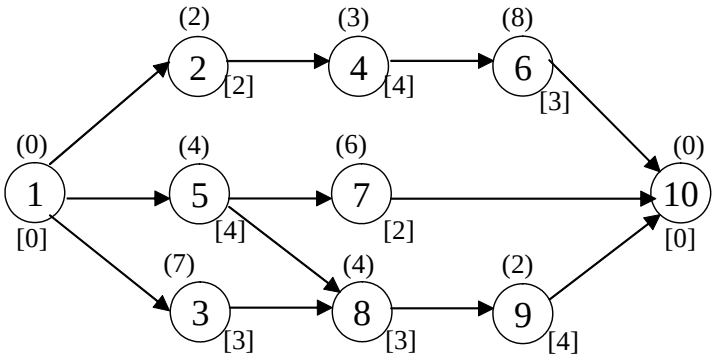

Figura 4.1: Activity on Node del progetto.

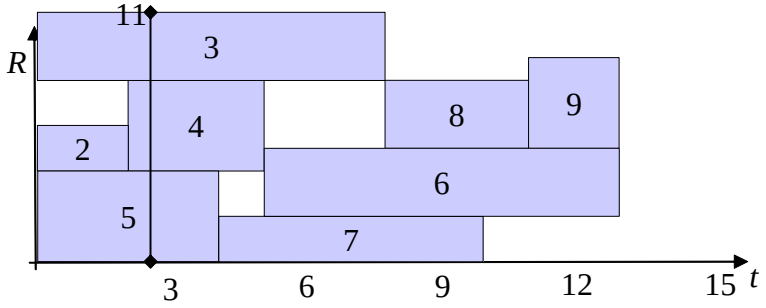

Figura 4.2: Diagramma di Gantt con vincoli sulla risorsa (11 unità).

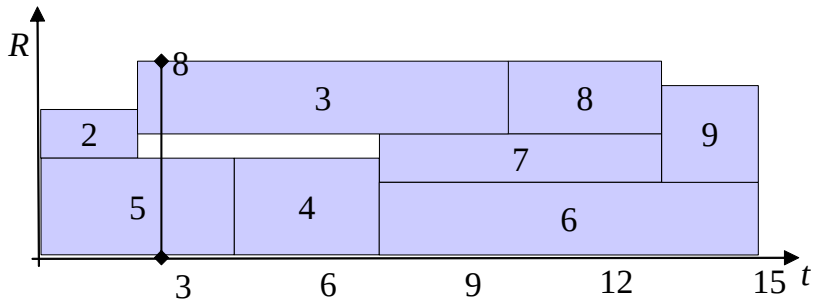

Figura 4.3: Diagramma di Gantt con vincoli sulla risorsa (8 unità).

all'interno di uno slot, non si verifichino variazioni nell'insieme delle attività correntemente in esecuzione, e di conseguenza nella quantità di risorsa impegnata. Una suddivisione troppo fine può riflettersi nella eccessiva complessità dei modelli matematici, come sarà più chiaro nel seguito. L'ampiezza di uno slot va dunque presa pari alla minima risoluzione necessaria, ad esempio 1 giorno, 1 settimana, 1 mese...

4.3 Formulazioni matematiche

In questa sezione vogliamo presentare alcune formulazioni di programmazione matematica per il RCPSP. In queste formulazioni facciamo riferimento a una suddivisione dell'orizzonte temporale in periodi. Ricordiamo che a indica la quantità di risorsa disponibile, r_i la quantità richiesta dall'attività i, e d_i la sua durata (nota e deterministica). Useremo inoltre le seguenti notazioni:

- T: orizzonte temporale. Tale valore dovrà rappresentare il più alto indice di un periodo in cui può essere in esecuzione un'attività
- s_i : istante di inizio dell'attività i
- $S(t)$: insieme delle attività *in corso di svolgimento* nel periodo t, $t = 1, \ldots, T$.

Il problema può essere allora formulato nel modo seguente:

$$\begin{aligned} \min\, & s_n & & & (4.1)\\ & s_j & \geq & s_i + d_i \quad (i,j) \in E \\ & s_1 & = & 0 \\ & \sum_{i \in S(t)} r_i & \leq & a \end{aligned}$$

Questa formulazione non è una formulazione di PL e ha solo un valore simbolico: la difficoltà sta nell'esprimere correttamente il vincolo sulle risorse, ossia, in definitiva, l'insieme $S(t)$. Vediamo di seguito alcune formulazioni del problema, in cui sono utilizzate diverse tecniche per esprimere il vincolo sulle risorse.

4.3.1 Formulazione di Pritsker

La prima formulazione che esaminiamo è stata introdotta da Pritsker nel 1969. In questa formulazione, si introducono delle variabili binarie che rappresentano quando l'attività i-esima può iniziare. Precisamente, per ogni $i \in V$ e $t \in \{1, \ldots, T\}$, definiamo una variabile binaria x_{it} tale che $x_{it} = 1$ se e solo se l'attività i comincia nell'istante t (ossia se $s_i = t$), e $x_{it} = 0$ altrimenti. Per limitare il numero di variabili, osserviamo che possiamo limitarci a prendere in considerazione valori dell'indice i non inferiori a EST_i, dove tale earliest start time è calcolato trascurando i vincoli di risorsa, e dunque con i normali algoritmi visti nel §2.2.2.

Vediamo a quali vincoli matematici devono sottostare le variabili per rappresentare uno schedule ammissibile:

1) *L'attività i comincia in un singolo istante di tempo.* Questo fatto può esprimersi scrivendo

$$\sum_{t=EST_i}^{T-d_i} x_{it} = 1, \forall i \in V \tag{4.2}$$

La (4.2) consente dunque di legare i valori delle x_{it} alle s_i, scrivendo

$$s_i = \sum_{t=EST_i}^{T-d_i} t x_{it} \tag{4.3}$$

Si noti che l'estremo superiore può essere reso ancora più stringente se sussistono ulteriori specifiche, ossia se ad esempio T è da considerarsi una deadline. In tal caso si può pensare di calcolare, sempre coi metodi visti nel §2.2.2, anche il valore LST_i, che può sostituire $T - d_i$ come estremo superiore.

