



UNIVERSITÀ DEGLI STUDI DI MILANO

Facoltà di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea in Informatica

Tesi di Laurea

ANALISI DELLA MODULARITÀ IN SOFTWARE COMPLESSI

Laureando

STEFANO LORENZI

Nr. Matricola 653819

Relatore

Dott. Mattia Monga

Anno accademico 2007-2008

A mio papà che con il suo Commodore 64
mi ha trasmesso la passione per l'informatica

A Monica per la pazienza
a sopportarmi e sostenermi

Al piccolo Leonardo che riesca a realizzare
i suoi sogni senza aspettare l'ultimo treno

La scienza è conoscenza organizzata.
La saggezza è vita organizzata.

Immanuel Kant

Indice generale

1 INTRODUZIONE.....	5
2 LA PROGRAMAZIONE MODULARE.....	7
2.1La programmazione modulare: caratteristiche e punti di forza.....	7
2.2La prima modularizzazione.....	8
2.3La seconda modularizzazione.....	10
3 IL DESIGN STRUCTURE MATRIX.....	14
3.1Matrici e grafi: definizioni e comparazione di efficacia.....	14
3.2Design Structure Matrix.....	16
3.3Costruire una DSM.....	21
3.4La partizione.....	21
3.5Indipendenza condizionata.....	24
4 GRAFI.....	27
4.1Reti Bayesiane.....	30
4.2Definizione di Markov Compatibility.....	34
4.3Il criterio della d-Separation.....	34
5 ANALISI DELLE DIPENDENZE.....	39
6 BIBLIOGRAFIA.....	43

1 INTRODUZIONE

La realizzazione di sistemi software complessi richiede che il prodotto finale venga strutturato in componenti modulari che incapsulano le maggiori decisioni di progetto. Ciò permette di ottenere sistemi relativamente flessibili in cui le parti possono evolversi senza che occorra necessariamente ristrutturare l'intero prodotto. Il principio dell'information hiding suggerisce di ridurre al minimo (nascondere) l'informazione che è necessario conoscere per modificare una decisione progettuale.

Le decisioni progettuali non sono però completamente indipendenti, spesso una scelta ne impone necessariamente delle altre. Questo problema, ben noto nella letteratura dell'ingegneria industriale, è stato spesso studiato considerando le cosiddette matrici della struttura progettuale (Design Structure Matrix, DSM) che descrivono le dipendenze fra le decisioni progettuali.

Grazie alle DSM è possibile ragionare sulle dipendenze dirette fra i moduli di un sistema, ricavandone importanti informazioni che permettono di prevedere i possibili effetti dei cambiamenti.

Le decisioni dei progettisti sono però spesso più complesse di quanto sia possibile dedurre dalla semplice analisi dell'impatto di un cambiamento. In effetti più in generale la corretta evoluzione di un componente X può dipendere dalla conoscenza di come si è evoluto il componente Y, ma tale informazione può diventare irrilevante qualora sia nota l'evoluzione di un insieme di componenti Z, i cui cambiamenti "schermano" i cambiamenti di Y.

Per facilitare questo tipo di ragionamenti è più opportuno considerare la DSM un grafo diretto fra i nodi delle decisioni progettuali e applicare la nozione di d-separation (Pearl, 1988): un insieme di nodi Z d-separa due altri nodi X e Y se blocca ogni percorso fra X e Y.

In questa tesi si è iniziato ad esplorare l'utilità dell'uso della d-separation nell'analisi della struttura progettuale di un sistema software complesso. È stato costruito un prototipo che permette di analizzare i grafi ricavabili dalle DSM e identificare le d-separazioni fra due decisioni progettuali. Con l'aiuto di questo strumento sono state poi analizzate due diverse soluzioni descritte da Parnas [Parnas, 1972a] per la realizzazione di un sistema per la produzione di indici analitici (KWIC), la cui struttura

progettuale e` gia` stata studiata in letteratura. E` stato cosi` possibile stabilire che, qualora si conosca (o si controlli) l'evoluzione di determinati componenti, l'evoluzione di altri risulta irrilevante.

2 LA PROGRAMMAZIONE MODULARE

2.1 *La programmazione modulare: caratteristiche e punti di forza*

Il concetto di modularità è fondamentale nello sviluppo del software moderno e rappresenta un'importante svolta per la scienza della programmazione.

La programmazione modulare si basa sul concetto di modulo. Un modulo software è una unità le cui dipendenze rispetto al resto del sistema sono esplicite e ridotte.

Ciò permette di organizzare gruppi di lavoro in modo che i moduli vengono sviluppati senza inutili scambi di informazione. Anche la flessibilità del sistema aumenta perché è possibile prevedere l'impatto dei cambiamenti nelle scelte implementative.

Un grande contributo alla programmazione modulare arriva da D.L. Parnas.

Nel 1972 Parnas introdusse l'Information Hiding come approccio per costruire strutture modulari per il progetto, al fine di migliorare completamente l'adattabilità del software. L'idea di Parnas è di scorporare le decisioni del Design che tendono a cambiare, in modo che esse possano cambiare indipendentemente.

La formulazione di Parnas ha contribuito allo sviluppo di linguaggi di programmazione di tipo "data type", e alla programmazione "Object-Oriented".

A questa conclusione Parnas era arrivato con uno studio nel quale comparava la capacità di un software di essere modificato (Parnas 1972a), sperimentandone due modalità di modularizzazione dello stesso programma, il KWIC (*Key Words in Context*), un programma per computare gli indici permutati. Questi indici sono usati ampiamente per redigere cataloghi di biblioteche; taluni indici analitici di libri si avvicinano agli indici KWIC. Il termine indice KWIC è stato introdotto quando si sono prodotti tali indici con il computer.

Consideriamo una locuzione utile per un indice, ad es. la locuzione:

Elenco di poligoni, poliedri e politopi

Accanto ad essa possono essere utili le varianti ottenute sottoponendola a permutazioni circolari limitatamente a quelle che cominciano con parole significative per chi utilizzerà l'indice. Nel caso preso ad esempio sono interessanti le

permutazioni circolari:

*poligoni, poliedri e politopi. * Elenco di
poliedri e politopi. * Elenco di poligoni,
politopi. * Elenco di poligoni, poliedri e*

Nel suo lavoro, Parnas procede presentando un'analisi comparativa della flessibilità di due Design, calcolando il numero di moduli che devono essere ridisegnati per ogni cambiamento da apportare nel sistema. Vengono così introdotte due differenti procedure di modularizzazione.

2.2 La prima modularizzazione

Nella prima modularizzazione Parnas suddivide il sistema secondo il flusso della computazione e descrive cinque moduli, ognuno con funzioni differenti:

1. Input
 - Questo modulo legge i dati di input medio e crea le strutture dati opportune
2. Circular shift
 - Questo modulo è chiamato dopo che il modulo di input ha terminato il suo lavoro, vengono calcolate tutte permutazioni circolari (non ordinati) e viene generato un indice che punta al primo carattere di ogni singola permutazione, il primo indice punta alla frase originale
3. Alphabetizing
 - Questo modulo riceve in input gli array prodotti dai primi 2 moduli, successivamente verranno prodotti dei vettori dello stesso formato che contengono le permutazioni calcolate nel secondo modulo ma in questo caso i vettori saranno ordinati alfabeticamente
4. Output
 - Usando gli array prodotti dal primo e dal terzo modulo stampa le permutazioni
5. Master Control
 - Questo modulo si occupa di chiamare i singoli moduli, inoltre gestisce

eventuali messaggi di errore.

Ad esclusione del Master, Parnas suddivide ulteriormente ogni modulo in 3 parti:

1. Type - le strutture dati esportate
2. Data - le dichiarazioni delle procedure
3. Alg - gli algoritmi

Procedendo in questo modo, si ottengono in totale 13 componenti software.

Di seguito andiamo a definirne i ruoli di ogni singolo modulo di tale DSM.

- A (Input Type), D (Circ Type) , G (Alph Type) e J (Out Type) modellano le interfacce delle procedure esportate, come regole del design, per determinare successivamente gli algoritmi dell'input, lo shift, il sort e l'output.
- B (Input Data), E (Circ Data), H (Alph Data) e K (Output Data) modellano le strutture dati in quanto regole del design . Parnas constata che l'accordo tra di essi deve essere stabilita prima dell'inizio dell'implementazione di moduli indipendenti.
- C (Input Alg), F (Circ Alg), I (Alph Alg), L (Output Alg) e M (Master) modellano i parametri che sono slegati: la scelta di algoritmi per manipolare le strutture dati fissate.

Al termine di tale lavoro, Parnas conclude: *“I documenti definitivi includeranno un numero di figure rappresentanti il format centrale, convenzioni per l'indicazione, convenzioni, ect. Tutto ciò che riguarda le interfacce tra i quattro moduli dev'essere specificato prima che si inizi a lavorare... Questa è una modularizzazione nel senso indicato da tutti i proponenti dei programmi di modularizzazione. Il sistema è diviso in un numero di moduli con interfacce ben definite; ognuna è sufficientemente piccola e semplice da essere compresa ampiamente e ben programmata.”* (Parnas 1972a)

Nella figura sottostante viene rappresentato la Design Structure Matrix (DSM – che approfondiremo nel capitolo 2) del software di Parnas, sviluppato secondo la prima modularizzazione. In tale rappresentazione è possibile vedere le dipendenze dirette dei moduli (ad esempio H dipende da E).

