Ragionamento con Vincoli

Nicola Fanizzi
Ingegneria della Conoscenza
CdL in Informatica • *Dipartimento di Informatica*
Università degli studi di Bari Aldo Moro

indice

Mondi Possibili, Variabili e Vincoli

Mondi Possibili, Variabili e Vincoli

Dagli spazi degli stati a quelli delle caratteristiche

- **Feature** descritte attraverso *variabili*, spesso non *indipendenti*, e
  - *vincoli rigidi* specificano combinazioni lecite di assegnazioni alle variabili
  - *vincoli flessibili* preferenze sulle assegnazioni
- *Ragionamento* come generazione di assegnazioni che
  - soddisfino i vincoli rigidi
  - ottimizzino i vincoli flessibili

Variabili e Mondi

Si considereranno *problemi* descritti in termini di **variabili**

- *algebriche*: simboli usati per denotare caratteristiche del mondo (reale o immaginario) → *mondi possibili*
- *notazione*: nomi che iniziano per maiuscola es. X
- ogni variabile ha un **dominio** associato: insieme di valori che può assumere es. $dom(X)$
    - *variabili discrete*: dominio finito o almeno enumerabile
        - *binarie*: dominio di 2 valori
            - caso particolare: *booleane*, dominio $\{true, false\}$
    - *variabili continue*: se non sono discrete
        - ad es. con dominio $\mathbb{R}$ o un suo intervallo, e.g. $[0, 1]$

Assegnazione funzione da un insieme di variabili ai loro domini:

- dato $\{X_1, X_2, \ldots, X_k\}$
  a X_i si assegna $v_i \in dom(X_i)$ per ogni $i = 1, \ldots, k$:

$$\{X_1 = v_1, X_2 = v_2, \ldots, X_k = v_k\}$$

  - un solo valore per variabile
- **assegnazione totale** se riguarda tutte le variabili, **parziale** altrimenti

Mondo possibile: assegnazione totale (uno *stato* del mondo)

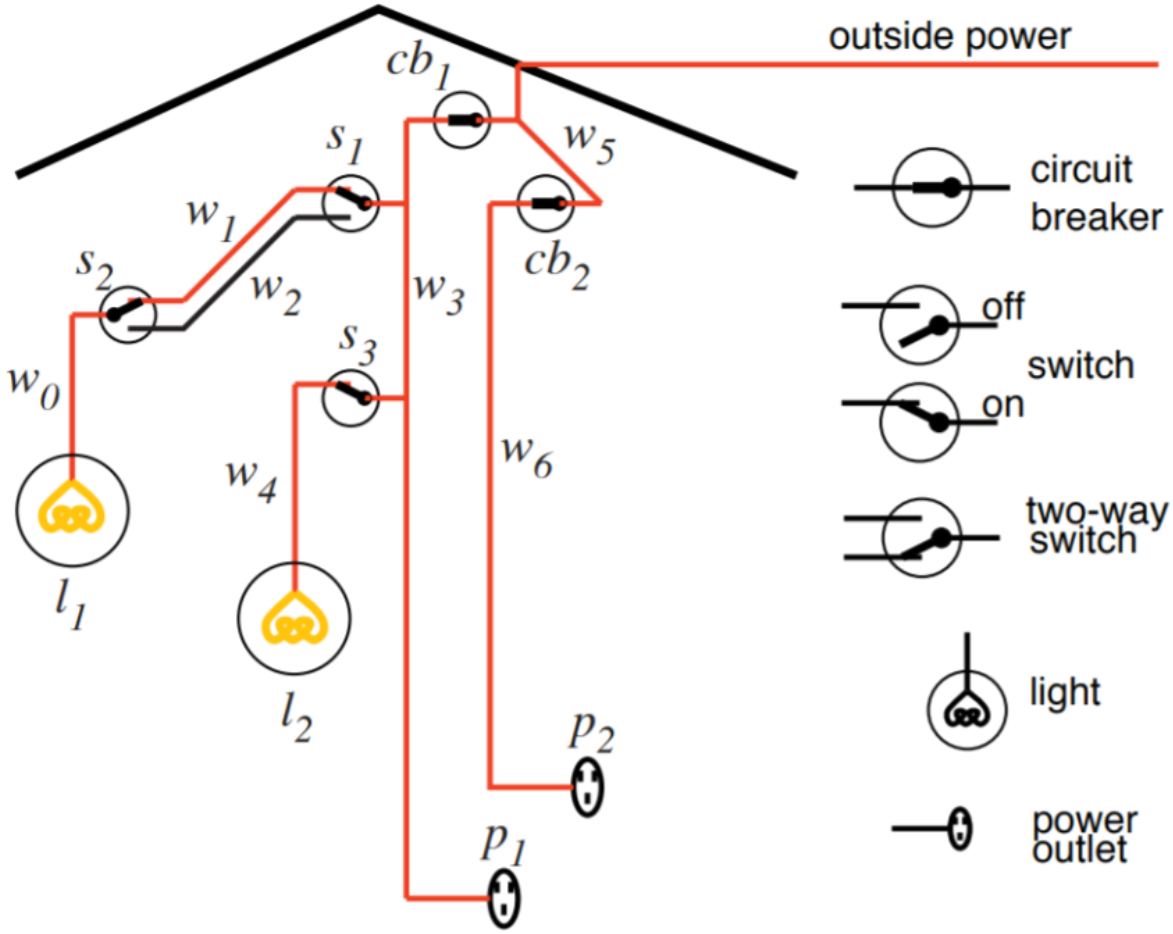
- funzione dalle variabili ai valori: a ognuna assegna un valore
  - dato il mondo $w = \{X_1 = v_1, X_2 = v_2, \ldots, X_k = v_k\}$
    si dice che: X_i *ha il valore* v_i *in* w

