**RAGIONAMENTO CON VINCOLI**

Invece di ragionare esplicitamente in termini di stati, in genere è meglio descrivere gli stati in termini di **caratteristiche** e ragionare in termini di queste caratteristiche. Le funzionalità sono descritte utilizzando **le variabili** . Spesso le caratteristiche non sono indipendenti e ci sono **vincoli rigidi** che specificano combinazioni legali di assegnazioni di valori a variabili.

**Variabili e Mondi**

I problemi di soddisfazione dei vincoli sono descritti in termini di variabili e mondi possibili. Un **mondo possibile** è un possibile modo”stato o situazione” in cui il mondo (il mondo reale o un mondo immaginario) potrebbe essere.

Una **variabile discreta** è quella il cui dominio è finito o numerabile infinito. Una **variabile binaria** è una variabile discreta con due valori nel suo dominio. Un caso particolare di variabile binaria è una **variabile booleana** , che è una variabile con dominio{Vero, falso}. Possiamo anche avere variabili che non sono discrete; ad esempio, una variabile il cui dominio corrisponde alla retta reale o a un intervallo della retta reale è una **variabile continua** .

Dato un insieme di variabili, **un'assegnazione** sull'insieme di variabili è una funzione dalle variabili nei domini delle variabili. Un assegnazione quindi su un insieme di variabili {X1,X2,…,XK} è definita come {X1=v1, X2=v2 ,… ,XK=vK }, Dove vi è dentro dom(Xi). Questa assegnazione specifica che, per ciascuno i, variabile Xi viene assegnato un valore vi.

Un **mondo possibile** è definito come un assegnazione totale. Cioè, è una funzione da variabili in valori che assegna un valore a ogni variabile. Se mondo w è l'assegnazione{X1=v1,X2=v2,…,XK=vK}, diciamo quella variabile Xi ha valore vi nel mondo w.

Il numero di mondi possibili è dato dal prodotto del numero di valori nei domini delle variabili che caratterizzano il mondo.

Graphical user interface, text

Description automatically generated

Se ci sono N variabili, ciascuna con la dimensione del dominio D, ci sono DN mondi possibili. Uno dei principali vantaggi del ragionamento in termini di variabili è il risparmio computazionale. Molti mondi possono essere descritti da poche variabili.

**VINCOLI**

In molti domini, non tutte le possibili assegnazioni di valori alle variabili sono consentite. Un **vincolo rigido** , o semplicemente **vincolo** , specifica combinazioni legali di assegnazioni di valori ad alcune delle variabili.

Un **ambito** è un insieme di variabili. Una **relazione** su un ambito S è una funzione che va dalle assegnazioni che riguardano S ovvero l’ambito all’ insieme dato da {Vero, falso}. Cioè, specifica se ogni assegnamento è vero(cioè è soddisfatto) o falso(il vincolo risulta violato). Un **vincolo** C è un ambito S e una relazione su S. Si dice che un vincolo **coinvolge** ciascuna delle variabili nel suo ambito.

Per cui avendo un vincolo C su di un ambito S, una assegnazione A che sussista o riguardi un ambito S′ dove S⊆S′ risulterà soddisfare il vincolo C se l’assegnazione A ristretta allo scope “ambito” di S è mappata con (true) ovvero rispetta il vincolo, se risulterà falso allora tale assegnazione A violerà il vincolo.

Un mondo possibile w **soddisfa** un insieme di vincoli se, per ogni vincolo, i valori assegnati nel mondo w alle variabili nell'ambito del vincolo soddisfano il vincolo. In questo caso diciamo che il mondo possibile è un **modello** dei vincoli. Cioè, un **modello** è un mondo possibile che soddisfa tutti i vincoli.

I vincoli possono essere dichiarati in forma implicita tramite formule oppure in modo esplicito tramite un elenco di essi in fine possono avere diverse arietà.

Ex vincolo intensionale:



Ex vincolo estensionale in forma di tabella:

Table

Description automatically generated

**CONSTRAINT SATIFACTIONS PROBLEMS**

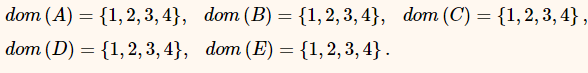
E’ costituito da:

* Un insieme di variabili per rappresentare il mondo e lo stato in cui si trova
* Un dominio per ogni variabile dichiarata
* Un insieme di vincoli

Un CSP è detto finito quando il # delle variabili è finito e il dominio di ogni variabile è a sua volta finito.