	A	D	G	J	B	E	H	K	C	F	I	L	M
A - Input Type													
D - Circ type													
G - Alph type													
J - Out Type													
B - In Data						X	X						
E - Circ Data					X		X						
H - Alph Data					X	X							
K - Out Data													
C - Input Alg	X				X								
F - Circ Alg		X			X	X							
I - Alph Alg			X		X	X	X						
L - Out Alg				X	X		X	X					
M - Master	X	X	X	X									

Figura 2.1 - DSM (Design Structure Matrix)

Il DSM immediatamente rivela le proprietà chiave del design. Primo, il design è una modularizzazione, come sostiene Parnas: i disegni sviluppano le loro parti indipendentemente come rivelato dall'assenza di segni nel riquadro in basso a destra nel DSM. Secondo, solo una piccola parte – gli algoritmi – è nascosta e liberamente intercambiabile.

Terzo, gli algoritmi sono strettamente legati alle regole della struttura dati del design. Inoltre, le strutture dati sono un nodo indipendente – nel quadrante in alto a sinistra.

2.3 La seconda modularizzazione

Una modularità più flessibile è invece basata sul principio dell'Information Hiding, ossia usando interfacce di tipi di dati estrapolati per scomporre decisioni basilari del Design, coinvolgendo scelte di strutture dati e algoritmi che possono cambiare senza effetti ondulatori eccessivamente costosi.

In questo caso, Parnas propone i seguenti moduli:

1. Modulo testo

- Memorizza le linee di testo
- Fornisce funzioni per accedere e modificare parole in ogni linea
- 2. Modulo Input
 - Legge il file di testo e chiama le funzioni definite nel modulo testo per memorizzarlo
- 3. Modulo Scorrimento circolare
 - Funzioni e strutture per trattare lo scorrimento circolare
- 4. Modulo Ordinamento
 - Ordina le permutazioni ottenute
- 5. Modulo Output
 - Stampa le permutazioni ottenute usando i moduli precedenti
- 6. Programma principale
 - Simile al Master della prima modularizzazione, ma con meno strutture globali

Possiamo notare che rispetto alla prima modularizzazione è stato aggiunto un modulo (il primo), mentre gli altri moduli concettualmente rimangono gli stessi.

Anche in questa nuova modularizzazione, Parnas suddivide ogni modulo in 3 parti, le strutture dati esportate (Type), le dichiarazioni delle procedure (Data) e gli algoritmi (Alg). Il Master non subisce, invece, tale suddivisione. In questo modo la seconda modularizzazione è fatta da 16 componenti software.

Il DSM della seconda modularizzazione è rappresentata nella figura 2.2.

Parnas conclude il suo articolo comparando le due modularizzazioni. Inizialmente egli sottolinea che la prima modularizzazione è quella tradizionale, mentre la seconda è stata usata con successo all'interno di molti progetti dei suoi studenti (Parnas 1972b).

Dalla sua successiva analisi emerge che entrambe le modularizzazioni rappresentano lo stesso prodotto, tanto che un buon compilatore potrebbe generare lo stesso file oggetto.

	N	A	D	G	J	O	P	B	C	E	F	H	I	K	L	M
N - Line Type																
A - In Type																
D - Circ type																
G - Alph type																
J - Out Type																
O - Line Data	X						X									
P - Line Alg	X					X										
B - In Data		X							X							
C - In Alg	X	X						X								
E - Circ Data	X		X							X						
F - Circ Alg	X		X							X						
H - Alph Data	X			X								X				
I - Alph Alg	X			X				X				X				
K - Out Data					X										X	
L - Out Alg	X				X									X		
M - Master	X	X	X	X	X											

Figura 2.2 - La seconda modularizzazione di Parnas

Le differenze tra le due modularizzazioni alternative consistono nel modo in cui sono stati suddivisi gli incarichi di lavoro e le interfacce tra i moduli. Gli algoritmi usati potrebbero, invece, essere identici, come anche l'ordinamento. Le due soluzioni sono sostanzialmente differenti nel design, anche se identici in fase di esecuzione.

Nel suo articolo Parnas sostiene che nella prima modularizzazione le interfacce tra i moduli sono abbastanza complesse, per via della gestione delle strutture dati e che quest'ultime rappresentano decisioni di progettazione che richiedono un'attenta e seria analisi prima di venir prese.

Le strutture dati sono essenziali per l'efficienza e devono, quindi, essere pensate con molta attenzione. Per tale motivo, si può dedurre che tali strutture siano identiche in entrambe le modularizzazioni.

Nella seconda modularizzazione le interfacce sono più astratte, e consistono principalmente nella dichiarazione del nome delle funzioni implementate e dei parametri che ricevono in input.

Terminata l'analisi comparativa delle due modularizzazioni, Parnas vede nella

seconda una migliore comprensibilità. Ciò dipende dal fatto che nella prima modularizzazione per capire il modulo di output è necessario capire qualcosa dei moduli precedenti (input, circular shift, sort), come ad esempio la gestione delle strutture dati a seconda degli algoritmi usati. Ne consegue che il sistema sarà comprensibile solo nella sua totalità.

Nella seconda modularizzazione, invece, le cose vanno diversamente. Essa, infatti, basandosi sull'Information Hiding risulta più comprensibile in quanto il Design è costituito da decisioni che tendono a cambiare e tale modalità di procedere permette un notevole risparmio nei termini di risorse e tempo.

Parnas conclude, quindi, che la modularizzazione Information Hiding è migliore e suggerisce ai designer di preferire l'utilizzo di un processo di Design basato sull'Information Hiding, rispetto a Design basati sulla prima modularizzazione.

Possiamo schematizzare la valutazione delle due modularizzazioni mettendo in evidenza le dipendenze che esse introducono.

Uno strumento molto utile per valutare le dipendenze sono le DSM, che andremo a presentare nel prossimo capitolo.

3 IL DESIGN STRUCTURE MATRIX

Una DSM è una rappresentazione matriciale di un progetto, particolarmente utile per analizzare e migliorare i processi di progettazione. La matrice contiene una lista di tutte le attività/moduli costituenti il progetto e la relativa struttura di scambi informativi. Essa esprime, quindi, quali informazioni sono richieste per avviare una certa attività e dove le informazioni generate da quel modulo vanno a finire. Inoltre, essa permette di evidenziare e determinare quali informazioni e quali conoscenze sono prodotte e trasferite nel processo di produzione.

Dati tali caratteristiche, la DSM fornisce indicazioni su come gestire un progetto complesso ed evidenzia i fabbisogni informativi, la sequenza delle attività e le iterazioni.

Le iterazioni sono una parte fondamentale della progettazione, in quanto migliorano la qualità del progetto. Non tutte le iterazioni sono però auspicabili, in quanto la ripetizione di un sotto-processo può causare ritardi e aumentare i costi. Le iterazioni danno un grosso apporto quando assicurano il raffinamento delle assunzioni fino al raggiungimento di una convergenza tra le varie decisioni. Quindi è fondamentale valutare se un processo iterativo è auspicabile o no e la DSM è in grado di dare indicazioni precise e di grande utilità a tale proposito.

3.1 Matrici e grafi: definizioni e comparazione di efficacia

Di seguito andiamo a definire le matrici.

Si consideri un sistema che sia composto da due moduli, modulo A e modulo B. La rappresentazione matriciale è una matrice quadrata, in quanto composta dallo stesso numero di righe e colonne. La forma della matrice è la seguente: i nomi dei moduli sono posti sul lato sinistro e superiore della matrice, nello stesso ordine come intestazione delle righe e delle colonne. Nella rappresentazione a matrice, è chiaro che gli elementi della diagonale non hanno alcun significato nel descrivere il sistema, in quanto rappresenterebbero la relazione di un elemento con se stesso e vengono

pertanto colorati di nero. Nella figura 3.1 è possibile vedere la rappresentazione matriciale sopra descritta.

	A	B
A		
B		

Figura 3.1 – Rappresentazione di una matrice

Esistono matrici binarie per la modellazione dei sistemi, le quali sono utili in quanto possono rappresentare la presenza o l'assenza di una relazione tra coppie di moduli del sistema.

Possiamo, invece, definire un grafo come un insieme di vertici (o nodi), ed un insieme di archi che connettono coppie di nodi. I vertici del grafo corrispondono a moduli e le frecce denotano la relazione esistente tra i moduli. Nella figura 3.2 è rappresentato un esempio di grafo.

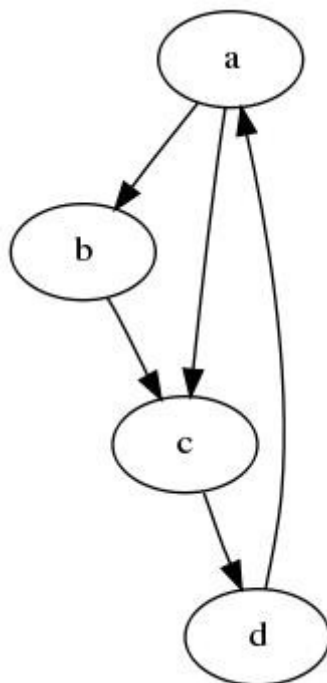


Figura 3.2 – Esempio di grafo

Un notevole vantaggio della rappresentazione a matrice rispetto ai grafi sta nella sua compattezza e nella sua capacità di fornire una mappatura sistematica degli elementi del sistema. Tale mappatura, infatti, risulta chiara e facile da leggere, indipendentemente dalla dimensione del sistema stesso. Inoltre, come avremo modo di spiegare e approfondire successivamente, in una matrice DSM le relazioni di feedback sono più facili da individuare che in un grafo.