Esempio — variabili e assegnazioni

- $Ora\_Lezione$ ***discreta***
  - **per denotare l'ora d'inizio**
    $dom(Ora\_Lezione) = \{9, 10, 11, 12, 13, 14, 15, 16, 17, 18\}$
- $Temperatura$ ***continua***
  - **in °C:** $dom(Temperatura)$ **intervallo di reali** $[-273.15; 50]$
- $Piove$ ***booleana*** **casuale**
  - **indica se stia piovendo o meno in un dato momento**

L'assegnazione: $\{Ora\_Lezione = 11, Temperatura = 21.3, Piove = false\}$
specifica che "*la lezione inizia alle 11, ci sono 21.3°C e non piove*"

Esempio — diagnostica impianto elettrico



- una variabile per ogni posizione di deviatore (*switch*): su / giù
- una variabile per ogni punto luce: acceso / spento
- una variabile per ogni componente: in funzione / rotto
- ecc...

(..cont.)

- esempi:
  - S_1_pos binaria:
    - *posizione* del deviatore (switch) s_1, con dominio $\{up, down\}$
  - S_1_st discreta:
    - *stato* del deviatore s_1 con dominio:
      $\{ok, upside\_down, short, intermittent, broken\}$
  - $Number\_of\_broken\_switches$ intera:
    - *numero di deviatori rotti*
  - $Current\_w_1$ continua:
    - *corrente* in Ampére che passa attraverso il cavo (wire) w_1
- un mondo specifica posizioni e stato di ogni dispositivo, ecc.
  - e.g. $S_1\_pos = up$, $S_2\_pos = down$, $Cb_1\_st = ok$, $W_3\_st = broken$, ...

Esempio — Cruciverba

- rappresentazioni in termini di variabili:
  1. *definizione* o *casella numerata* + *direzione* (orizzontale o verticale)
     - dominio: parole di una data lunghezza
       - es., $Due\_orizzontale$ con dominio dato dalle parole di 3 lettere, come $\{'\texttt{ant}','\texttt{big}','\texttt{bus}','\texttt{car}','\texttt{has}'\}$
     - mondo possibile: assegnazione di una parola a ogni variabile
  2. *singola casella*
     - dominio: insieme delle lettere dell'alfabeto
       - ad es., $P00$ casella in alto a sinistra (o destra) con dominio $\{a, \ldots, z\}$
     - mondo possibile: assegnazione di una lettera ad ogni casella

Esercizio — Sudoku ?