**Esempio di un CSP:**

*Supponiamo che il robot di consegna debba svolgere una serie di attività di consegna,*a, b, c, d ed e*. Supponiamo che ogni attività avvenga in qualsiasi momento* 1*,* 2*,* 3*,* 4*. Supponiamo che A sia la variabile che rappresenta il tempo dell’attività a, e analogamente per le altre attività. I domini delle variabili, che rappresentano i tempi possibili per ciascuna delle consegne, sono:*



Mentre i seguenti rappresentano i vincoli del problema sulle variabili:

A close-up of a calculator

Description automatically generated with low confidence

Dato un CSP è possibile porsi diverse domande e compiere i seguenti task:

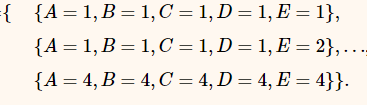
* Determinare se c’è o meno un modello per il CSP. Equivale a chiedere di determinare se esiste una soluzione per il CSP senza sapere quale sia.
* Trovare una soluzione al CSP
* Enumerare i modelli del CSP, quindi contare tutte le soluzioni possibili
* Trovare il miglior modello data una misura di costo (trovare la migliore soluzione)
* Determinare se alcune assegnazioni sono valide in tutti i modelli del CSP

Determinare se c’è un modello per un CSP finito è un problema NP-completo e per tali ragioni non esistono algoritmi capaci di risolvere il problema in minor tempo di quello esponenziale nel peggiore dei casi.

**ALGORITMO GENERATE-AND-TEST**

L' algoritmo di  **(generazione e verifica)** per trovare un modello controlla a turno ogni assegnazione totale; se questa assegnazione soddisfa tutti i vincoli, la restituisce come soluzione. Un algoritmo di generazione e verifica per trovare tutti i modelli è lo stesso tranne che, invece di restituire il primo modello trovato, salva tutti i modelli trovati per poi restituirli.

Per il precedente esempio visto sulle 5 attività a, b, c, d , e vediamo che l’algoritmo generate and test andrebbe di forza bruta a generare tutte le possibili combinazioni che forniscono una assegnazione andando a crearsi lo spazio di ricerca “D” così definito



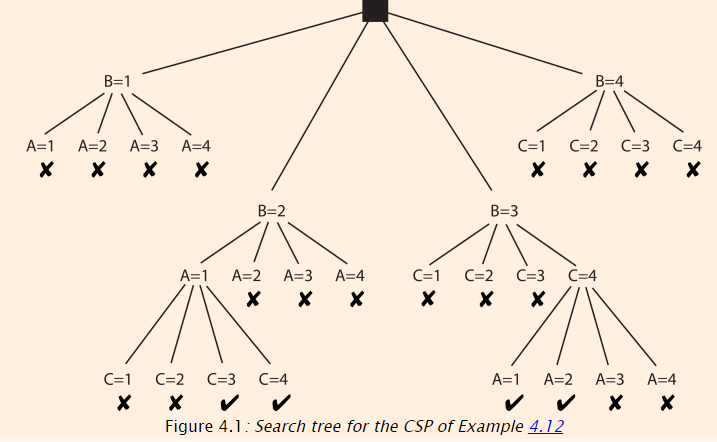
*In questo caso ci sono* |D|=  =1,024, Infatti se il # di variabili del CSP è **n** ed il dominio di ogni variabile ha cardinalità pari a d allora lo spazio di ricerca degli assegnamenti D sarà uguale a . E con un # di vincoli pari ad **e** avremo una complessità di test effettuati di .

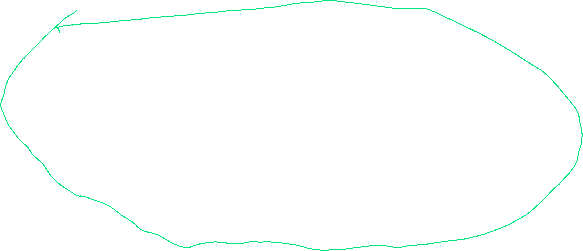
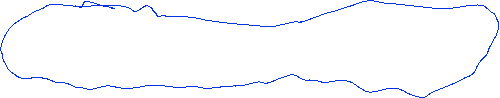
**RISOLUZIONE DI CSP CON LA RICERCA**

Il precedente algoritmo G&T lavora male poiché prima va a genere qualsiasi assegnazione e poi la testa per la verifica del soddisfacimento dei vincoli, si potrebbe migliorare se il test venisse prima eseguito solo sulle variabili legate ai vincoli e una volta sistemate tali variabili di modo da rispettare i vincoli continuare ad assegnare valori alle restanti variabili per trovare una soluzione.