2) *Vincoli di precedenza.* Trattandosi di relazioni finish-start senza time lag, deve aversi $s_j \geq s_i + d_i$. La (4.3) rende immediata la scrittura di questi vincoli:

$$\sum_{t=EST_j}^{LST_j} t x_{jt} \geq \sum_{t=EST_i}^{LST_i} t x_{it} + d_i, \forall (i,j) \in E. \tag{4.4}$$

3) *Vincolo sulla disponibilità di risorsa.* L'attività i è sicuramente in svolgimento nel periodo t (quindi utilizza r_i unità di risorsa nel periodo t) se inizia in un qualunque istante compreso fra $t - d_i$ e $t - 1$ (estremi inclusi). Dunque, se si ha

$$\sum_{q=t-d_i}^{t-1} x_{iq} = 1 \tag{4.5}$$

allora l'attività i *è in svolgimento* nel periodo t (in quanto $t - d_i \leq s_i \leq t - 1$). Se invece vale

$$\sum_{q=t-d_i}^{t-1} x_{iq} = 0 \tag{4.6}$$

allora l'attività i *non* è in svolgimento nel periodo t. Dalla (4.5), si ha dunque che se i è in svolgimento nel periodo t vale la

$$r_i \sum_{q=t-d_i}^{t-1} x_{iq} = r_i$$

mentre altrimenti vale la (4.6), e dunque

$$r_i \sum_{q=t-d_i}^{t-1} x_{iq} = 0$$

perciò, l'utilizzo di risorsa da parte dell'attività i nel periodo t è dato da

$$r_i \sum_{q=t-d_i}^{t-1} x_{iq} \tag{4.7}$$

Tenendo conto che l'attività i può cominciare non prima di EST_i e non dopo LST_i, si può raffinare la (4.7) scrivendo

$$r_i \sum_{q=\max\{t-d_i, EST_i\}}^{\max\{t-1, LST_i\}} x_{iq}.$$

A questo punto il vincolo sulla disponibilità complessiva di risorsa relativa al periodo t diventa immediato:

$$\sum_{i=1}^{n} r_i \sum_{q=\max\{t-d_i, EST_i\}}^{\min\{t-1, LST_i\}} x_{iq} \leq a, \qquad t = 0, \ldots, T. \tag{4.8}$$

4) *Funzione obiettivo.* Si vuole minimizzare il tempo di completamento del progetto, e questo si può facilmente esprimere come

$$s_n = \min \sum_{t=EST_n}^{LST_n} t x_{nt} \tag{4.9}$$

La formulazione di Pritsker consiste dunque della funzione obiettivo (4.9), sotto i vincoli di unicità dell'istante d'inizio delle attività (4.2), di rispetto delle precedenze temporali (4.4), di disponibilità della risorsa (4.8) e tenendo conto che le variabili sono binarie, ovvero $x_{it} \in \{0, 1\}$.

4.3.2 Formulazione di Kaplan

Vediamo ora una formulazione alternativa per il RCPSP, introdotta da Kaplan nel 1988. Anche in questa formulazione si fa uso di variabili binarie, ma a differenza della formulazione di Pritsker, qui le variabili binarie indicano *in quali periodi* l'attività i è in svolgimento. Definiamo cioè delle variabili binarie x_{ip} come $x_{ip} = 1$ se l'attività i è in svolgimento nel periodo p, e 0 altrimenti, per ciascuna attività i e $p \in P = \{1, \ldots, T\}$.

Un piccolo inconveniente è che così facendo, le attività fittizie di inizio e fine progetto non sono mai in svolgimento (perché hanno durata 0). Questo problema si risolve assegnando una durata arbitraria pari ad 1 alle attività fittizie. Di conseguenza EST_i e LST_i aumentano di 1, per ogni i.

Come già per la formulazione di Pritsker, sfruttiamo il fatto che l'attività i non può svolgersi prima del proprio earliest start time (calcolato senza tener conto della risorsa limitata) e dopo il proprio latest finish time (laddove vi sia una deadline), ossia sarà $x_{ip} = 0$, per $p \leq EST_i$ e $p \geq LFT_i + 1$. I vincoli saranno espressi come segue:

1) *Vincoli di durata.* L'attività i deve essere in svolgimento per esattamente d_i periodi:

$$\sum_{p=EST_i}^{T} x_{ip} = d_i, i \in V \tag{4.10}$$

2) *Nonpreemption.* L'attività i non può essere interrotta e ripresa, ovvero i periodi di svolgimento dell'attività devono essere contigui. Questo può esprimersi per mezzo di questo vincolo:

$$d_i(x_{ip} - x_{i,p+1}) - \sum_{q\in[p-d_i+1,p-1]} x_{iq} \leq 1, \quad i \in V, \quad p = EST_i, \ldots, T-1 \tag{4.11}$$

Vediamo il perché di questa espressione. Anzitutto, osserviamo che se $x_{ip} \leq x_{i,p+1}$, questo vincolo è sicuramente soddisfatto, in quanto il primo termine o vale 0 oppure vale $-d_i$. Rimane dunque da considerare il caso in cui $x_{ip} = 1$ e $x_{i,p+1} = 0$, ossia vediamo cosa accade allorché i è in esecuzione nel periodo p e non lo è più nel periodo $p+1$. Perché la (4.11) valga, deve essere

$$\sum_{q\in[p-d_i+1,p-1]} x_{iq} \geq d_i - 1$$

e poiché i termini della sommatoria sono esattamente $d_i - 1$, si ha che *tutte* le variabili x_{ip} nella sommatoria devono essere pari a 1. Considerando poi i vincoli (4.10), ecco che non vi possono essere altre variabili $x_{ip} = 1$, e dunque in definitiva, per ogni attività i, in una soluzione ammissibile devono esserci esattamente d_i variabili consecutive pari a 1.

3) *Vincoli di precedenza.* Consideriamo due attività i e j per cui esiste un vincolo di precedenza temporale. Anzitutto, le due attività non possono svolgersi simultaneamente:

$$x_{jp} + x_{ip} \leq 1 \quad p = 1, \ldots, T \tag{4.12}$$

Inoltre, l'attività j non può essere in svolgimento nel periodo p se l'attività i non è stata in svolgimento in qualche periodo precedente a p

$$x_{jp} \leq \sum_{q=1}^{p} x_{iq} \qquad p = 1, \ldots, T, (i,j) \in A \tag{4.13}$$

I vincoli (4.12) e (4.13) possono essere rafforzati con semplici considerazioni. Infatti, se j è in svolgimento in $EST_i + 1$ lo è anche in $EST_i + d_i = EFT_i$, per cui la sommatoria nella (4.13) può partire da EFT_i. Inoltre, se j è in svolgimento nel periodo p, ciò vuol dire che i deve essere finita entro il periodo $p-1$, e dunque doveva essere attiva in almeno un periodo non successivo a $p - d_i$. Quindi si può scrivere

$$x_{jp} \leq \sum_{q=EFT_i}^{p-d_i} x_{iq} \qquad p = 1, \ldots, T, (i,j) \in A. \tag{4.14}$$