Esempio — Guida turistica

- pianificazione delle attività escursionistiche
- due variabili per attività
  - $data$: sui giorni per l'attività
  - $luogo$: sull'insieme delle città da visitare
- mondo possibile: assegnazione di data e luogo a ogni attività

In alternativa:

- *date* come variabili
  - dominio: insieme di tutte le coppie *attività*-*luogo*
- #mondi possibili = prodotto delle cardinalità dei domini delle variabili

---

Esempio — Colorazione Grafi (carte geografiche) con #colori limitato

Esempio — Date A e B con $dom(A) = \{0, 1, 2\}$ e $dom(B) = \{true, false\}$, mondi possibili:

- $w_0 = \{A = 0, B = true\}$
- $w_1 = \{A = 0, B = false\}$
- $w_2 = \{A = 1, B = true\}$
- $w_3 = \{A = 1, B = false\}$
- $w_4 = \{A = 2, B = true\}$
- $w_5 = \{A = 2, B = false\}$

Compattezza della Rappresentazione

n variabili, con domini di cardinalità $d \longrightarrow d^n$ assegnazioni

- *Vantaggio* poche variabili descrivono molti mondi
  - 10 variabili binarie → $2^{10} = 10^3+$
  - 20 variabili binarie → $2^{20} = 10^6+$
  - 30 variabili binarie → $2^{30} = 10^9+$
  - 100 variabili binarie → $2^{100} = 10^{30}+$
    - $10^{30} = 1\ 267\ 650\ 600\ 228\ 229\ 401\ 496\ 703\ 205\ 376$
- ragionare con 30 variabili più facile che con un miliardo di *mondi*/*stati*
  - anche con 100 variabili non ci sono grossi problemi
- *impraticabile* ragionare esplicitamente con 2^{100} stati
- molti *problemi reali* definiti in termini di migliaia / milioni di variabili
  - es. previsioni meteo

In ogni problema: assegnazioni *lecite* | *non lecite*

- **vincolo rigido** / *hard constraint*:
  specifica le assegnazioni ammissibili per una o più variabili

Formalmente:

- **ambito** / *scope*: ins. di variabili
  - sotto-insieme di quelle *coinvolte* nel vincolo
- **relazione** sull'ambito S
  - funzione booleana sulle assegnazioni a variabili in S
    - per distinguere quelle *lecite*
- **vincolo** c: ambito S e relazione su S
  - valutabile su ogni assegnazione che coinvolga tutte le variabili in S:
    dati c su S e l'assegnazione A su S', con $S \subseteq S'$
    - A **soddisfa** c se A, ristretta a S, è $true$ per la relazione
    - A **viola** c in caso contrario

Sintassi e Semantica

Definizione dei vincoli

- **intensionale**: in termini di formule
- **estensionale**: elencando le assegnazioni lecite
  - come tabelle/relazioni di tuple/assegnazioni nei RDB

Soddisfacimento dei Vincoli e *Modelli*

- Un mondo possibile w **soddisfa** un insieme di vincoli se *ognuno* di essi è soddisfatto dai valori assegnati in w alle variabili nel suo ambito
  - in tal caso, w si dice anche loro **modello**

Vincoli e Arietà

- Vincolo *unario*: su singola variabile
  - es $B \leq 3$
- Vincolo *binario*: su coppia di variabili
  - es. $A \leq B$
- In generale, vincolo *n-ario*: ambito di cardinalità n
  - es. $A + B = C$ vincolo ternario

Esempio — Vincoli sulle possibili *date* per *attività* rappresentate da A, B e C tutte con lo stesso dominio $\{1, 2, 3, 4\}$

- vincolo *intensionale* su $\{A, B, C\}$:

$$(A \leq B) \wedge (B < 3) \wedge (B < C) \wedge \neg(A = B \wedge C \leq 3)$$

  - A deve precedere o svolgersi assieme a B
  - B deve svolgersi prima del giorno 3
  - B deve precedere C
  - non è possibile che A e B si svolgano nella stessa data mentre C si svolge prima o giusto nel giorno 3

(..cont.)