Per migliorare il G&T dobbiamo pensare ad una rappresentazione differente del nostro CSP e in particolare possiamo ricorrere alla rappresentazione a grafo vista con i problemi di ricerca ed applicare in seguito una strategia ormai nota tra quelle di ricerca già viste che permetteranno di esplorare lo spazio di assegnazioni in modo più efficiente e rapido rispetto al G&T. Per dare vita al grafo di ricerca supponiamo che un nodo del grafo sia costituito da un numero variabile di variabili alle quali è assegnato un valore per cui risultano essere dei gruppi di variabili sulle quali vi è un assegnazione e i vicini di un nodo (o i figli di tale nodo) risultano essere nodi con lo stesso gruppo di variabili con in aggiunta una nuova variabile precedentemente non presente ed una sua assegnazione di modo da avere un assegnazione con un maggior numero di variabili che si avvicina ad una assegnazione totale per il CSP e quindi andiamo a costruire passo dopo passo l’assegnazione totale che risulterà essere un modello per il problema, quando per il CSP esisterà una soluzione, il goal della ricerca risulterà essere un nodo con assegnazione totale che rispetta i vincoli imposti. Come possiamo vedere dall’esempio qui lo start node è dato da una nodo senza alcuna variabile e senza alcuna assegnazione:

*Supponiamo di avere un CSP molto semplice con le variabili* A*,*B*,* C*, ciascuno con dominio*{1,2,3,4}*. Supponiamo che i vincoli siano* A<B *e*B<C*. Un possibile albero di ricerca:*





Nell’albero di ricerca i percorsi con una x sono i percorsi tagliati o eliminati ( potati) perché le assegnazioni non rispettavano i vincoli. Come è possibile notare il vantaggio rispetto al G&T è notevole, infatti, mentre questo genera e testa 64 assegnazioni totali ovvero tutte le possibili assegnazioni, con la ricerca (**tipicamente in questo caso DFS con backtracking**) generiamo solo 8 assegnazioni totali testate nel livello più profondo cerchiate in blu, e solo 16 assegnazioni parziali testate ed evidenziate in verde.



**CONSISTENCY ALGORITHM**

IL DFS presenta tuttavia alcune inconsistenze come ad esempio scopre che gli assegnamenti con A < B sono violati dopo averli testati, si potrebbe pensare a eliminare direttamente tali valori inconsistenti dal dominio di A in modo da no doverli testare nella costruzione dell’assegnazione lungo l’albero di ricerca. Per implementare questa idea abbiamo bisogno di una nuova struttura a grafo che prende il nome di **IPERGRAFO o costraint network,** il quale è così strutturato:

* Un nodo per ogni variabile del CSP raffigurato come cerchio
* Un nodo per ogni vincolo del CSP raffigurato come rettangolo
* Un insieme di valori associabili alle variabili ovvero ai nodi di tale grafo ove questo insieme di valori rappresenta i valori possibili che inizialmente coincidono con quelli del dominio, ma in seguito l’algoritmo andrà a potare restringendo il dominio della variabile ai soli valori consistenti con i vincoli del CSP
* Infine per ogni vincolo C e per ogni variabile X che sia implicata nell’ambito del vincolo C avremo un arco del tipo <X, C> che mette in relazione il nodo variabile al nodo vincolo inerente per tale variabile.

**Esempio di ipergrafo:**

A picture containing text, table

Description automatically generated

Gli archi di tale ipergrafo sono dati dalle seguenti coppie di nodi:

Text, letter

Description automatically generated

Quando ogni arco è consistente ovvero quando le variabili implicate nei vincoli hanno assegnazioni che rispettano i vincoli imposti possiamo dire che il grafo è consistente.

Per rendere un arco consistente si eliminano valori dai domini delle rispettive variabili

**GAC 🡪 GENERALIZED ARC CONSISTENCY ALGORITHM:**

**Graphical user interface, text, application

Description automatically generated**

L’ insieme to\_do contiene gli archi del ipergrafo che ci dicono quali variabili sono coinvolte in quali vincoli. ND sta per new domain ed è una variabile usata per restringere il dominio di ogni variabile chiamata in causa in un vincolo.