4) *Vincolo sulla disponibilità di risorsa.* Questo vincolo si esprime imponendo che la somma delle richieste di risorsa delle attività in svolgimento in ciascun periodo p non deve superare la disponibilità della stessa, ovvero

$$\sum_{i=1}^{n} r_i x_{ip} \leq a, i = 1, \ldots, T. \tag{4.15}$$

5) *Funzione obiettivo.* La minimizzazione del makespan (periodo in cui l'attività n di fine progetto, che dura un solo periodo, è in svolgimento) si può scrivere semplicemente

$$\min \sum_{p=1}^{T} p x_{np}. \tag{4.16}$$

In definitiva, la formulazione di Kaplan consiste della funzione obiettivo (4.16), con i vincoli di durata delle attività (4.10), di non interruzione (4.11), di precedenza temporale (4.14) e di disponibilità della risorsa (4.15), tenendo conto che le variabili sono binarie, ovvero $x_{ip} \in \{0, 1\}$.

4.3.3 Formulazione di Mingozzi

Vediamo infine una formulazione (di Mingozzi) apparentemente più complessa, che però all'atto pratico risulta superiore, in termini di tempi di calcolo.

Anche questa formulazione fa uso del concetto di slot temporale, e si basa sulla considerazione che in ogni slot è possibile definire un insieme di operazioni in esecuzione. L'idea è quindi di decomporre il problema rispetto agli slot e scegliere le operazioni in esecuzione per ogni slot in modo da rispettare tutti i vincoli del problema.

Prima di introdurre i vincoli, è necessario dare alcune definizioni. Dato un insieme $S \subseteq V$ di attività, diremo che è *ammissibile* se non sono legate da vincoli di precedenza e se, eseguite contemporaneamente, non violano la disponibilità di risorsa. Indichiamo con $\mathcal{S}$ l'insieme di tutti gli insiemi ammissibili.

Dato un insieme $S_k \in \mathcal{S}$, il *subset earliest start time* ($EST(S_k)$) è il minimo slot in cui tutte le attività di S_k possono essere in svolgimento simultaneamente (si ricordi che lo slot p va dall'istante $p-1$ all'istante p):

$$EST(S_k) = \max_{i \in k}\{EST_i\} + 1 \tag{4.17}$$

Analogamente, il *subset latest finish time* ($LFT(S_k)$) è il massimo slot in cui tutte le attività di k possono essere ancora in svolgimento simultaneamente:

$$LFT(S_k) = \min_{i \in k}\{LFT_i\} \tag{4.18}$$

Data un'attività $i \in V$, la *famiglia ammissibile* per i è la famiglia $S(i) \subseteq \mathcal{S}$ degli insiemi ammissibili che contengono l'attività i

$$S(i) = \{S_k \in \mathcal{S} : i \in S_k\} \tag{4.19}$$

Le variabili della formulazione di Mingozzi sono definite come segue. Si consideri un insieme ammissibile S_k, e l'insieme degli slot in cui tutte le attività di S_k possono essere in esecuzione simultaneamente, $I_k = \{EST(S_k), EST(S_k)+1, \ldots, LFT(S_k)\}$. Per ogni $p \in I_k$, definiamo una variabile binaria y_{kp}, pari a 1 se tutte le attività dell'insieme S_k sono in svolgimento nel periodo p, e pari a 0 altrimenti. Inoltre, per ogni attività $i \in V$ e per ogni istante $t \in \{EST_i, \ldots, LST_i\}$, definiamo una variabile binaria x_{it}, pari a 1 se l'attività i inizia all'istante t ($s_i = t$), e pari a 0 altrimenti. Vediamo i vincoli che è necessario imporre.

1) *Un solo insieme ammissibile k può essere in svolgimento in ogni periodo.* Questo può esprimersi semplicemente come

$$\sum_{k=1}^{|\mathcal{S}|} y_{kp} \leq 1 \quad p \in [1 \dots T] \tag{4.20}$$

2) *Vincoli di durata.* Ogni attività i deve essere in svolgimento in esattamente d_i periodi

$$\sum_{S_k \in S(i)} \sum_{p=EST(S_k)}^{LFT(S_k)} y_{kp} = d_i \qquad i \in V \tag{4.21}$$

3) *Svolgimento delle attività in periodi consecutivi.* Questi vincoli legano le variabili y_{kp} alle x_{it}. Se, nel periodo $t+1$, è in svolgimento un insieme $S_k \in S(i)$, ma nessun insieme $S_k \in S(i)$ è in svolgimento nel periodo t, allora vuol dire che l'attività i *inizia* proprio nell'istante t

$$x_{it} \geq \sum_{S_k \in S(i)} y_{k,t+1} - \sum_{S_k \in S(i)} y_{kt} \qquad t \in [EST_i, LST_i] \tag{4.22}$$

Inoltre, ogni attività comincia in esattamente un istante

$$\sum_{t \in EST_i}^{LST_i} x_{it} = 1, i \in V \tag{4.23}$$

Si osservi che gli ultimi due vincoli (4.22) e (4.23) implicano che tutti gli insiemi in svolgimento che contengono una stessa attività i (appartenenti a $S(i)$), sono in svolgimento in periodi *consecutivi*. Infatti, se l'attività i avesse luogo in due intervalli non consecutivi, chiamati p e q gli slot iniziali dei due intervalli, per le (4.22) sarebbe $x_{ip} = 1$ e $x_{iq} = 1$, che violerebbe però il vincolo (4.23).

4) *Vincoli di precedenza.* Se due attività i e j, legate da un vincolo di precedenza, hanno intervalli $[EST_i, LFT_i]$ e $[EST_j, LFT_j]$ non disgiunti, è necessario imporre esplicitamente il vincolo di precedenza. Con le variabili x_{it} questo è comunque semplice, in quanto possiamo scrivere

$$x_{jt} \leq \sum_{q=EST_i}^{t-d_i} x_{iq} \qquad t = 1, \dots, T, (i,j) \in A \tag{4.24}$$

5) *Funzione obiettivo.* Analogamente alla formulazione di Kaplan, la funzione obiettivo si può scrivere:

$$\min \sum_{t=EST_n}^{LST_n} t x_{nt} \tag{4.25}$$

In definitiva, la formulazione di Mingozzi consiste nella funzione obiettivo (4.25), soggetta ai vincoli che un solo insieme ammissibile sia in svolgimento (4.20), che ogni attività i deve essere in svolgimento in esattamente d_i periodi (4.21), che tutte le attività sono in svolgimento in periodi contigui (4.22,4.23) e che tutti i vincoli di precedenza sono rispettati (4.24).