- stesso vincolo definito *estensionalmente*:

A	B	C
2	2	4
1	1	4
1	2	3
1	2	4

- $\{A = 1, B = 2, C = 3, D = 3, E = 1\}$ lo soddisfa:
  - $\{A = 1, B = 2, C = 3\}$ corrisponde alla terza riga in tabella ristretta al suo ambito

Esempio — cruciverba

- dominio fatto di *parole*:
  - vincolo: stesse lettere negli incroci (di def. orizzontali e verticali)
- dominio fatto di *lettere*:
  - vincolo: ogni sequenza di lettere contigue forma una parola lecita (nel dizionario)

Un **problema di soddisfacimento di vincoli** (*constraint satisfaction problem*, CSP) consiste in:

- un insieme di *variabili*
  - ognuna con un proprio *dominio*
- un insieme di *vincoli*

CSP finito: numero finito di variabili di dominio finito

- si prenderanno in considerazione, oltre a metodi per CSP finiti, anche algo. per casi con variabili dal dominio infinito o addirittura *continuo*

Esempi — Giochi

- Sudoku
- Criptoaritmetica

Esempio — robot consegne

- ***attività***: a, b, c, d ed e
- ***momenti***: 1, 2, 3 o 4
- ***variabili*** corrispondenti (stesso dominio per tutte):
  $dom(A) = dom(B) = dom(C) = dom(D) = dom(E) = \{1, 2, 3, 4\}$
- insieme di ***vincoli***:

$$\left\{ \begin{array}{l} (B \neq 3), (C \neq 2), (A \neq B), (B \neq C), (C < D), (A = D), \\ (E < A), (E < B), (E < C), (E < D), (B \neq D) \end{array} \right\}$$

Esercizio: trovare un modello (assegnazione valida)

Task tipici con CSP

- Determinare se esista un modello o meno
- Trovare un modello
- Contare il numero di modelli
- Enumerare tutti i modelli
- Trovare il modello migliore, data una misura di qualità
- Determinare se alcuni enunciati siano veri in tutti i modelli

Osservazioni

- CSP *difficili* per il loro carattere multidimensionale
  - una dimensione per variabile
- compito-base: *trovare un modello* (se esiste)
  - CSP con domini finiti: problema *NP-completo*
    - metodi di complessità esponenziale
    - ove possibile si sfrutta la *struttura*

Algoritmo Generate-and-Test

Algoritmo Generate-and-Test

Algoritmo *esaustivo* per CSP finiti

- $\mathcal{D}$: spazio delle *assegnazioni totali*
- l'algoritmo restituisce *uno* o *tutti* i modelli

Per *un* modello:

- si controlla un elemento di $\mathcal{D}$ alla volta
- si restituisce la prima assegnazione che soddisfa *tutti* i vincoli

Per avere *tutti* i modelli:

- si continua a iterare conservando i modelli trovati

Esempio — Nell'esempio precedente

$$\begin{aligned}\mathcal{D} = \{ \quad & \{A=1, B=1, C=1, D=1, E=1\}, \\ & \{A=1, B=1, C=1, D=1, E=2\}, \\ & \quad \vdots \qquad \vdots \qquad \vdots \qquad \vdots \qquad \vdots \\ & \{A=4, B=4, C=4, D=4, E=4\} \quad \}\end{aligned}$$

- $|\mathcal{D}| = 4^5 = 1024$ assegnazioni distinte da testare
- con 15 variabili, 4^{15} ossia circa un miliardo
- con 30 variabili, improponibile

Osservazioni

- n domini di cardinalità d → $\mathcal{D}$ ha cardinalità d^n
- e vincoli → numero totale di test $O(ed^n)$
  - al crescere di n diventa rapidamente intrattabile
  - servono soluzioni alternative

Risoluzione di CSP tramite Ricerca

Risoluzione di CSP tramite Ricerca

Idea: un vincolo coinvolge solo alcune variabili
→ vincoli testabili su assegnazioni *parziali*

- assegnazione *non consistente*[1] rispetto a un vincolo
  → non consistente ogni sua *estensione* che coinvolga altre variabili

[1] Consistenza logica: *coerenza*, *non contraddittorietà*

Esempio — Pianificazione di consegne (es. precedente)

- assegnazioni con $A = 1$ e $B = 1$ non consistenti rispetto a $A \neq B$ indipendentemente dai valori assegnati alle altre
- assegnando prima i valori ad A e B si può anticipare la scoperta di tale non consistenza senza considerare C, D o E e relativi test, risparmiando lavoro