Un limite del DSM consiste nel fatto che sono facilmente visibili solo le dipendenze dirette, mentre nel grafo, anche le dipendenze non dirette sono meglio visibili (ad esempio nella figura 3.2 è visibile che D dipende da A)

3.2 *Design Structure Matrix*

Se il sistema da rappresentare è un progetto composto da un insieme di attività, si può applicare l'approccio matriciale ricorrendo alla metodologia *Design Structure Matrix*. Righe e colonne sono intestate con la lista completa delle attività che devono essere realizzate. I simboli all'interno della matrice indicano se vi sono relazioni informative tra i vari moduli. I simboli di una stessa riga (della DSM) indicano tutte le attività i cui output sono necessari per realizzare l'attività corrispondente a quella riga. Allo stesso modo una specifica colonna indica quali attività ricevono informazioni dall'attività corrispondente a quella colonna. Se l'ordine delle attività nelle righe e nelle colonne indica una sequenza temporale, allora i simboli sotto la diagonale rappresentano trasferimenti di informazione in avanti, verso attività che verranno svolte successivamente.

Questa tipologia di simboli è chiamata "forward mark" o "forward information link". Con tale termine, quindi, definiamo i simboli sopra la diagonale che indicano ritorni di informazioni verso attività, le quali sono già state eseguite. In questa condizione, tali attività sono dipendenti da attività successive, e quindi "feedback mark".

La comprensione del comportamento dei singoli moduli ci consente di comprendere il comportamento del sistema. Per comprendere tale concetto, osserviamo la figura 3.3. In tale matrice, il modulo B si dice indipendente dal modulo A se non sono necessari scambi di informazione tra i due moduli, dipendente se viceversa.

	A	B
A	■	
B	X	■

Figura 3.3 – Esempio di DSM

In una configurazione sequenziale, i parametri dell'elemento B sono impostati sulla base dei parametri del modulo di A, quindi possiamo affermare che un modulo influenza il comportamento di un altro modulo in modo uni-direzionale.

Se prendiamo in considerazione una configurazione parallela, gli elementi non interagiscono tra di loro. In questo caso, l'attività B è detta indipendente dall'attività A e lo scambio di informazioni non è richiesto tra le due attività.

Inoltre, il flusso delle informazioni in un sistema interdipendente risulta interconnessa, vale a dire che il modulo A influenza il modulo B e il modulo B influenza il modulo A. Questa condizione ha luogo quando il parametro A non può essere determinato con certezza se non dopo aver conosciuto il parametro B e viceversa.

Questa dipendenza ciclica è chiamata “Circuito” o “Ciclo informativo”.

Nella figura 3.4 troviamo rappresentate graficamente le configurazioni appena descritte.

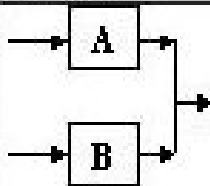
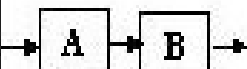
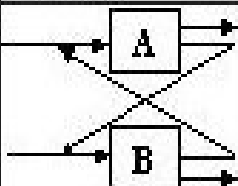
Three Configurations that Characterize a System																																								
Relationship	Parallel			Sequential		Coupled																																		
Graph Representation																																								
DSM Representation	<table><tr><td></td><td>A</td><td>B</td></tr><tr><td>A</td><td></td><td></td></tr><tr><td>B</td><td></td><td></td></tr></table>		A	B	A			B			<table><tr><td></td><td>A</td><td>B</td></tr><tr><td>A</td><td></td><td></td></tr><tr><td>B</td><td>X</td><td></td></tr></table>		A	B	A			B	X		<table><tr><td></td><td>A</td><td>B</td></tr><tr><td>A</td><td></td><td></td></tr><tr><td>B</td><td>X</td><td></td></tr></table>		A	B	A			B	X		<table><tr><td></td><td>A</td><td>B</td></tr><tr><td>A</td><td></td><td>X</td></tr><tr><td>B</td><td>X</td><td></td></tr></table>		A	B	A		X	B	X	
	A	B																																						
A																																								
B																																								
	A	B																																						
A																																								
B	X																																							
	A	B																																						
A																																								
B	X																																							
	A	B																																						
A		X																																						
B	X																																							

Figura 3.4 – Configurazione caratterizzanti un sistema

Un esempio di DSM viene mostrato in figura 3.5, figura che riporta la DSM di Parnas (con la prima modularizzazione). In questa figura è possibile vedere che B alimenta E, H,C,F,I,L mentre H è alimentato da B e E.

In altre parole, le marcature su una determinata colonna indicano quali moduli vengono alimentati dal modulo in questione (la colonna); mentre le marcature presenti su una determinata riga indicano da quali moduli è alimentato il modulo in questione (riga).

Inoltre, all'interno della matrice, le marcature simmetriche tra loro indicano la presenza di un ciclo. Nell'esempio della figura 3.5, B ed E rappresentano un ciclo, in quanto B dipende da E ed E dipende da B.

Tutti i simboli al di sopra della diagonale sono “feedback mark”, come abbiamo in precedenza descritto. Essi corrispondono agli input necessari che non sono disponibili al momento dell'esecuzione dell'attività. In questo caso, tale esecuzione si baserà su assunzioni relative allo status dell'attività che fornisce l'input. Man mano che il progetto si sviluppa, tali assunzioni vengono riviste alla luce delle nuove informazioni, e, se necessario, l'attività dipendente può anche essere rieseguita.

E' importante notare come le relazioni di feedback siano facilmente individuabili in una matrice DSM, diversamente che in un grafo. Ciò rende la DSM una rappresentazione potente e contemporaneamente semplice di un sistema o di un progetto complesso.

Come afferma Yassine (1999), è possibile eliminare o ridurre i “feedback mark” manipolando la matrice. Questo processo, chiamato partizione, verrà discusso in seguito.

Una volta fatto ciò, comincia ad emergere una struttura trasparente della rete, che consente una migliore programmazione del progetto.

Da tale struttura si può vedere quali attività sono sequenziali, e quali sono interdipendenti. Una volta che la DSM è stata ripartita, le attività in serie sono identificate ed eseguite in sequenza. Per quanto riguarda quelle interdipendenti si rende necessaria una precedente pianificazione.

Saremo ad esempio in grado di sviluppare un piano di iterazione grazie al quale determinare quali attività dovrebbero avviare il processo di iterazione sulla base di un'ipotesi iniziale o di una stima di quelli che saranno gli elementi informativi

mancanti.

Il blocco E-B-H della figura... potrebbe essere eseguito nella maniera seguente:

1. l'attività E parte con una ipotesi iniziale circa gli output di H,
2. l'output di E è passato all'attività di B,
3. l'output di B alimenta H,
4. l'output di H viene passato all'attività E.

A questo punto E è in grado di confrontare l'output di H con la sua ipotesi iniziale e decidere se è necessaria o meno una ulteriore iterazione in relazione a quanto l'ipotesi si discosta dalle informazioni realmente ricevute da H. Questo processo iterativo continua finché non si riscontra una convergenza tra output ipotetico ed output reale.

	A	D	G	J	B	E	H	K	C	F	I	L	M
A													
D													
G													
J													
B						X	X						
E					X		X						
H					X	X							
K													
C	X				X								
F		X			X	X							
I			X		X	X	X						
L				X	X		X	X					
M	X	X	X	X									

Figura 3.5 – Esempio di rappresentazione di una matrice DSM
tratta dalla prima modularizzazione del lavoro di Parnas

Nelle figure 3.5 e 3.6 è stato rappresentato lo stesso progetto attraverso le matrici DSM, nella figura 3.5, e con il grafo, nella figura 3.6. Si può facilmente osservare quanto affermato sopra, ossia la maggiore chiarezza nella DSM nel vedere le dipendenze dirette (in particolare su software complessi che utilizzano molti moduli),

mentre, utilizzando un grafo, con molti nodi, potrebbe essere meno chiaro a prima vista, soprattutto se il numero delle dipendenze è alto (in questo caso avremmo un numero di archi notevoli), ma il grafo, permette di vedere anche le dipendenze non dirette.