Input: Istanza di RCPSP
Output: Schedule
begin
  while la soluzione è parziale
    Seleziona un insieme di attività
    Assegna un istante di inizio a queste attività compatibilmente con gli assegnamenti precedenti
  end
end

Figura 4.4: Algoritmo greedy

La superiorità della formulazione di Mingozzi rispetto alle altre due è legata al fatto che tutti i vincoli hanno coefficienti pari a 0, 1 o -1, il che implica che il suo rilassamento sia tipicamente più forte degli altri.

4.4 Euristiche

Le formulazioni viste consentono, in linea di principio, di trovare la soluzione ottima di un'istanza di RCPSP. Purtroppo, in casi di grandi dimensioni, quali quelli che possono capitare in pratica, l'inerente complessità del problema può impedire di risolvere all'ottimo il problema in tempi *ragionevoli*. Per questo si ricorre spesso ad approcci *euristici*, che, in tempi di calcolo contenuti, forniscono mediamente soluzioni di buona qualità, ma non sono in grado di darne una garanzia teorica: ossia, su alcune istanze può capitare che la soluzione fornita dall'euristica sia piuttosto lontana dall'ottimo. Tuttavia, se l'euristica è "buona", in genere il risparmio in tempo di calcolo è tale da rendere il risultato accettabile.

Tipicamente, le euristiche si possono suddividere in due grandi famiglie:

- *Euristiche Costruttive*: partono da uno schedule vuoto e aggiungono attività fino a ottenere uno schedule ammissibile.
- *Euristiche Migliorative*: partono da uno o più piani ammissibili (eventualmente calcolati con euristiche costruttive) e cercano di migliorarli mediante una sequenza di aggiustamenti. A questa grande categoria appartengono gli algoritmi meta-euristici.

4.4.1 Euristiche costruttive

Vedremo due tipi di euristiche di tipo costruttivo, ambedue di tipo *greedy* (Figura 4.4), ossia costruiscono lo schedule iterativamente. In particolare, tutte le attività sono inizialmente ordinate in base a una *regola di priorità*, dopo di che, ad ogni iterazione dell'algoritmo, una (greedy seriale) o più attività (greedy parallelo) sono selezionate tenendo conto dell'ordinamento e dei vincoli di precedenza e di risorsa. Le attività selezionate sono allora aggiunte allo schedule parziale.

Due elementi fondamentali definiscono dunque un algoritmo greedy:

- La regola di priorità, in base alla quale viene definita una permutazione delle attività $\sigma = (\sigma_1, \sigma_2, \ldots \sigma_n)$
- Schema di schedule: criterio con cui è assegnato l'istante iniziale alle attività selezionate

Classificazione delle regole di priorità

Le regole di priorità definiscono la permutazione σ, in base alla quale sono poi inserite nello schedule. Le regole di priorità possono far riferimento a uno o più differenti aspetti del problema: le attività stesse, la topologia del grafo AoN, il cammino critico, le risorse.

Le regole di priorità del primo tipo si basano sulle durate delle attività, indipendentemente dalla loro posizione nel grafo AoN. Le due più usate sono:

- (SPT) Tempo di svolgimento più breve
- (LPT) Tempo di svolgimento più lungo

Le regole *basate sul grafo* fanno riferimento a proprietà strutturali del grafo AoN:

- (NIS) Numero di successori immediati - ha priorità maggiore l'attività avente un numero più alto di successori *immediati*
- (LNRA) Largest Number of Related Activities - per ogni attività si calcola un indice dato dal numero di attività raggiungibili da essa sul grafo; ha priorità maggiore l'attività avente indice più alto
- (GRPW) Greatest Ranked Positional Weight - è la versione pesata dell'indice precedente, ossia per ogni attività si calcola un peso dato dalla somma della propria durata e di quella di tutte le attività raggiungibili da essa sul grafo; ha priorità maggiore l'attività avente peso più alto. Indicando con $\mathcal{S}_j$ l'insieme di tutti in odi raggiungibili da j, l'indice è dunque dato da
$$d_j + \sum_{i \in \mathcal{S}_j} d_i$$

Gli ordinamenti basati sul cammino critico utilizzano invece informazioni ottenibili calcolando appunto earliest start/finish schedules e tempi di slittamento:

- (EST) Earliest Start Time - ha priorità maggiore l'attività avente minimo EST_i
- (EFT) Earliest Finish Time - come sopra, ma rispetto agli EFT_i
- (LST) Latest start time - come sopra, ma rispetto ai LST_i
- (MSLK) Minimum slack - ha priorità maggiore l'attività avente minimo total float.

Altre regole di priorità fanno riferimento all'impiego di risorse:

- (RD) Resorce/Duration - ha priorità maggiore l'attività avente massimo prodotto della durata d_i per la richiesta di risorsa r_i.

- (GRD) Generalized RD - Come sopra, ma considera anche i successori immediati.

Alcune regole, più elaborate, fanno riferimento a informazioni diverse, eventualmente con pesi diversi. Tra queste menzioniamo la regola seguente (Ulusoy e Özdamar 1989), che a valle di un'ampia sperimentazione si è rivelata fornire permutazioni mediamente di buona qualità:

- (WRUP) Weighted Resource Utilization and Precedence - ha priorità maggiore l'attività avente massimo valore della quantità

$$0.7|\mathcal{S}_j| + 0.3r_j/a$$

che, come si può osservare, premia le attività aventi un maggior numero di successori (anche non immediati) e un maggior utilizzo di risorsa.

Infine, accenniamo al fatto che può essere utile (specialmente quando sia necessario generare più ordinamenti, come nel caso degli algoritmi genetici, §**??**) generare σ in modo casuale: anziché fare in modo che ogni ordinamento sia equiprobabile, si può orientare la generazione della permutazione tenendo conto di uno o più degli indici di priorità menzionati.

La coppia regola-di-priorità/metodo-di-scheduling definisce un possibile algoritmo greedy.

Detta σ la permutazione ottenuta dall'applicazione di una regola di priorità, indichiamo con σ_i l'elemento in posizione i, mentre $\sigma(v)$ è la posizione dell'attività v nella permutazione. Una utile definizione, per alcuni metodi di scheduling, è la seguente. Dato il grafo $G = (V, E)$ della rappresentazione AoN di un progetto, un ordinamento $\sigma = (\sigma_1, \sigma_2, \ldots \sigma_n)$ su V è detto *compatibile con* $G = (V, E)$ se $i \prec j$ in G implica $\sigma(i) < \sigma(j)$.