L' algoritmo **(GAC)**  rende coerente tutti gli archi della rete considerando un insieme “to\_do” di archi potenzialmente inconsistenti.L'insieme to\_do consiste inizialmente di tutti gli archi nel grafo. Fintanto che l'insieme non è vuoto, un arco ⟨X, C⟩ viene rimosso dall’insieme to\_do e considerato. Se l'arco non è coerente, viene reso coerente potando il dominio della variabile X. Tutti gli archi precedentemente coerenti che potrebbero, a seguito della potatura X, diventare incoerenti, vengono rimessi nel insieme to\_do . Questi sono gli archi ⟨Z, C′⟩, Dove C′ è un vincolo diverso da C che riguarda X e Z è una variabile coinvolta in C′ e diversa da X.

Il GAC non risolve il problema ma lo semplifica andando a ridurre enormemente in presenza di molti vincoli i domini delle variabili del CSP che conseguentemente risulta più semplice da affrontare con strategie di ricerca ed essere risolto.

Indipendentemente dall’ordine con cui vengono considerati gli archi la terminazione e quindi il risultato fornito sarà sempre lo stesso e in particolare se uno dei domini nel risultato restituito dovesse essere vuoto né consegue che il CSP non ha soluzione, mentre nel qual caso i domini di ogni variabile siano stati ridotti a dei singleton allora la soluzione al CSP sarà unica.

Il GAC ha una complessità spaziale di  e una complessità temporale data da .

Questo algoritmo risulta valido nel momento in cui i CSP sono finiti, ovvero hanno un dominio finito per ogni una delle variabili in gioco. Ci sono delle tecniche basate sull’idea di consistenza degli archi che anziché controllare di volta in volta una singola variabile, controllano una tupla di variabili per volta dando vita al **PATH CONSISTNECY ALGORITM.**

**DOMAIN SPLITTING**

Un'altra tecnica per semplificare il grafo (constraint network o ipergrafo) è quella del **Domain splitting o anche detta del case analysis**, dove l’idea è quella di dividere il problema in casi separati e risolverli separatamente per poi riunire le soluzioni date da ogni sotto problema.

Per splittare un dominio e quindi ridurre la complessità del problema da affrontare abbiamo differenti approcci infatti supponiamo di avere il seguente dominio {1, 2, 3, 4} per la variabile A i due approcci potrebbero essere:

* Un caso per ogni valore A=1, A=2, A=3 e A=4 ogni uno di questi risulta un problema ridotto ove A assume solo il valore indicato
* Altro approccio è dividere il dominio di partenza in due sotto insiemi disgiunti tali da ridurre il numero dei valori contenuti ovvero caso1🡪A prende valori in {1,2} e caso2-> A prende valori in {3,4}.

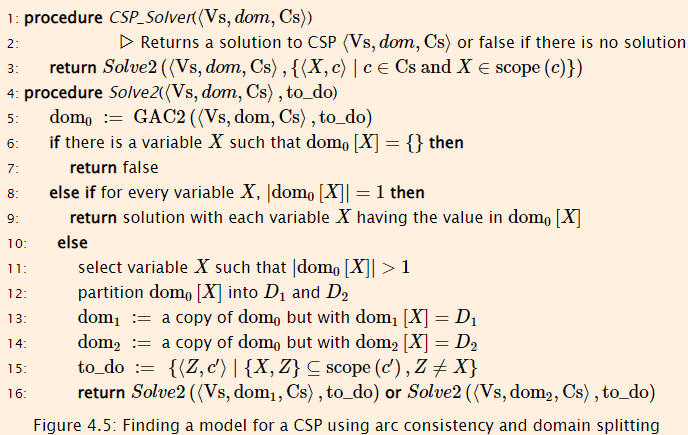
Risolvendo ricorsivamente i sotto problemi definiti dai casi analizzati con il domain splitting possiamo facilmente individuare se è presente una soluzione o meno utilizzando algoritmi di ricerca assieme agli algoritmi di consistenza degli archi GAC, Per cui l’idea generale dietro il **domain splitting** è dividere i domini per casi semplificando il CSP e su questi CSP semplificati adottare le tecniche precedentemente discusse di ricerca e consistenza dell’arco. È possibile adoperare prima il GAC e in seguito il domain splitting.