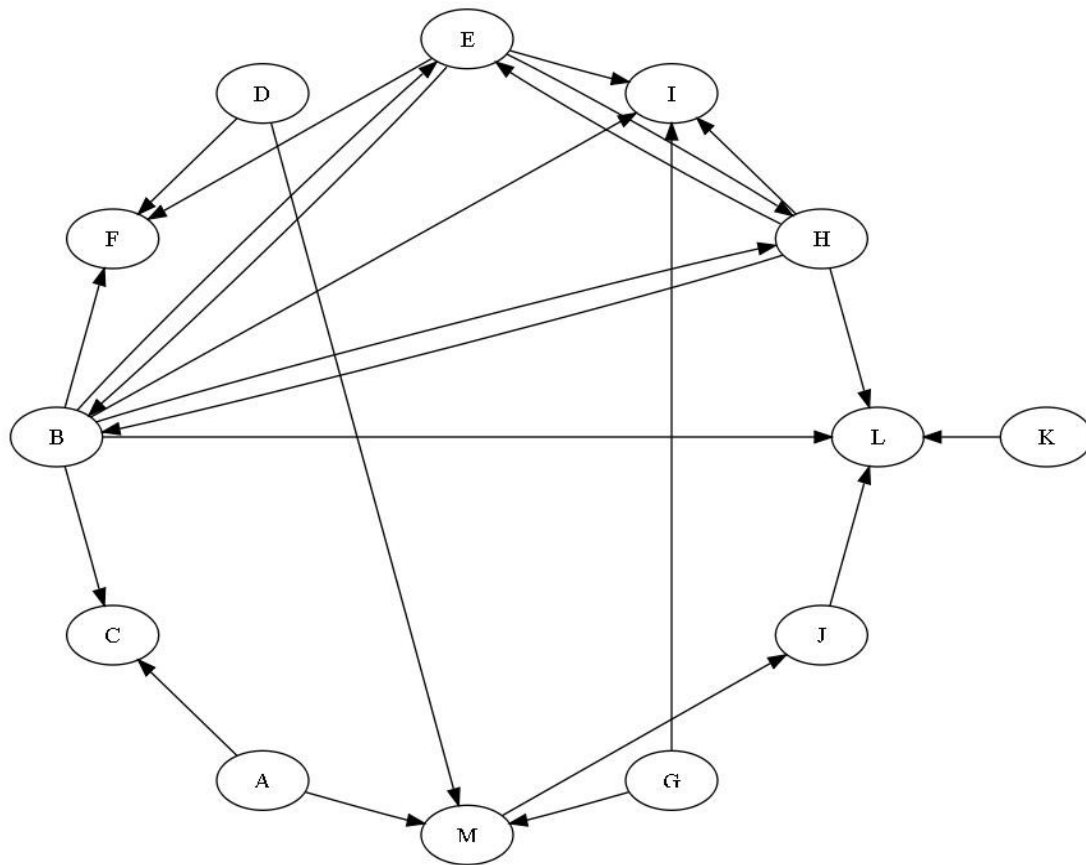


Figura 3.6 – Esempio di rappresentazione di un grafo tratta dalla prima modularizzazione del lavoro di Parnas

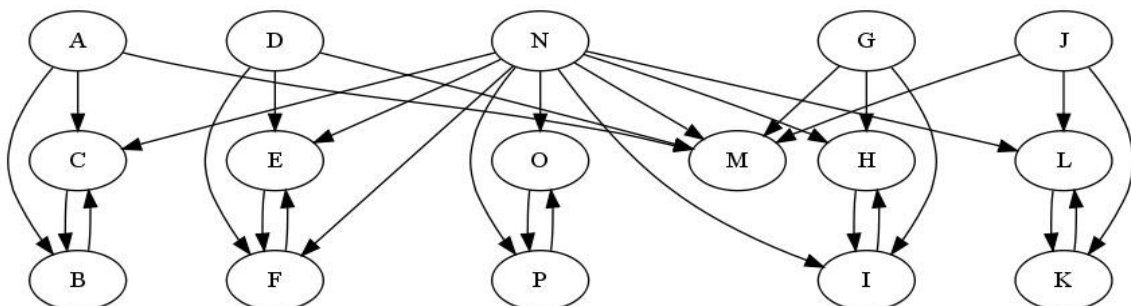


Figura 3.7 – Esempio di rappresentazione di un grafo tratta dalla seconda modularizzazione del lavoro di Parnas

3.3 Costruire una DSM

Per sfruttare al meglio le potenzialità della DSM è importante costruire una DSM in modo adeguato e corretto. Infatti, il metodo DSM risulta essere particolarmente efficace se, nel processo di costruzione, vengono adeguatamente realizzati due compiti: l'appropriata scomposizione del sistema e l'accuratezza delle relazioni di dipendenza. Se ciò non avvenisse in modo corretto, arriveremmo ad ottenere una DSM sbagliata e quindi inutile, se non deleteria ai fini lavorativi.

Entreremo ora nella spiegazione del processo di costruzione della DSM, delineando le tappe fondamentali di tale percorso.

1. La fase preliminare imprescindibile nella costruzione di una DSM è studiare attentamente il sistema al fine di scomporlo adeguatamente in elementi significativi, ossia in moduli. Per portare a termine tale compito è possibile, ad esempio, analizzare la documentazione esistente o condurre interviste strutturate agli esperti. Condizione ottimale, sarebbe ricorrere ad entrambi approcci, uno consecutivo all'altro.
2. Una volta individuati tali moduli, essi vengono inseriti nelle intestazioni della matrice (righe e colonne) nello stesso ordine; quindi vengono inseriti i simboli all'interno della matrice (generalmente con una X) a seconda delle varie dipendenze tra i moduli. Tali simboli vengono scelti all'interno della lista, identificando gli elementi minimi necessari che influenzano i loro sotto-sistemi e il loro comportamento.
3. Al termine del processo di rappresentazione del sistema, è possibile cominciare a progettare nuovamente, usando criteri differenti, che in seguito mostreremo.

3.4 La partizione

Con il termine “partizione” definiamo il processo di manipolazione delle righe e delle colonne della DSM, ossia la ridefinizione dell'ordine delle stesse al fine di eliminare i “feedback mark”. Ciò permette di trasformare la DSM in una matrice triangolare

inferiore.

Poiché dalla semplice partizione è assai difficile eliminare tutti i feedback mark, più frequentemente tale manipolazione si pone come obiettivo di spostare i feedback mark il più vicino possibile alla diagonale. Questo lavoro permetterà di rendere il processo di sviluppo più veloce in quanto un minor numero di elementi verrà coinvolto nei cicli interattivi.

Esistono approcci differenti, seppur simili tra loro, alla partizione della DSM. Tali approcci si differenziano per l'identificazione differente che essi fanno dei cicli di informazione.

Di seguito riportiamo le fasi degli algoritmi per la partizione e rappresentata graficamente nella figura 3.8:

1. identificazione degli elementi del sistema che possono essere determinati senza output da parte degli elementi della matrice. Ciò avviene osservando le righe vuote della DSM e collocandoli in cima alla stessa. Questa operazione viene eseguita per ogni elemento e ogni elemento spostato, viene eliminato dalla DSM.
2. Identificazione degli elementi del sistema che non forniscono informazioni agli altri elementi della matrice. In questo caso, si identificano tali elementi osservando le colonne e ponendoli in fondo alla DSM. Anche in questo caso, l'operazione viene eseguita per ogni elemento e ogni elemento spostato, viene eliminato dalla DSM.
3. Al termine delle fasi precedenti, la matrice è completamente ripartita se nella DSM non rimangono elementi. Nel caso opposto, gli elementi rimanenti contengono almeno un circuito informativo.
4. A questo punto, è necessario utilizzare uno dei seguenti metodi per identificare i circuiti: ricerca del percorso (un flusso informativo tracciato in avanti o all'indietro finché un'attività viene incontrata due volte. Le attività incontrate in questo percorso tra due attività costituiscono un circolo informativo) o metodo delle potenze della matrice di adiacenze (come sostiene Warfielf (1973), l'elevamento della DSM alla ennesima potenza indica

quale elemento può essere raggiunto da se stesso in n passi, osservando il valore non nullo tra gli elementi sulla diagonale).

5. Raggruppamento degli elementi coinvolti in un singolo circuito in un unico elemento rappresentativo. A questo punto è possibile ritornare alla fase 1.

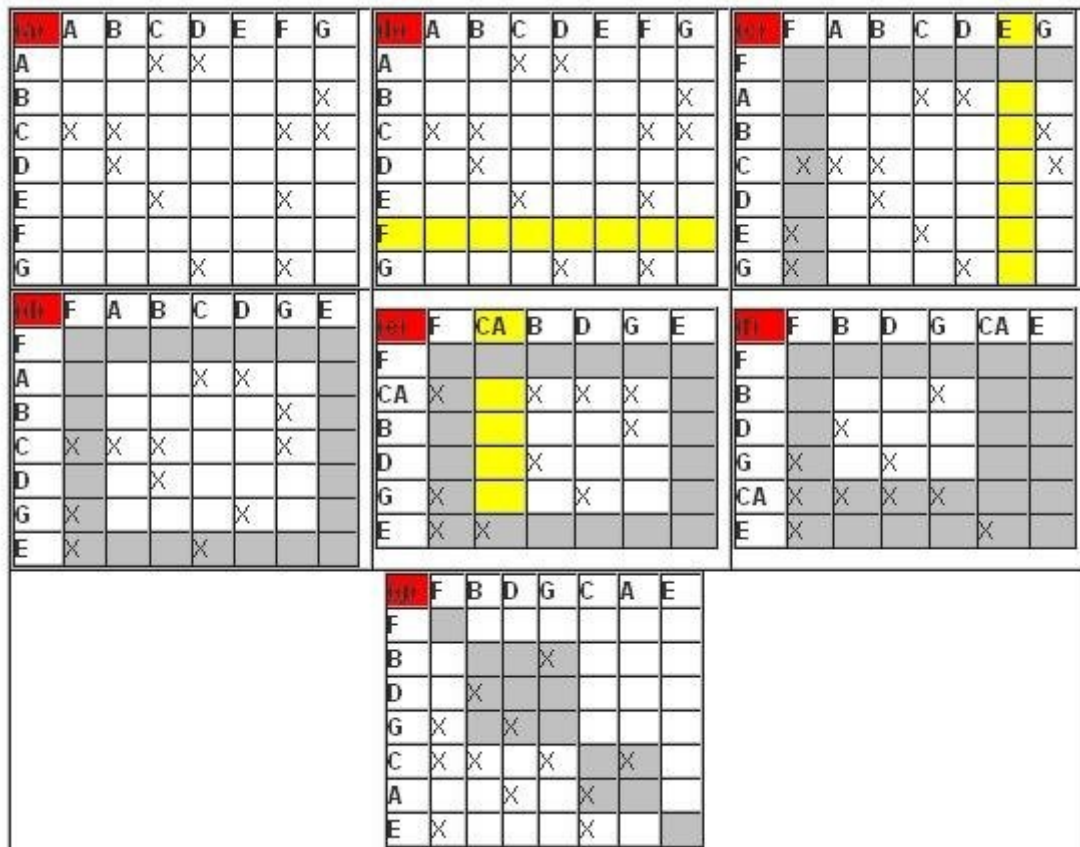


Figura 3.8 – Fasi della partizione

I ragionamenti sopra descritti, però prendono in considerazione solo le dipendenze “dirette”, ossia incondizionate. Molto spesso però le scelte che i progettisti devono affrontare sono più complicate. Infatti i sistemi odierni sempre più spesso includono componenti la cui evoluzione è al di fuori del controllo.