Alcune delle regole di priorità viste forniscono permutazioni compatibili.

LEMMA 6 *L'ordinamento per EST_i non decrescenti è compatibile.*

Dimostrazione Supponiamo il contrario. Allora esisterebbero $i, j \in V$ tali che $i \prec j$ e $\sigma(j) < \sigma(i)$. Ma $\sigma(j) < \sigma(i)$ implica $EST_j \leq EST_i$, mentre il fatto che i preceda j implica che $EST_j > EST_i$, contraddizione. □

Analogamente, si dimostra che gli ordinamenti per EFT_i, LFT_i o LST_i non decrescenti sono compatibili.

Greedy seriale

L'euristica *Greedy Seriale* parte da una permutazione σ compatibile con G. A ogni iterazione, l'euristica seleziona la prima attività non ancora schedulata. Il criterio di scheduling consiste nell'assegnare a questa attività l'istante iniziale minimo tale da non violare i vincoli di risorsa e di precedenze, tenendo conto delle attività già schedulate. Lo schema algoritmico della greedy seriale è illustrato in Figura 4.5.

ESEMPIO 20 *A partire dal progetto di Tabella 4.1 e rappresentato come AoN (Figura 4.6), in cui sono disponibili $a = 5$ unità di risorsa, si applichi l'euristica greedy seriale a partire dall'ordinamento compatibile $\sigma = (1, 2, 6, 5, 7, 4, 8, 3, 9)$ (ordinamento LFT non decrescenti).*

Input: Istanza di RCPSP
Output: Schedule
Inizializzazione: Scegli un ordinamento compatibile $\sigma = (\sigma_1, \sigma_2, \ldots \sigma_n)$
begin
  for $k = 1 \ldots n$
    Seleziona l'attività $j = \sigma_k$
    Poni s_j pari al minimo istante per cui $(s_{\sigma_1}, s_{\sigma_2}, \ldots, s_j)$ è ammissibile
  end
end

Figura 4.5: Greedy seriale

Attività	Predecessori	Durata	Risorse
1	-	0	0
2	1	1	1
3	1	2	2
4	1	4	2
5	1	3	2
6	2	1	2
7	6	5	1
8	5	3	2
9	3,4,7,8	0	0

Tabella 4.1: Dati progetto

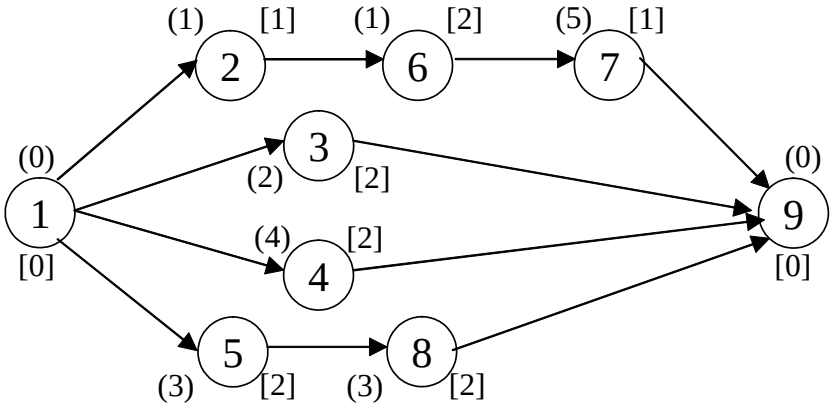

Figura 4.6: Rete AoN del progetto (durata)/[risorsa].

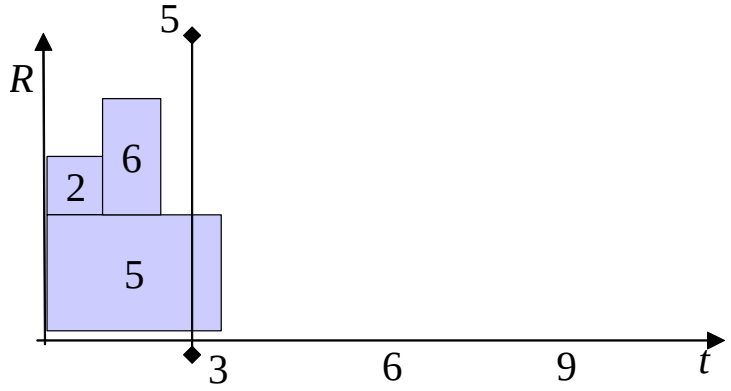

Figura 4.7: Fine Iterazione 4 del greedy seriale.

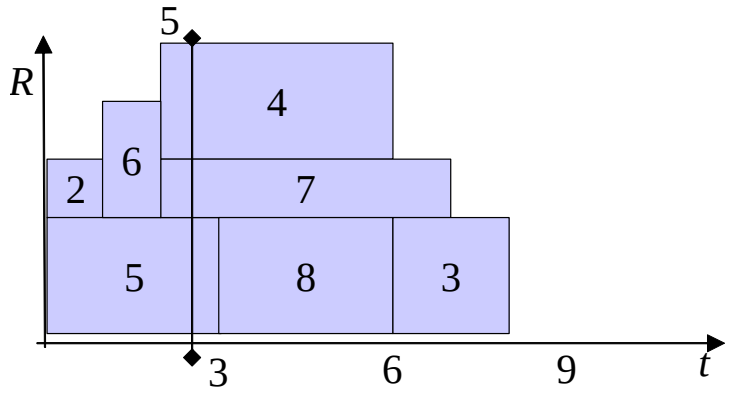

Figura 4.8: Diagramma di Gantt (finale) del greedy seriale.

Dopo aver saltato l'attività fittizia di inizio progetto, la greedy seriale sceglie l'attività 2, assegnandole l'istante di inizio 0. Successivamente, l'attività 6 dovrà cominciare quando l'attività 2 (vincolo di precedenza) è terminata, ovvero all'istante 1. Alla successiva iterazione, viene scelta l'attività 5, che non ha vincoli di precedenza con le altre attività e non viola il vincolo di risorsa, quindi può iniziare all'istante 0. Dopo la quarta iterazione il Gantt parziale costruito è mostrato in Figura 4.7. Le successive iterazioni aggiungono l'attività 7 all'istante 2, e l'attività 4 all'istante 2. Segue poi l'attività 8 schedulata a partire dall'istante 3, mentre l'ultima iterazione aggiunge all'istante 6 l'attività 3. Il diagramma di Gantt così ottenuto è mostrato in Figura 4.8, ed in Tabella 4.2 vengono mostrati gli istanti di inizio di tutte le operazioni.