Grafo di Ricerca

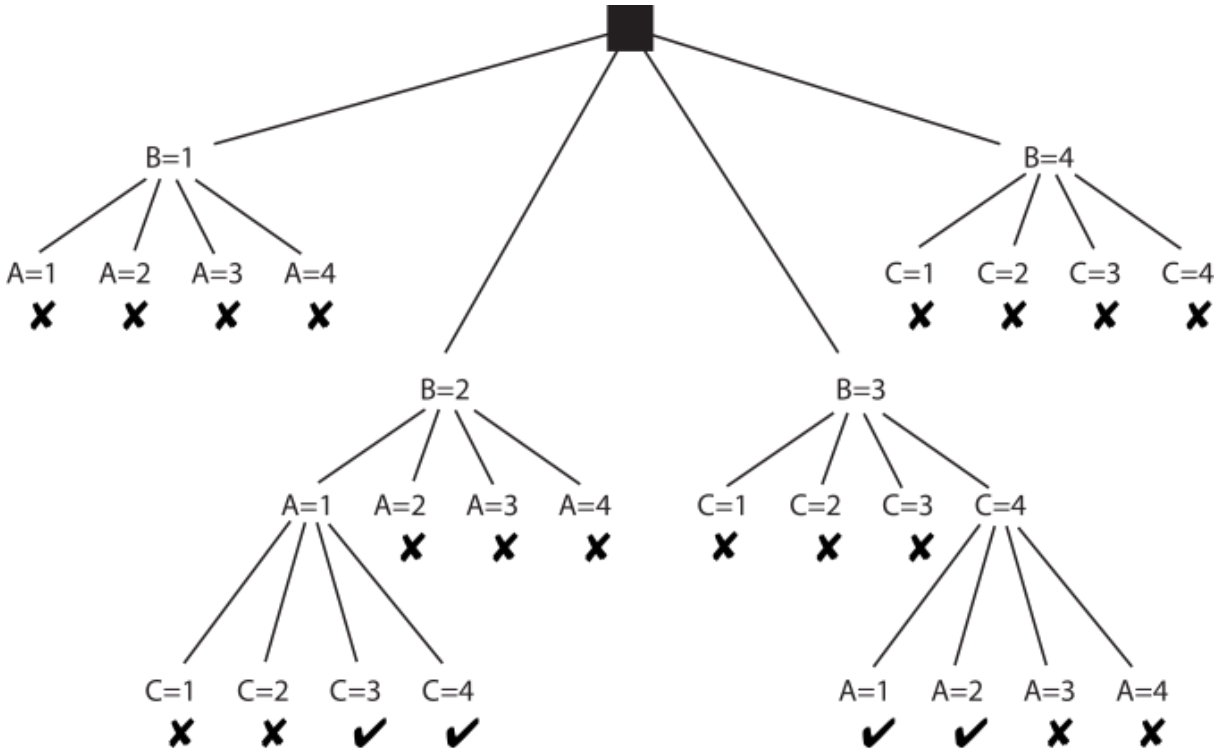
Spazio di ricerca a grafo da costruire:

- *nodi*: assegnazioni parziali n $= \{X_1 = v_1, \ldots, X_k = v_k\}$
- *vicini* di n: assegnazioni *consistenti estese* con assegnazioni a nuove variabili
  - scelti $Y \notin \{X_1, \ldots, X_k\}$ e $y_i \in dom(Y)$,
    n$' = \{X_1 = v_1, \ldots, X_k = v_k, Y = y_i\}$ *vicino* di n se soddisfa tutti i vincoli
- *nodo di partenza*: assegnazione vuota
- *nodi-obiettivo*: assegnazioni totali
- *soluzione*: nodo-obiettivo consistente con tutti i vincoli

in questo contesto, le soluzioni interessano più dei cammini

Esempio — Semplice CSP

- variabili: A, B e C, tutte con dominio $\{1, 2, 3, 4\}$
- vincoli: $A < B$ e $B < C$



possibile albero di ricerca per il CSP

(..cont.)