Se riprendiamo la prima modularizzazione di Parnas notiamo che il modulo B è dipendente da dal modulo E, e il modulo C è dipendente al modulo B, queste due dipendenze sono dirette e ben visibili dal DSM, ma il modulo C risulta dipendente, seppur in modo indiretto al modulo E, e dal DSM questa dipendenza non è facilmente visibile. Ne consegue che il modulo C risente dell'eventuali modifiche sul modulo E, quindi è interessante capire come una modifica o un vincolo su un

determinato modulo, possa influire sull'intero progetto. In altre parole, da un punto di vista probabilistico ci domandiamo se

$$C \perp E | B$$

3.5 Indipendenza condizionata

Di seguito riporto la definizione di indipendenza condizionata, come presentata da J. Pearl:

“Dato $V = \{V_1, V_2, \dots\}$ un insieme finito di variabili. Dato $P(\cdot)$ una funzione di probabilità congiunta oltre le variabili in V , e sia X, Y, Z tre sottoinsiemi di variabili in V . Gli insiemi X ed Y sono dette condizionate e indipendenti dato Z se

$$P(X | y, z) = P(x | z) \quad \text{per ogni } P(y, z) > 0$$

In altre parole, la conoscenza del valore di Y non fornisce ulteriori informazioni su X , una volta che sappiamo Z . (Metaforicamente, Z scherma X da Y)”

L'equazione sopra citata è un modo per esprimere quanto segue: per ogni x configurazione delle variabili nella serie X e per qualsiasi configurazione y e z le variabili in Y e Z soddisfano $P(Y = y, Z = z) > 0$, abbiamo

$$P(X = x | Y = y, Z = z) = P(X = x | Z = z)$$

Possiamo usare la notazione di Dawid (1979), quindi $(X \perp Y | Z)$ o semplicemente $(X \perp Y | Z)$ per denotare la condizione di indipendenza di Z e Y dato Z ; quindi:

$$(X \perp Y \vee Z) \text{ se e solo se } P(X \vee y, z) = P(X \vee Z)$$

per tutti i valori x, y, z così che $P(y, z) > 0$. L'indipendenza incondizionata (chiamata anche *indipendenza marginale*) sarà denotata $(X \perp Y | \theta)$; così che:

$$(X \perp Y | \theta) \text{ se e solo se } P(x \vee y) = P(x) \forall P(y) > 0$$

E' necessario notare che $(X \perp Y|Z)$ implica l'indipendenza condizionata di tutte le coppie di variabili $V_i \in X$ e $V_j \in Y$, ma non è necessariamente vero il contrario.

La seguente lista (parziale) rappresenta le proprietà delle relazioni delle indipendenze condizionate.

$$\textit{Simmetria} (X \perp Y|Z) \rightarrow (Y \perp X|Z)$$

$$\textit{Decomposizione} (X \perp YW|Z) \rightarrow (X \perp Y|Z)$$

$$\textit{Unione debole} (X \perp YW|Z) \rightarrow (X \perp Y|ZW)$$

$$\textit{Contrazione} (XY|Z) \wedge (X \perp W|ZY) \rightarrow (X \perp YW|Z)$$

$$\textit{Intersezione} (X \perp W|ZW) \wedge (X \perp Y|ZW) \rightarrow (X \perp YW|Z)$$

Da notare che l'intersezione è valida nelle distribuzioni di probabilità strettamente positive. La prova di queste proprietà può essere derivata dall'assioma della teoria della probabilità e dalla seguente equazione:

$$(X \perp Y|Z) \text{ se } P(x|y, z) = P(x|z)$$

Pearl e Paz (1987) e Geiger (1990) chiamarono queste proprietà “*graphoid axiom*” e hanno dimostrato di controllare il concetto di rilevanza informazionale in un ampio spettro di interpretazioni (Pearl, 1988). Nei grafi, per esempio, queste proprietà sono soddisfatte se noi interpretiamo $(X \perp Y|Z)$ come segue: “*tutti i percorsi da un nodo del sottoinsieme X ad un nodo del sottoinsieme di Y sono intercettati un nodo del sottoinsieme Z*” (Pearl, 1988)

L'interpretazione intuitiva degli assiomi dei grafi (graphoid axioms) è quella che segue (Pearl, 1988):

- L'assioma di *simmetria* afferma che, ogni volta che Z è noto, se Y non ci dice

nulla di nuovo su X, allora X non ci dice nulla di nuovo su Y.

- L'assioma di *decomposizione* afferma che se due item combinati di informazione sono giudicati irrilevanti per X, allora ogni item di separazione è allo stesso modo irrilevante.
- L'assioma di *unione debole* afferma che conoscere informazioni irrilevanti W non permette alle informazioni irrilevanti Y di diventare rilevanti per X.
- L'assioma di *contrazione* afferma che se giudichiamo W irrilevante per X dopo aver conosciuto alcune irrilevanti informazioni Y, allora W dovrebbe essere irrilevante prima che noi conosciamo Y. Insieme, le proprietà di unione debole e di contrazione indicano che informazioni irrilevanti non alterano lo stato di rilevanza delle altre proposizioni nel sistema: ciò che era rilevante rimane rilevante, ciò che era irrilevante rimane tale.
- L'assioma di *intersezione* afferma che se Y è irrilevante per X quando conosciamo W e se W è irrilevante per X quando conosciamo Y, allora né W né Y (neppure la combinazione dei due) è rilevante per X.

4 GRAFI

Un grafo consiste in un insieme di vertici (o nodi) ed un insieme di archi che connettono coppie di nodi. I vertici del nostro grafo corrisponderanno ai nostri moduli e le frecce denotano una certa relazione tra i moduli.

Due moduli connessi da una freccia sono detti adiacenti. Ogni nodo di un grafo può essere orientato (marcato con una freccia) o non orientato (marcato con una linea), come rappresentato nelle figure 4.1 e 4.2.

Se un nodo è orientato significa che il secondo nodo dipende dal primo. Se, invece, due nodi non sono orientati, allora c'è una dipendenza reciproca.

Il percorso di un grafo è la sequenza di nodi necessari per andare dal nodo X al nodo Y.

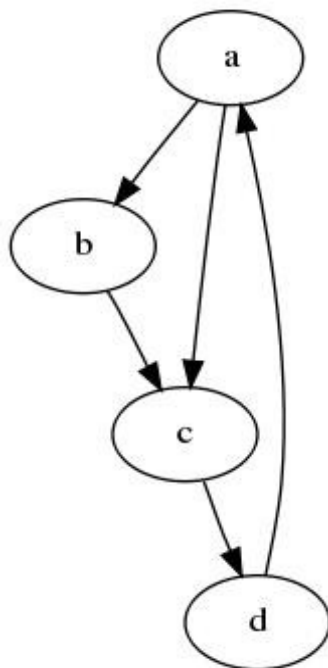


Figura 4.1 – Esempio di grafo orientato

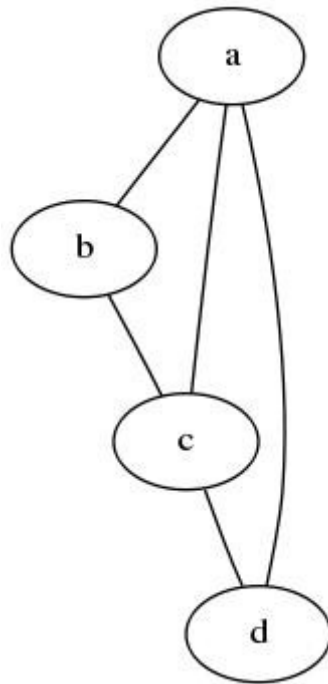


Figura 4.2 – Esempio di grafo non orientato

Un grafo orientato può includere dei cicli, ($X \rightarrow Y, Y \rightarrow X$), ma non dei cappi ($X \rightarrow X$). I grafi che non contengono cicli sono chiamati aciclici; la figura 4.3 rappresenta un esempio di grafo aciclico.

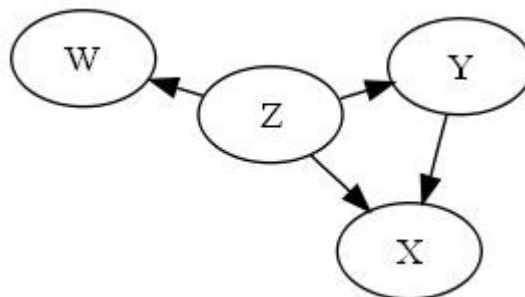


Figura 4.3 - Esempio di grafo aciclico (10)

La terminologia usata nei grafi per determinare le relazioni tra i nodi è quella parentale (genitori, figli, discendenti, antenati, coniugi etc), ad esempio nella figura 4.3, W è figlio di Z, mentre Z è padre di W Y e X, mentre il nodo W (della figura 4.4) è un antenato di Y, mentre Y ha due genitori X e Z. La famiglia in un grafo è un un

insieme di nodi, contenente un nodo con tutti i suoi antenati, ad esempio $\{W\}$, $\{Z,W\}$, $\{X\}$ e $\{Y,Z,X\}$ sono delle famiglie del grafo di figura 4.4.