**ALGORITMO CHE COMBINA DOMAIN SPLITTING E ARC CONSISTENCY**

Vs 🡪 sono le variabili del CSP

Dom 🡪 i domini delle variabili

Cs 🡪 i vincoli sulle variabili

****

Questo algoritmo CSP\_solver richiama la procedura di risoluzione che a sua volta chiama il GAC2(line5) già visto per verificare la consistenza degli archi, nel caso il dominio di una variabile sia ridotto a {} allora non ci saranno soluzioni e verrà restituito false(line7), altrimenti se il dominio sarà per ogni variabile un unico valore allora sarà restituita la soluzione unica del CSP invece nel momento in cui ci sono più valori nei domini sarà eseguito il ramo di codice che esegue il domain splitting per ridurre il problem e chiamare nuovamente in modo ricorsivo GAC2 e split domain(line16).

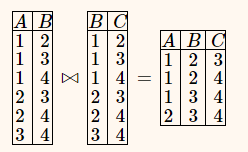
Il Domain Splitting permette di creare un nuovo spazio di ricerca ove qualsiasi degli algoritmi di ricerca visti è utilizzabile fatte le dovute considerazioni del caso inoltre poiché non siamo interessati ai percorsi, ma solo alle soluzioni date come assegnazioni totali ritrovate solo nei nodi foglia DFS è il + utilizzato.

**VARIABLE ELIMINATION [eliminazione di variabile]**

L’algoritmo VE [variable elimination] anziché semplificare i CSP andando a sfoltire i domini delle variabili, va a ridurne la complessità eliminando delle variabili e per ogni variabile eliminata crea un nuovo vincolo capace di simulare l’affetto di tale variabile su di altre variabili implicate in uno stesso vincolo.

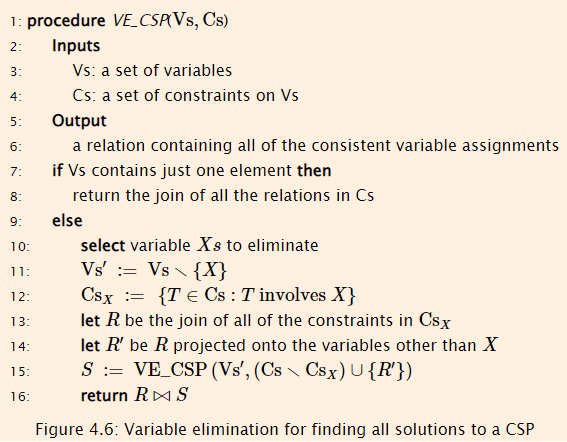
Ex:

*Si consideri un CSP che contiene le variabili A,* B*,* C*, ciascuna con dominio*{1,2,3,4}*. Supponiamo i vincoli che coinvolgono* B *sono* A<B *e* B<C*. Potrebbero esserci molte altre variabili, ma se* B*non ha vincoli in comune con queste variabili, eliminando* B *non si imporranno nuovi vincoli a queste altre variabili. Per rimuovere* B*, prima si necessita effettuare un join sulle relazioni che coinvolgono* B*:*



In seguito per ottenere una relazione su A e C indotta da B e dal suo vincolo effettuiamo una proiezione del precedente join solo su A e C ottenendo:





L'efficienza dell'algoritmo VE dipende dall'ordine in cui le variabili vengono selezionate. In generale, VE è efficiente quando la rete di vincoli è sparsa. Il numero di variabili nella relazione più grande restituita per un particolare ordinamento di variabili è chiamato **larghezza dell'albero(treewidth)** del grafico per quell'ordinamento di variabili. La larghezza dell'albero di un grafico è la larghezza minima dell'albero per qualsiasi ordinamento. La complessità di VE è esponenziale nella larghezza dell'albero e lineare nel numero di variabili. Tuttavia trovare un ordinamento di eliminazione che si traduca nella larghezza dell'albero più piccola è NP-difficile, per affrontare questo problema nel trovare un ordinamento idoneo ci sono alcune euristiche utilizzate:

* **min-factor** : in ogni fase, seleziona la variabile che risulta nella relazione più piccola
* **carenza minima(minimum deficiency)** o **riempimento minimo(minimum fill)** : in ogni fase, selezionare la variabile che aggiunge il minor numero di archi alla rete di vincoli rimanente.

VE può anche essere combinato con la consistenza dell'arco; ogni volta che VE rimuove una variabile, la consistenza dell'arco può essere utilizzata per semplificare ulteriormente il problema.