□

Greedy parallelo

L'euristica *Greedy Parallelo* è leggermente più articolata. Sia V l'insieme delle attività e $W \subseteq V$ un qualunque sottoinsieme. Dato un ordinamento $\sigma = (\sigma_1, \sigma_2, \ldots \sigma_n)$ su V, si definisce un *ordinamento indotto* in W l'ordinamento $\sigma' = (\sigma'_1, \sigma'_2, \ldots \sigma'_{|W|})$ ottenuto da σ

Attività	Istante di inizio
1	0
2	0
3	6
4	2
5	0
6	1
7	2
8	3
9	8

Tabella 4.2: Start time ottenuti dal greedy seriale.

rimuovendo gli elementi non appartenenti a W.

L'algoritmo Greedy Parallelo effettua una scansione temporale in avanti, e a ogni passo, in generale, alloca *più* attività.

- Viene scelto un ordinamento σ delle attività, $t := 0$.
- Si considera l'insieme W_t di attività non ancora assegnate e tali che tutte le attività che le precedono sono già terminate all'istante t corrente.
- Si considerano le attività di W_t secondo l'ordinamento σ' indotto da σ, assegnando a ciascuna l'istante t come istante iniziale, laddove questo non violi il vincolo di risorsa.
- Si incrementa t al minimo istante di completamento di una delle attività correntemente in esecuzione, si calcola il nuovo W_t, e così via.

L'algoritmo Greedy Parallelo è mostrato in Figura 4.9, in cui R indica l'insieme delle attività non ancora schedulate. (Al solito, l'attività 1 è l'attività di inizio progetto mentre n è l'attività di fine progetto.)

Una differenza rilevante rispetto al Greedy Seriale è che, in generale, non è necessario che l'ordinamento σ sia compatibile. Infatti, per come è definito l'insieme W_t, l'unico controllo che è necessario effettuare riguarda il vincolo sulla risorsa.

Esempio 21 *Riprendiamo il progetto dell'esempio 20. Si applichi stavolta il greedy parallelo, a partire dallo stesso ordinamento* $\sigma = (1, 2, 6, 5, 7, 4, 8, 3, 9)$.

All'istante $t = 0$ *l'insieme delle attività disponibili è* $W_1 = \{2, 3, 4, 5\}$*, il loro ordinamento (all'interno di* σ*) è* $\sigma' = (2, 5, 4, 3)$*. L'attività 2 può cominciare subito, così come l'attività 5 ed anche l'attività 4, mentre l'attività 3 non può cominciare all'istante 0 perché verrebbe superato il vincolo sulle risorse disponibili. A questo punto il tempo viene incrementato al valore* $t = 1$*, coincidente con il termine della prima attività schedulata. All'istante* $t = 1$ *c'è una sola unità di risorsa disponibile e l'insieme delle attività disponibili è* $W_1 = \{3, 6\}$*, il loro ordinamento è* $\sigma' = (6, 3)$*, ma entrambe richiedono due unità di risorsa per essere processate. L'algoritmo quindi non schedula nessuna attività e il tempo viene incrementato al valore* $t = 3$*, allorché, terminando l'attività 5, si rendono disponibili*

Input: Istanza di RCPSP
Output: Schedule
Inizializzazione: Scegli $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ con $\sigma_1 = 1$ e $\sigma_n = n$. $R = V$. $t = 0$.
begin
  while R non è vuoto
    Costruisci l'insieme $W_t \subseteq R$ delle attività che possono cominciare in t
    if $W_t \neq \emptyset$
    then
      Sia σ' indotto da σ in W_t
      for $q = 1$ to $|W_t|$
        Poni $j = \sigma'_q$
        if $s_j = t$ è (risorsa) compatibile con gli assegnamenti precedenti
          Poni $s_j = t$ e $R = R - \{j\}$.
      end
    end
    Poni t pari al minimo tempo di completamento di un'attività in esecuzione
  end
end .

Figura 4.9: Schema dell'algoritmo greedy parallelo.

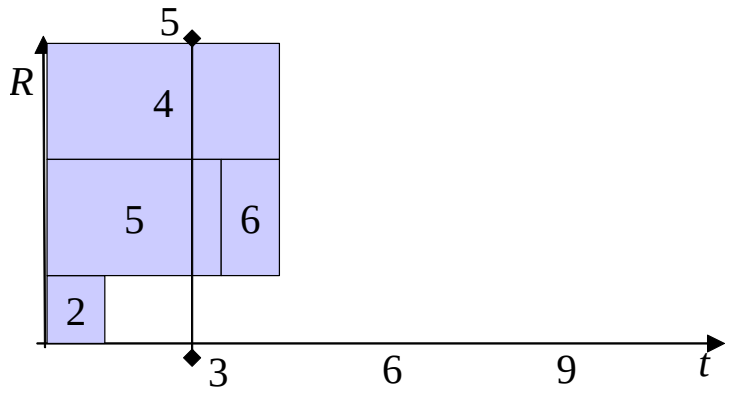

Figura 4.10: Fine Iterazione 3 del greedy parallelo.

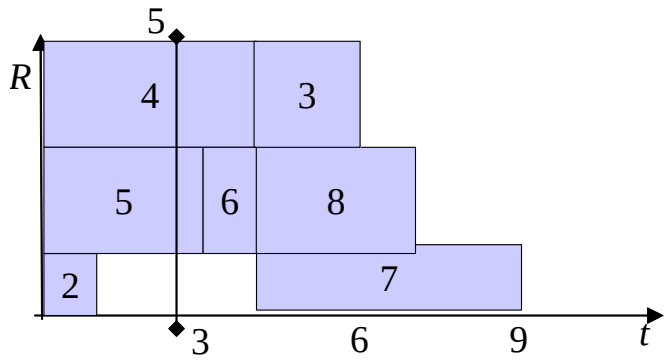

Figura 4.11: Diagramma di Gantt (finale) del greedy parallelo.

Attività	Istante di inizio
1	0
2	0
3	4
4	0
5	0
6	3
7	4
8	4
9	9

Tabella 4.3: Start time ottenuti dalla greedy parallela.