Un nodo in un grafo orientato è chiamato radice se non ha genitori, foglia se non ha figli.

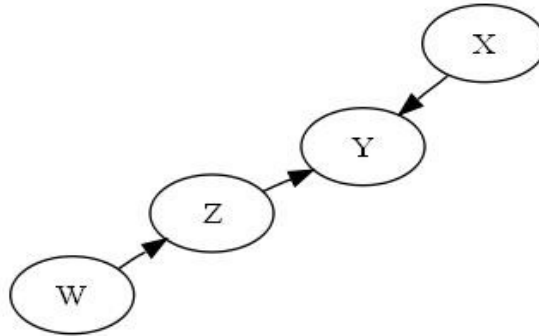


Figura 4.4

Tutti i DAG (Direct Acyclic Graph) hanno almeno una radice e una foglia. Un DAG orientato in cui ogni nodo ha al più un genitore si chiama albero (figura 4.5), e un albero in cui tutti i nodi hanno un solo figlio è detto catena (figura 4.6).

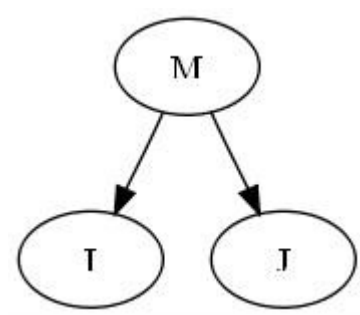


Figura 4.5 - Albero

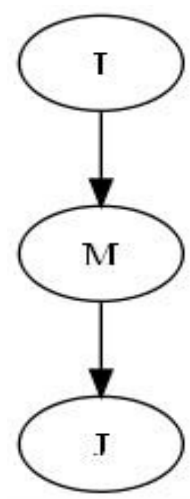


Figura 4.6 - Catena

4.1 Reti Bayesiane

Il ruolo dei grafi nel calcolo della probabilità facilita la rappresentazione della funzione di probabilità congiunta.

Consideriamo una distribuzione congiunta $P(X_1, \dots, X_n)$ per n variabili. Per memorizzare la $P(X_1, \dots, X_n)$ è richiesta una tabella con 2^n voci. Ne consegue che otterremmo una tabella particolarmente estesa.

È possibile, invece, economizzare lavorando sulle relazioni di dipendenza tra le variabili. Considerando che ogni variabile dipende solo da un piccolo sottoinsieme di variabili, essere a conoscenza di tali relazioni di dipendenza ci permette di decomporre la funzione di ripartizione di grandi dimensioni in funzioni di distribuzione più piccole (ciascuna per un piccolo sottoinsieme di variabili).

I grafi svolgono un ruolo essenziale di tale decomposizione, in quanto forniscono una viva rappresentazione del set di variabili che sono rilevanti per la rappresentazione della funzione di probabilità congiunta. Da questo punto di vista, i grafi sono da considerarsi migliori rispetto alla DSM.

I grafi non orientati, a volte chiamati Markov network (Pearl 1988), sono utilizzati principalmente per rappresentare relazioni simmetriche (Cox and Wermuth 1996).

I grafi orientati, in particolare i DAG, sono stati usati per rappresentare relazioni (Lauritzen 1982, Cox and Wermuth 1983) e sono chiamati Bayesian network. Tale

termine è stato coniato da Pearl (1985) per sottolineare tre aspetti:

1. La natura soggettiva degli input
2. La dipendenza condizionata come base per l'aggiornamento delle informazioni
3. La distinzione tra il nesso di causalità e i modi di ragionamento probabilistico

La decomposizione di base offerta da un grafo orientato e aciclico può essere illustrata come viene descritto di seguito. Supponiamo di avere una distribuzione P definita su n variabili discrete X_1, \dots, X_n .

Il calcolo della probabilità ci permette di decomporre P come un prodotto di n distribuzioni condizionate:

$$P(X_1, \dots, X_n) = \prod P(X_j | X_1, \dots, X_{j-1})$$

Supponiamo ora che la probabilità condizionata di alcune variabili X_j non sia sensibile a tutti i predecessori di X_j , ma solo ad un piccolo sottoinsieme dei predecessori. In altre parole, si supponga che X sia indipendente da tutti gli altri predecessori. Una volta che sappiamo il valore di un gruppo selezionato di predecessori chiamato PA_j , calcoliamo la probabilità di dipendenza solo su quelli.

Quindi, possiamo scrivere:

$$P(X_j | X_1, \dots, X_{j-1}) = P(X_j | pa_j)$$

Invece di specificare la probabilità di X_j condizionata su tutte le possibili realizzazioni dei suoi predecessori X_1, \dots, X_{j-1} abbiamo bisogno soltanto della possibile realizzazione di set PA_j .

L'insieme pa_j è chiamato "Markovian parents" di X_j .

Definiamo come di seguito i Markovian parents:

Sia $V = \{X_1, \dots, X_n\}$ un set di variabili, e sia $P(v)$ una comune distribuzione di probabilità

su queste variabili. Una serie di variabili PA_j sarebbe Markovian parent di X_j se PA_j è di un insieme minimo del predecessore di X_j che rende indipendenti tutti i suoi altri predecessori. In altre parole, PA_j è un qualsiasi sottoinsieme di $\{X_1..X_{j-1}\}$

$$P(X_j | paj) = P(X_j | X_1, \dots, X_{j-1}) \quad (3.0)$$

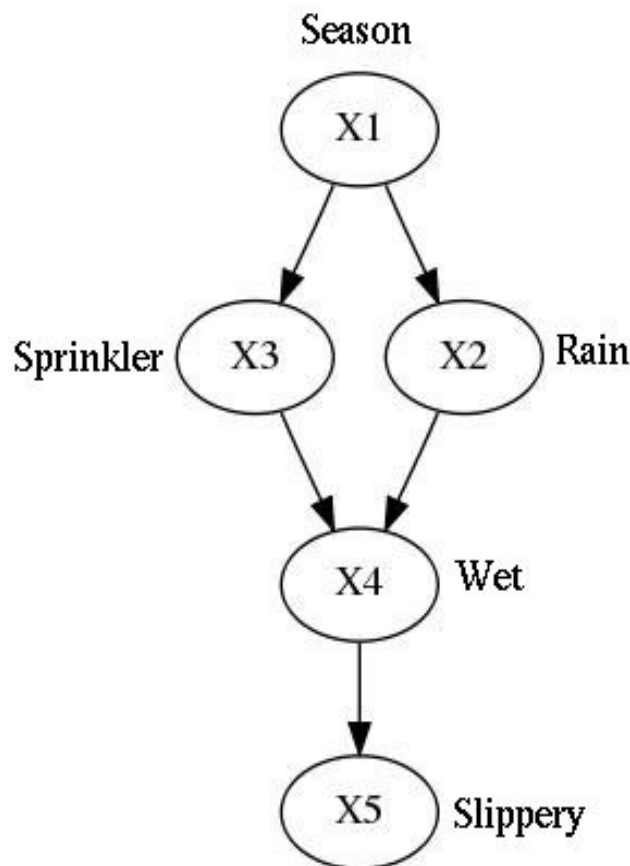


Figura 4.7

La definizione sopra citata assegna a ciascuna variabile X_j un insieme PA_j (serie di variabili che precedono X_j) che sono sufficienti per determinare la probabilità di X_j .

Questi assegnamenti possono essere rappresentati nella forma di un grafo aciclico (DAG) in cui ogni variabile è rappresentata da un nodo e da una freccia che collega il genitore con il nodo figlio.

La definizione ci suggerisce anche un semplice metodo ricorsivo per costruire una DAG: partendo dalla coppia (X_1, X_2) , disegniamo una freccia da X_1 a X_2 se e solo se

le due variabili sono dipendenti. Continuando con X3 non disegniamo la freccia nel caso in cui X3 è indipendente da (X1,X2); diversamente analizziamo se X2 scherma X3 da X1, o se X1 scherma X3 da X2. Nel primo caso disegniamo la freccia da X2 a X3, nel secondo caso disegniamo una freccia da X1 a X3.

Chiamiamo PA_j l'insieme di tutti i predecessori del nodo X_j e tracciamo una freccia tra ogni predecessore ed il nodo X_j otteniamo così un DAG

La figura 4.7 illustra un semplice Bayesian network, Essa descrive le relazioni tra le stagioni dell'anno (X1), se cade la pioggia (X2) se pioviggina(X3) se il marciapiace è bagnato (X4) se il marciapiende è scivoloso (X5). Tutti le variabili sono binarie (vere o false) eccetto per la radice (X1) che può assumere 4 valori (in quanto rappresenta le 4 stagioni).

L'assenza di un legame diretto tra X1 e X5, ad esempio, ci indica che l'influenza delle oscillazioni stagionali della scivolosità della pavimentazione è mediato da altre condizioni, dal momento che sappiamo che X4 rende X5 indipendente da (X1, X2, X3).