- "✘": nodo scartato per violazione dei vincoli
  - ad es. il più a sinistra corrisponde a $\{A = 1, B = 1\}$ che viola $A < B$
- 4 soluzioni:
  - ad es. la più a sinistra è $\{A = 1, B = 2, C = 3\}$
- dim. dell'albero (ed efficienza) dipendono dall'*ordine* di scelta delle variabili
  - *statico* — es. sempre prima A poi B poi C — meno efficiente di uno *dinamico*
    - quello ottimale potrebbe essere più difficile da trovare
  - 8 assegnazioni totali e 16 parziali generate di cui si testa la consistenza
    - contro le $4^3 = 64$ del GENERATE-AND-TEST

Osservazioni

- DFS (*backtracking*) molto più efficiente del GENERATE-AND-TEST
  - GENERATE-AND-TEST:
    - test *dopo* aver generato le foglie
  - DFS:
    - test *anticipati* consentono di potare sotto-alberi, risparmiando lavoro

Algoritmi Basati su Consistenza

Esempio — Nell'es. precedente A e B correlate dal vincolo

$$A < B$$

- $A = 4$ non consistente con ogni possibile assegnazione a B
  essendo $dom(B) = \{1, 2, 3, 4\}$
- nella ricerca con backtracking:
  non consistenza *riscoperta* ogni volta per le diverse assegnazioni a B e C
- inefficienza evitabile eliminando 4 da $dom(A)$

Rete di vincoli

Rete di vincoli (*constraint network*) indotta da CSP:

- un nodo circolare per ogni variabile
  - per ogni variabile X, un insieme D_X di possibili valori
    - inizialmente impostato a $dom(X)$
- un nodo rettangolare per ogni vincolo
- un arco $\langle X, c \rangle$ per ogni variabile X nell'ambito del vincolo c

Esempio — sempre nell'es. precedente

- *variabili*: A, B e C, tutte con dominio $\{1, 2, 3, 4\}$
- *vincoli*: $A < B$ e $B < C$
- *rete dei vincoli* corrispondente:

Esempio

- il vincolo $X \neq 4$ ha un arco:
  - $\langle X, X \neq 4 \rangle$
- il vincolo $X + Y = Z$ avrà 3 archi
  - $\langle X, X + Y = Z \rangle$
  - $\langle Y, X + Y = Z \rangle$
  - $\langle Z, X + Y = Z \rangle$

Consistenza degli Archi rispetto ai Domini

Arco $\langle X, c \rangle$ **consistente rispetto al dominio** (*domain consistent*) se $\forall x \in D_X$:

$$X = x \text{ soddisfa } c$$

Esempio — Data B con $D_B = \{1, 2, 3\}$ si consideri il vincolo $B \neq 3$

- arco $\langle B, B \neq 3 \rangle$ non consistente: assegnando 3 a B si viola il vincolo
  - eliminando 3 da D_B diventerebbe consistente

Consistenza degli Archi e delle Reti

Dato il vincolo c su $\{X, Y_1, \ldots, Y_k\}$,
arco $\langle X, c \rangle$ **consistente** se $\forall x \in D_X,\ \exists y_1, \ldots, y_k :\ y_i \in D_{Y_i}$
$c(X = x, Y_1 = y_1, \ldots, Y_k = y_k)$ **soddisfatto**

Una **rete consistente** (rispetto agli archi) contiene solo archi consistenti

Osservazione

- $\langle X, c \rangle$ non consistente
  → per qualche valore di X non ci sono valori di $Y_1, \ldots, Y_k$ che soddisfino c
  - *eliminando* tali valori da D_X si può ripristinare la consistenza di $\langle X, c \rangle$

⚠ l'eliminazione di valori può rendere altri archi *non consistenti*

Esempio — Nella rete precedente

tutti archi non consistenti
dati i domini $\{1, 2, 3, 4\}$:

- $\langle A, A < B \rangle$ perché per $A = 4$ non ci sono valori per B per i quali $A < B$
  - togliendo 4 dal dominio di A, diventerebbe consistente
- $\langle B, A < B \rangle$ perché non c'è un valore per A quando $B = 1$
- ... per **Esercizio**

Algoritmo Basato sulla Consistenza degli Archi

Idea Rendere la rete consistente restringendo i domini

Si considera l'insieme to_do degli archi potenzialmente non consistenti:

- si inizializza to_do con tutti gli archi del grafo
- si ripete fino a svuotare to_do:
  - estrarre $\langle X, c \rangle$ da to_do
  - se $\langle X, c \rangle$ non consistente, restringere il dominio di X
  - aggiungere a to_do gli archi resi non consistenti dal passo precedente:
    - $\langle Z, c' \rangle$, $c' \neq c$, con ambito che comprende X e una diversa Z

→ algoritmo Generalized Arc Consistency

```
procedure GAC(⟨Vs, dom, Cs⟩)
   return GAC2(⟨Vs, dom, Cs⟩, {⟨X, c⟩ | c ∈ Cs ∧ X ∈ scope(c)})

 procedure GAC2(⟨Vs, dom, Cs⟩, to_do)
   while to_do ≠ {} do
      seleziona e rimuovi ⟨X, c⟩ da to_do
      let {Y_1, ..., Y_k} = scope(c) ∖ X
      ND ← {x | x ∈ dom[X] ∧ ∃y_1 ∈ dom[Y_1], ..., y_k ∈ dom[Y_k] :
                        c(X = x, Y_1 = y_1, ..., Y_k = y_k)}
      if ND ≠ dom[X] then
         to_do ← to_do ∪ {⟨Z, c'⟩ | {X, Z} ⊆ scope(c'), c' ≠ c, Z ≠ X}
         dom[X] ← ND
   return dom
```

Esempio — Dato il CSP visto prima con la rete:

Possibile sequenza di selezioni:

- to_do contiene tutti i 4 archi
- estratto $\langle A, A < B \rangle$ da to_do
  - per $A = 4$, non c'è valore di B che soddisfi il vincolo
  - 4 eliminato da D_A
  - nessuna modifica a to_do: tutti gli altri archi già presenti
- estraendo $\langle B, A < B \rangle$
  - 1 eliminato da D_B
  - nessuna modifica a to_do

(..cont.)

- estraendo $\langle B, B < C \rangle$
  - 4 eliminato da D_B
  - $\langle A, A < B \rangle$ aggiunto a to_do
    - potendosi restringere D_A essendo stato ridotto D_B
- estraendo $\langle A, A < B \rangle$
  - 3 eliminato da D_A
- resta da estrarre il solo $\langle C, B < C \rangle$ rimasto in to_do
  - 1 e 2 rimossi dal dominio di C
  - nessun arco aggiunto a to_do
    - C non coinvolta in altri vincoli quindi to_do resta vuoto