3 unità di risorsa. Si ha ora $W_3 = \{3, 6, 8\}$ *con* $\sigma' = (6, 8, 3)$. *Si aggiunge l'attività 6 allo schedule, ma non le attività 8 e 3. Il diagramma di Gantt parziale, al termine di questa iterazione, è mostrato in Figura 4.10. La successiva iterazione comincia con* $t = 4$ *e tutte e 5 le unità di risorsa disponibili. L'insieme delle attività disponibili per la schedulazione è* $W_4 = \{3, 7, 8\}$, *il loro ordinamento è* $\sigma' = (7, 8, 3)$. *E' possibile iniziare le attività 7, 8 e 3 senza eccedere il vincolo sulle risorse disponibili. L'algoritmo, avendo schedulato tutte le attività, termina. Il diagramma di Gantt così ottenuto è mostrato in Figura 4.11, ed in Tabella 4.3 vengono mostrati gli istanti di inizio di tutte le operazioni.*

□

4.5 Euristiche migliorative (ricerca locale)

Una vasta classe di algoritmi euristici si basa sul concetto di *ricerca locale* (*LS*, local search). L'idea di questi algoritmi è quella di migliorare iterativamente una soluzione data percorrendo una *traiettoria* nello spazio delle soluzioni.

Gli algoritmi di ricerca locale si basano sul concetto di *vicinato* o *intorno*. Dato un problema di ottimizzazione, il suo insieme ammissibile S, e una soluzione ammissibile x,

viene definito un opportuno insieme di soluzioni $I(x) \subseteq S$ detto *intorno di* x. Tipicamente, l'intorno viene definito mediante una procedura (*mossa*) che modifica parzialmente la soluzione x. Ovvero, l'intorno viene definito come tutte le soluzioni che possono essere ottenute da x applicando la mossa.

Il vicinato deve essere esplorato in tempi brevi per permettere l'eventuale esecuzione di un elevato numero di iterazioni dell'algoritmo di LS. Due diverse strategie di esplorazione sono comunemente sfruttate:

- (*Best improvement.*) A ogni iterazione, la local search cerca la soluzione ottima in un insieme ristretto di soluzioni ammissibili (l'intorno). Per fare questo tipicamente si ricorre ad un approccio enumerativo, ovvero in pratica l'intorno è definito in modo da essere sufficientemente piccolo da trovare la migliore soluzione in tempi brevi semplicemente enumerando (e verificando il valore di) tutte le soluzioni dell'intorno.

- (*First improvement.*) A ogni iterazione, la local search si limita a esplorare il vicinato solo fino a che non si trova una soluzione migliore della soluzione corrente. Questa strategia è ovviamente più veloce rispetto alla precedente, ma di solito porta a soluzioni di qualità peggiore.

4.5.1 Intorni per RCPSP

Nel definire un approccio di ricerca locale per RCPSP, occorre definire l'intorno (ovvero, una mossa) di uno schedule ammissibile. Per fare questo è necessario avere un'opportuna rappresentazione della soluzione e definire la mossa in questa rappresentazione. Ad esempio, si consideri l'algoritmo Greedy Parallelo. Come abbiamo visto, data una permutazione σ, l'algoritmo genera univocamente uno schedule ammissibile, sia esso $S(\sigma)$. Dunque, la permutazione σ può essere a tutti gli effetti considerata una rappresentazione di uno schedule. (In effetti, è una rappresentazione perfino più "ricca" degli schedule stessi, nel senso che diverse permutazioni possono corrispondere allo stesso schedule.) Cambiando la permutazione, si otterrà in generale uno schedule diverso. Quindi, modificando localmente una permutazione σ in molti modi diversi, si genera l'intero intorno $I(\sigma)$, e dunque una serie di schedule diversi, ma "vicini" a $S(\sigma)$.

Un esempio di intorno basato su questa rappresentazione è ad esempio l'intorno *insertion*: *data* σ, $I(\sigma)$ *è l'insieme di tutte le permutazioni ottenute da* σ *spostando un singolo elemento in una nuova posizione.*

Uno schema di massima di un algoritmo di Ricerca Locale per il RCPSP è mostrato in Figura 4.12.

Nel caso raffigurato, l'algoritmo si arresta non appena l'esplorazione dell'intorno non ha prodotto alcuno schedule migliore di quello corrente.

Chiaramente, il limite principale di questo approccio sta nel fatto che non appena si incontra una soluzione tale da non avere, nel proprio intorno, soluzioni migliori (ossia, un "minimo locale"), l'algoritmo termina. Ovviamente, se l'intorno è definito in modo da contenere un gran numero di soluzioni, la qualità della soluzione restituita potrà essere accettabile, tuttavia potrebbe diventare troppo onerosa l'esplorazione.

Sono stati allora progettati approcci metaeuristici, come tabu search e simulated annealing, che adottano specifiche misure per contrastare questo fenomeno.

Input: Soluzione σ
Output: Soluzione σ migliorata
begin
  σ := permutazione iniziale; *stop*:=*false*
  while (*stop*=*false*)
    Costruisci l'intorno $I(\sigma)$
    Applicando il greedy parallelo, calcola il makespan $z(\tilde{\sigma})$ per ciascuna permutazione $\tilde{\sigma} \in I(\sigma)$
    Trova la permutazione $\sigma^* \in I(\sigma)$ cui corrisponde il minimo $z(\tilde{\sigma})$
    if $z(\sigma^*) < z(\sigma)$, **then** $\sigma := \sigma^*$
    else *stop:=true*
  end
end

Figura 4.12: Schema di Local Search per RCPSP.

Un altro problema è che, adottando una determinata rappresentazione delle soluzioni (come ad esempio la permutazione σ nel caso del RCPSP), alcune delle soluzioni incontrate nell'esplorazione dell'intorno possono essere non ammissibili. Nel caso in figura 4.12, a qualunque permutazione corrisponde uno schedule ammissibile, magari di scarsa qualità. Ma usando lo schema di greedy seriale invece di quello parallelo, non necessariamente tutte le mosse portano a permutazioni compatibili. Questo può riflettersi in una perdita di efficienza dell'algoritmo, che può perdere tempo a generare permutazioni inutili ai fini dell'ottimizzazione.

Un approccio metaeuristico notevolmente efficace è quello degli *algoritmi genetici*.

4.5.2 Algoritmi genetici per RCPSP

Gli algoritmi genetici costituiscono una categoria di metauristiche basata sull'analogia tra individui di una popolazione e soluzioni ammissibili di un problema di ottimizzazione. L'idea di fondo è quella di lavorare su un *insieme* di soluzioni (anziché una singola soluzione), che costituiscono una *popolazione* di soluzioni. Queste, combinandosi tra loro, danno vita a nuove soluzioni, delle quali le uniche a sopravvivere saranno quelle più "adatte" (avente maggiore *fitness*), ossia, nel nostro caso, quelle aventi miglior valore della funzione obiettivo. Le soluzioni che sopravvivono alla "selezione" si combineranno ancora, dando vita alla successiva generazione di soluzioni, e così via. L'idea è che, rappresentando le soluzioni in modo opportuno, vale il principio che soluzioni buone sono costituite da un buon "materiale genetico", e combinando queste tra loro si ha maggiore probabilità di generare altre soluzioni buone.