Definendo come sopra riportato “Bayeasian Network” come un vettore di indipendenza condizionale lungo l'ordine di costruzione. Ne consegue chiaramente che ogni distribuzione che soddisfa 3.0 deve essere decomposta

$$P(X_1, \dots, X_n) = \prod_i P(X_i | pa_i) \quad (3.1)$$

Per esempio il DAG di figura 3.7 induce la decomposizione

$$P(X_1, X_2, X_3, X_4, X_5) = P(X_1) P(X_2 | X_1) P(X_3 | X_1) P(X_4 | X_2, X_3) P(X_5 | X_4)$$

Il prodotto della decomposizione 3.1 non è più ordine specifico nel momento in cui, dati P e G, possiamo verificare se P si decompone nel prodotto dato da 3.1 senza fare alcun riferimento alla variabile di ordinazione.

Possiamo pertanto concludere che una condizione necessaria per un DAG G affinché sia un “Bayeasian Network” della distribuzione di probabilità P è per P di ammettere il prodotto di decomposizione dettata dalla G come indicato nella

produttoria 3.1.

4.2 Definizione di Markov Compatibility

Se una funzione di probabilità ammette la fattorizzazione (3.1) vista sopra relativa ad un DAG G , diciamo che G rappresenta P , che G e P sono compatibili, o che P è relativo a Markov dato G .

Accertare la compatibilità tra il DAG e la probabilità è importante nella costruzione di modelli statistici perché la compatibilità è una condizione necessaria e sufficiente per un DAG G per spiegare un insieme di dati empirici rappresentata da P , ossia, per descrivere un processo stocastico in grado di generare P . Se il valore di ciascuna variabile X_i è scelta casualmente con una certa probabilità P , basate esclusivamente sui valori PA_j precedentemente scelto per PA_j , quindi la distribuzione globale di P che genera istanze di $(X_1, X_2 \dots X_n)$ sarà Markov relativa a G .

Viceversa, se P è Markov relativa a G , allora esiste un insieme di probabilità $P_i (X_i | pa_i)$ secondo il quale siamo in grado di scegliere il valore di ciascuna variabile X_i , in modo che la distribuzione dei casi generato $x_1, x_2, \dots x_n$ sarà pari a P .

Un modo conveniente di caratterizzare l'insieme di distribuzione compatibile con un DAG G è quello di elencare la serie di indipendenze condizionate che ciascuna di tali distribuzione deve soddisfare. Queste indipendenze possono essere lette da un Dag utilizzando un criterio grafico chiamato *d-separation*, che svolgerà un ruolo importante in molte discussioni in questo lavoro.

4.3 Il criterio della d-Separation

Prendiamo in considerazione tre insiemi disgiunti di variabili, X , Y e Z , che sono rappresentati come nodi di un DAG G . Per verificare se X è indipendente da Y dato Z , in ogni distribuzione compatibile con G , dobbiamo verificare se i nodi corrispondenti alle variabili Z "blocco" percorso da tutti i nodi di X in Y .

Per percorso si intende una sequenza di nodi consecutivi (di qualsiasi direzionalità) in un grafo, mentre il blocco ha come funzione di fermare il flusso di informazioni (o

di dipendenza) tra le variabili che sono collegati da un siffatto percorso, come definito segue:

Un percorso P è detto d-separated (o bloccato) dato un insieme di nodi Z , se e solo se:

1. P contiene una catena (chain) $i \rightarrow m \rightarrow j$ o un fork $i \leftarrow m \rightarrow j$ tale che il nodo centrale m è in Z oppure
2. p contiene un fork invertito (o collider) $i \rightarrow m \leftarrow j$ tale che il nodo centrale m non è in Z e tale che m non sia discendente di Z .

Un insieme Z è detto d-separated X da Y se e solo se Z è bloccato da ogni percorso da un nodo in X ad un nodo in Y .

L'intuizione alla base del d-separazione è semplice e può essere meglio riconosciuta, attribuendo un senso al nesso di causalità con le frecce nel grafo: in una catena $i \rightarrow m \rightarrow j$ e in un fork $i \leftarrow m \rightarrow j$ le due variabili estreme sono marginalmente dipendenti ma diventano indipendenti l'una dall'altra ossia (bloccate) una volta condizionata la variabile centrale (nel nostro caso m)

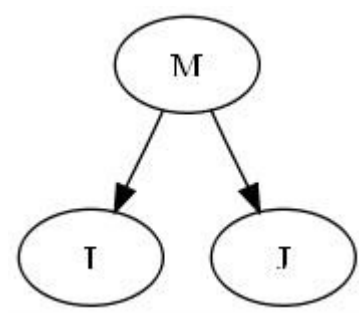


Figura 4.8 - Fork

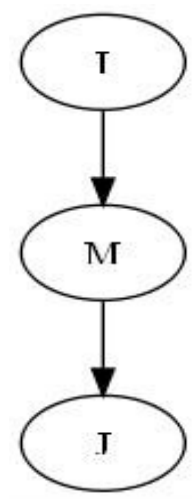


Figura 4.9 - Catena (chain)

Condizionando m , essa "blocca" il flusso di informazioni lungo il percorso, dal momento che, qualora condizionassimo i non ha alcun effetto sulla probabilità di j , dato m .

Il fork invertito (o collider) $i \rightarrow m \leftarrow j$, che rappresenta due cause con un comune effetto, funziona in modo opposto. Se le due variabili estreme sono (marginalmente) indipendenti, esse diventeranno dipendenti (cioè connesse tramite un percorso sbloccato) una volta che la variabile nel mezzo (nel nostro caso m) o ogni suo discendente viene condizionata.

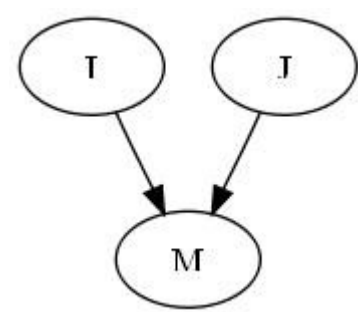


Figura 4.10 - Collider

Questo può essere confermato nella Figura 4.7.

Una volta che sappiamo la stagione, X_2 e X_3 sono indipendenti, ma constatato che il pavimento è bagnato o scivoloso, ciò rende X_2 e X_3 dipendenti, perché rifiutare una di queste spiegazioni aumenta la probabilità delle altre.

Nella figura 3.7 $X=\{X_2\}$ e $Y=\{X_3\}$ sono d-separated da $Z=\{X_1\}$ perché entrambi i percorsi X_2 e X_3 sono bloccati da Z . Il percorso $X_2 \leftarrow X_1 \rightarrow X_3$ perché esso è un fork in cui il nodo centrale X_1 è in Z , mentre il percorso $X_2 \rightarrow X_4 \leftarrow X_3$ è bloccato perché essendo un collider in cui il nodo centrale X_4 e tutti i suoi discendenti sono fuori Z . In ogni caso Z e Y non sono d-separated dall'insieme $Z' = \{X_1, X_5\}$: il percorso $X_2 \rightarrow X_4 \leftarrow X_3$ (collider) non sono bloccati da Z' dal nodo X_5 , un discendente del nodo centrale X_4 che è in Z' . Metaforicamente, conoscendo il valore della conseguenza X_5 rende le sue cause X_2 e X_3 dipendenti, come se un percorso fosse aperto lungo le frecce convergenti X_4 .

A prima vista, si potrebbe trovare un po' strano che condizionando un nodo che non si trova su un percorso bloccato ciò possa sbloccare il percorso stesso. Tuttavia, questo corrisponde a un modello generale di relazioni casuali: osservazioni su una conseguenza comune di due cause indipendenti tende a rendere queste cause dipendenti, in quanto le informazioni su una delle cause tende a rendere gli altri più o meno probabile, dato che ciò implica la conseguenza.

Nelle figure 4.11a e 4.11b mostrano un esempio elaborato di d-separation: l'esempio (a) contiene un arco bidirezionale $Z_1 \leftrightarrow Z_3$ e l'esempio (b) coinvolge un ciclo direzionato $X \rightarrow Z_2 \rightarrow Z_1 \rightarrow X$. Nell'esempio a i percorsi tra X e Y sono bloccati mentre nessuno tra $\{Z_1, Z_2, Z_3\}$ sono vincolati.

Tuttavia, il percorso $X \rightarrow Z_1 \leftrightarrow Z_3 \leftarrow Y$ diventa sbloccato quando Z_1 è vincolato.

Questo è così perché Z_1 sblocca il "colliders" a livello sia Z_1 e Z_3 : primo perché Z_1 è il nodo di collisione del collider, secondo perché Z_1 è un discendente del nodo di collisione Z_3 attraverso il percorso il percorso $Z_1 \leftarrow Z_2 \leftarrow Z_3$

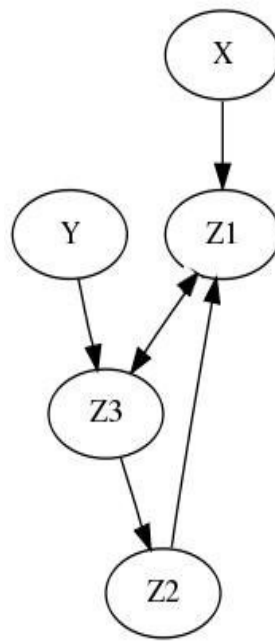


Figura 4.11 (a)

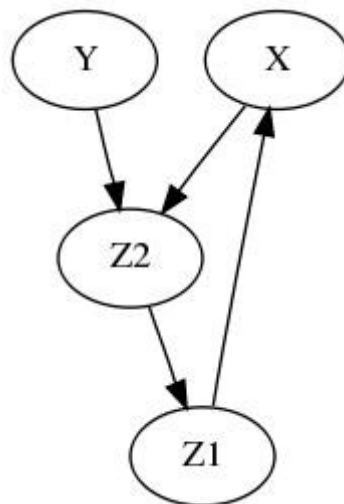


Figura 4.11 (b)

Nella figura 4.11 b, X e Y non possono essere d-separated di qualsiasi insieme di nodi, incluso l'insieme vuoto. Se condizioniamo Z_2 , blocchiamo il percorso $X \leftarrow Z_1 \leftarrow Z_2 \leftarrow Y$ ma sblocciamo $X \rightarrow Z_2 \leftarrow Y$. Se condizioniamo Z_1 blocchiamo nuovamente il percorso $X \leftarrow Z_1 \leftarrow Z_2 \leftarrow Y$ e sblocciamo il percorso $X \rightarrow Z_2 \leftarrow Y$, perché Z_1 è un discendente del nodo di collisione Z_2 .

Le connessioni tra i d-separated e le indipendenze condizionate sono stabilite attraverso il seguente teorema dovuto a Verna e Pearl (1988):

Se X e Y sono d-separated da Z in un DAG G, allora X è indipendente da Y, condizionatamente in Z in tutte le distribuzioni compatibili con G. Viceversa, se X e Y non sono d-separated da Z in un DAG G, allora X e Y sono dipendenti condizionatamente in Z in almeno una minima distribuzione compatibile con G.

5 ANALISI DELLE DIPENDENZE

Al termine della presentazione teorica fin qui esposta, può sorgere la perplessità rispetto alla possibilità, una volta posti di fronte ad un software complesso sviluppato con la programmazione modulare, che identificare le relazioni di indipendenza tra i differenti moduli possa essere un compito alquanto difficile.

Perplessità lecita, in quanto tale compito, considerata la complessità strutturale del software e del flusso di informazioni che lo determina, risulta infatti particolarmente ostico. È però indispensabile individuare relazioni dipendenti e indipendenti tra i moduli, poiché ciò rende più veloce ed efficace (soprattutto in termini di economizzare energie e risorse) la modificazione del software e l'adattamento dello stesso ad esigenze differenti da quelle per le quali è stato creato.

Per ovviare a questo problema, abbiamo sviluppato un software il cui obiettivo è quello di identificare e valutare le dipendenze di un progetto.

Per sviluppare tale software si è deciso di lavorare sul grafo della prima modularizzazione di Parnas, e vedere il comportamento delle dipendenze dopo aver posto dei vincoli su alcuni moduli.

A seguito di quanto esposto in questo lavoro, è interessante osservare come le dipendenze giochino un ruolo essenziale sui vari moduli.

Se analizziamo la prima modularizzazione di Parnas e ci concentriamo su alcuni percorsi, a secondo che il sottografo preso in considerazione sia un fork (la catena è un albero particolare dove ogni nodo ha un solo figlio) oppure sia un collider, abbiamo comportamenti diversi.

Prendiamo in considerazione i nodi E e F (ricordando che il nodo E rappresenta le strutture dati del modulo di shift mentre F gli algoritmi di shift), ci sono 2 percorsi E-B-F e il percorso diretto E-F. Grazie alla presenza di quest'ultimo percorso, F rimarrà sempre dipendente al nodo E, anche nel caso vincolassimo B.

Tale dipendenza significa che qualora cambiamo il modulo E, siamo tenuti a cambiare anche il modulo F.

Ora consideriamo un caso decisamente più interessante: i nodi E e L (algoritmi di output). In questo caso, ci sono ben 4 percorsi (E-B-L E-H-L E-B-H-L E-H-B-L). Ne consegue che se abbiamo bisogno di rendere L indipendente (o d-separated) da E, dobbiamo vincolare entrambi i nodi centrali (B e H, rispettivamente Input Data e Alph Data). Ciò comporta che qualora modifichiamo il modulo E, non occorre conoscere i cambiamenti di L, e viceversa.

Analogamente con la seconda modularizzazione di Parnas, i nodi E e F () Circ Data e Circ Alg) hanno un collegamento diretto; la dipendenza rimane, in questo caso, tale, mentre E ed L sono d-separated dal modulo N (Line Type) e quindi possono essere resi indipendenti controllando i cambiamenti di N.

Questa è una dimostrazione che la seconda modularizzazione è migliore rispetto la prima. Infatti poter avere gli algoritmi in ogni caso indipendenti dalle strutture dati ci permette maggior flessibilità e riusabilità del codice, mentre con la prima modularizzazione potevamo ottenere tale indipendenza, ma dovendo vincolare ben 2 moduli.

Riassumendo, se il percorso rappresenta un fork, e il percorso più lungo dal nodo A al nodo B richiede n salti, e vogliamo valutare la $P(B|X)$ (dove X è un nodo che si trova sul percorso AB), ha senso provare vincolando al massimo $n-2$ nodi (ossia tutti gli ancestor di B, ad esclusione del nodo di partenza, e naturalmente escluso B). Se vincolo più di $n-2$ nodi, è evidente che qualche nodo non è un ancestor di B e, di conseguenza, non si trova nel percorso AB. Quindi, questi nodi in “eccesso” non andrebbero ad influire sulla $P(B|X)$.

Con la prima modularizzazione di Parnas i percorsi, di tipo fork, più lunghi richiedono 3 salti, non ha senso vincolare più di 2 nodi (quelli che stanno sul percorso, AB esclusi). Infatti, analizzando il percorso EBHL, se vincolo i 2 nodi (BH) notiamo che L è indipendente da E. Se, invece, vincolo un qualsiasi altro nodo, esso non va ad influire su $P(L|BH)$.

Un caso decisamente differente è invece il percorso di tipo collider. In questo caso il ruolo delle dipendenze è più interessante e nello stesso tempo più complesso, in

quanto a differenza del fork, dove possiamo solo bloccare un percorso, con il collider possiamo bloccarlo e sbloccarlo a ripetizione.

Ricordando che un collider è nella forma $i \rightarrow m \leftarrow j$, e che inizialmente i e j sono indipendenti, essi diventano marginalmente dipendenti una volta vincolato il nodo centrale m .

In questo caso per cambiare m devo conoscere i cambiamenti di i e j . Se controllo m , quindi, dovrò accordare opportunamente i cambiamenti di i a quelli di j e viceversa.

Prendiamo ora come riferimento i nodi H (Alph Data) e M (Master). Con la prima modularizzazione di Parnas, essi sono tra loro indipendenti, ma vincolando il nodo L (Out Alg) essi diventano dipendenti. Quindi, ne consegue che qualora modifichiamo un modulo, dobbiamo modificare anche l'altro. Se, invece, vincoliamo anche J (Out Type), allora i nodi H e M tornano nuovamente indipendenti.

Analogamente, con la seconda modularizzazione di Parnas, prendiamo in considerazione il collider dei nodi N (Line Type) e D (Circ Type). Essi sono d-separated dal nodo F (Circ Algh) e, grazie al percorso diretto N-F, il nodo E non influisce sulla dipendenza dei due nodi. Quindi, possiamo renderli dipendenti, ma non possiamo più renderli indipendenti.

Se analizziamo attentamente il grafo della seconda modularizzazione di Parnas, notiamo che tutti i collider hanno al massimo un nodo centrale: ciò implica che possiamo solo creare la dipendenza una sola volta, e senza più d-separare i due nodi (salvo rigenerando il modulo centrale originale). Questo fatto porta ad una maggiore chiarezza nel disegno del software, perché anche le dipendenze risultano maggiormente chiare e non mettono in atto un meccanismo di reciproco rimando.

L'uso delle proprietà topologiche del grafo permette perciò fare considerazioni più raffinate, rispetto a quelle ottenibili dalla DSM.

6 BIBLIOGRAFIA

- Parnas D.L.(1972a), “On the Criteria To Be Used in Decomposing Systems into Modules”, Communications of the ACM, December, Volume 15, Number 12
- Parnas D.L.(1972b), “A course on software engineering techniques”, ACM SIGCSE Bulletin archive, Volume 4 , Issue 1 March Pages: 154 - 159 New York, NY, USA
- Yassine A. (1999), “Engineering Design Management. An information structure approach”, International journal of production research. Vol 37 no. 19
- Warfield J.N. (1973), “Binary Matrices in System Modeling”, IEEE Transaction on System, Man, and Cybernetics, vol. 3, pp. 441-449
- Pearl e Paz (1987) “Graphoids: A graph based logic for reasoning about relevancy relations”, in B. D. Boulay, D. Hogg and L. Steel (eds), Advances in Artificial Intelligence II, North-Holland, Amsterdam, pp. 357—363.
- Pearl Judea (2001) “Casuality Models Reasoning and Inference”, Cambridge University Press
- Pearl J. (1988) “Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference”, Morgan Kaufmann Publishers Inc. San Francisco, CA, USA
- Geiger D.(1990) “Graphoids: a qualitative framework for probabilistic inference”, University of California at Berkeley Berkeley, CA, USA
- Cox and Wermuth (1996): “Multivariate dependencies: Models, analysis and interpretation.” in S.L. Lehmann (1986): “Testing statistical hypotheses, 2nd ed.”
- Lauritzen S.L.(1982) “Lectures on Multivariate Analysis”. Univ. Cop. Inst. Math. Stat.
- Sullivan K., Cai Y., Hallen B. (2001) “The structure and value ao modularity in software design”, University of Virginia Department of Computer Science Technical Report CS-2001-13, submitted for publication to ESEC/FSE