Questa idea di fondo può realizzarsi in molti modi diversi, dando vita a diversi algoritmi. Tra gli aspetti caratterizzanti di un algoritmo genetico vi sono comunque:

- La rappresentazione delle soluzioni, ossia gli individui della popolazione
- La cardinalità della popolazione corrente

- Il meccanismo con cui due individui si combinano per dare luogo a nuovi individui (*crossover*)
- Il meccanismo con cui un individuo subisce un cambiamento del proprio "patrimonio genetico"(*mutazione*)
- Il numero di generazioni considerate prima del termine dell'algoritmo.

Uno schema di massima di un algoritmo genetico è riportato in Figura 4.13.

La validità di un algoritmo genetico dipende dalle scelte effettuate ed è difficilmente prevedibile a priori. In genere, la messa a punto di un algoritmo genetico richiede un'attività di sperimentazione molto ampia, al termine della quale comunque è possibile avere uno strumento talora molto efficace. Nel caso del RCPSP, gli algoritmi genetici sono stati oggetto di notevoli studi. È possibile accennare ad alcuni aspetti realizzativi.

Per quanto riguarda gli individui della popolazione, essi sono permutazioni σ delle attività: come abbiamo visto, l'adozione di un prefissato schema di generazione di schedule, garantisce che a una certa permutazione corrisponde una e una sola soluzione. Ciascun elemento della permutazione, secondo la terminologia corrente, prende il nome di *gene*.

L'aspetto forse più caratterizzante di un algoritmo genetico è l'operatore di crossover. Nel caso del RCPSP, descriviamo nel seguito un possibile operatore. Si considerino due individui, chiamiamoli Madre e Padre, ossia due permutazioni:

$$\sigma^M = (\sigma_1^M, \sigma_2^M, \ldots, \sigma_n^M) \qquad \sigma^P = (\sigma_1^P, \sigma_2^P, \ldots, \sigma_n^P)$$

da questi due, è possibile generare due individui figli, chiamiamoli D e S, nel seguente modo. Viene generato in modo casuale, con probabilità uniforme, un numero q compreso tra 1 e $n-1$. Per quanto riguarda D, esso ha in comune i primi q geni con la Madre, mentre gli altri $n-q-1$ geni sono ordinati allo stesso modo in cui compaiono nel Padre. Simmetricamente, S eredita i primi q geni dal Padre e gli altri sono presi nell'ordine in cui compaiono nella Madre. Ad esempio:

$$\sigma^M = (1, 3, 4, 2, 5, 6) \qquad \sigma^P = (2, 4, 1, 6, 5, 3)$$

se ad esempio $q = 2$, si ha

$$\sigma^D = (1, 3, 2, 4, 6, 5) \qquad \sigma^S = (2, 4, 1, 3, 5, 6)$$

Una proprietà interessante di questo operatore di crossover è che, se σ^M e σ^P sono permutazioni compatibili, anche σ^D e σ^S lo sono: dunque, partendo da una popolazione di permutazioni compatibili, tale proprietà è preservata – ciò è importante se ad esempio il meccanismo di associazione dello schedule alla permutazione è effettuato dal greedy seriale.

Si noti che l'operatore di crossover è un operatore binario, in quanto genera nuove soluzioni a partire da due soluzioni. L'operatore di mutazione è invece un operatore unario. Il meccanismo più utilizzato per introdurre una mutazione è il seguente. Data una permutazione $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$, e considerata ciascuna coppia di geni adiacenti (σ_i, σ_{i+1}), questi vengono scambiati con una certa probabilità p_m, che è un parametro

Input: $\mathcal{POP}$, GEN
Output: Insieme di soluzioni migliorata
begin
  genera una popolazione iniziale $\mathcal{POP}$ di soluzioni
  per ciascun elemento di $\mathcal{POP}$, calcola il valore di fitness
  $g := 0$
  while $(g < GEN)$
    $g := g + 1$;
    genera un nuovo insieme $\mathcal{CHI}$ di soluzioni da $\mathcal{POP}$ per mezzo del crossover
    applica l'operatore mutazione agli elementi di $\mathcal{CHI}$
    calcola la fitness di $\mathcal{POP} \cup \mathcal{CHI}$
    seleziona i $|\mathcal{POP}|$ individui migliori di $\mathcal{POP} \cup \mathcal{CHI}$
  end
end

Figura 4.13: Schema di algoritmo genetico.

dell'algoritmo. Se il meccanismo di schedulazione è il greedy seriale, lo scambio viene accettato solo se la nuova permutazione continua a essere compatibile.

A parte l'analogia con ciò che avviene in natura, l'importanza del meccanismo di mutazione è che rende possibile generare soluzioni che non sarebbero state mai raggiungibili con il solo meccanismo di crossover.

Quelli visti sono solo esempi di realizzazione di un algoritmo genetico per RCPSP; in letteratura esistono molti altri modi di definire la rappresentazione delle soluzioni, il crossover, la mutazione. Per un survey molto dettagliato si rimanda a Hartmann (1999).

Riferimenti

Damiani, M., Lo Valvo, P.P., Pipitone, I., 2004, *Le dimensioni del Project Management: Organizzazione, metodi, relazioni*, Edizioni Il Sole 24 Ore.

Demeulemeester, E.L., Herroelen, W.S., 2001, *Project scheduling, a research handbook*, Kluwer Academic Publishers.

Gagliardi, P., 2000, Forme organizzative emergenti, *XXII Congresso AIDP*, Bologna.

Hartmann, S., 1999, *Project Scheduling under Limited Resources*, Lecture Notes in Economics and Mathematical Systems, 478, Springer-Verlag.

Klein, R., 1999, *Scheduling of resource-constrained projects*, Kluwer Academic Publishers.

Morris, P.W.G., 1988, Managing Project Interfaces: Key Points for Projects Success, in Cleland and King (eds.), Project Management Handbook, Prentice-Hall, Englewood Cliffs, NJ.

Muench, D., et al., 1994, *The Sybase Development Framework*, Oakland, CA, Sybase Inc.

Murphy, P.L., 1989, Pharmaceutical Project Management: is it different?, *Project Management Journal*.

Project Management Institute, 2000, *A Guide to the Project Management Body of Knowledge*, 2000 Edition.

Ulusoy, G., Özdamar, L., 1989, Heuristic performance and network/resource characteristics in resource-constrained project scheduling, *Journal of the Operational Research Society*, 40, 1145–1152.