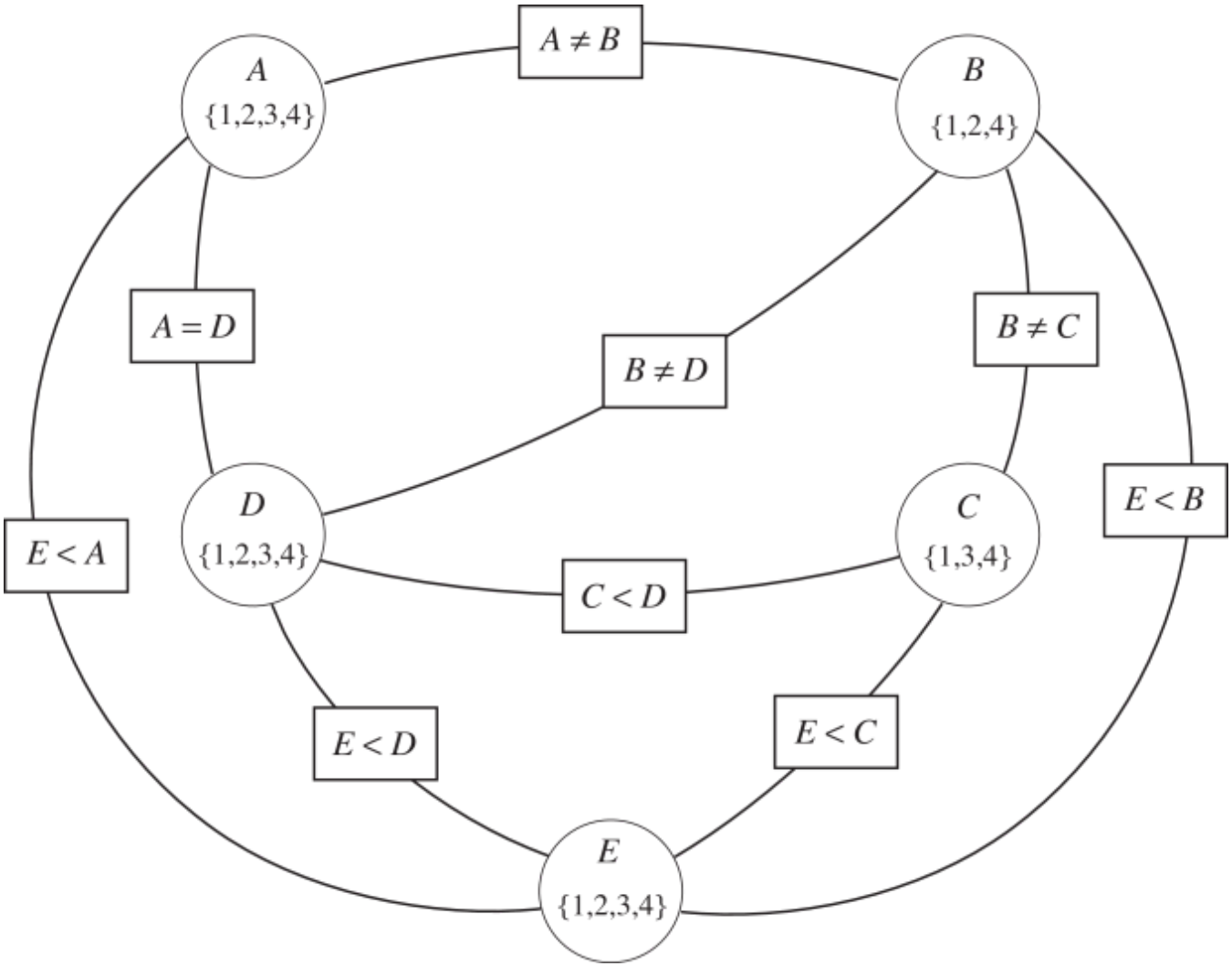
GAC termina con $D_A = \{1, 2\}, D_B = \{2, 3\}, D_C = \{3, 4\}$

- problema non risolto ma *semplificato*
- calcolo soluzione veloce via DFS + backtracking

Esempio — CSP precedente con A, B, C, D, E dallo stesso dominio $\{1, 2, 3, 4\}$ e vincoli:

$$(B \neq 3), (C \neq 2), (A \neq B), (B \neq C),$$
$$(C < D), (A = D), (E < A), (E < B),$$
$$(E < C), (E < D), (B \neq D)$$

(..cont.) la rete sarà:



Il resto nell'Esempio 4.19 del testo

o **Esercizio**

Terminazioni Possibili

GAC termina con una rete consistente con domini *ridotti* e 3 casi possibili:

1. **dominio vuoto → *nessuna soluzione***
    - **se uno è vuoto, lo saranno anche altri domini collegati già prima della terminazione**
2. **domini tutti ridotti a un solo valore → *soluzione unica***
3. **altrimenti, CSP (rete) *semplificato* cui applicare altri metodi**

Metodi Alternativi

Algoritmi basati sulla consistenza dei percorsi

Complessità ∢

- c vincoli binari e domini di cardinalità d
  - $2c$ archi
- controllare $\langle X, r(X,Y) \rangle$ richiede nel caso pessimo di iterare su ogni valore di $dom(Y)$ per ogni valore di $dom(X)$ → tempo: $O(d^2)$
- tale arco potrebbe essere controllato una volta per ogni valore di $dom(Y)$ → GAC per variabili binarie è $O(cd^3)$ in *tempo*
  - lineare in c
- *spazio*: $O(nd)$
  - n numero di variabili

Separazione dei Domini

Separazione dei Domini / Analisi dei Casi

Idea decomporre il CSP in una serie di *casi disgiunti* da risolvere separatamente → soluzioni riunendo quelle trovate per i diversi casi

Esempi

- X binaria, dominio $\{t, f\}$ → due problemi ridotti:
  - trovare le soluzioni con $X = t$ e quelle con $X = f$
  - se ne basta una, secondo caso considerato solo se il primo non ha soluzione
- A con dominio $\{1, 2, 3, 4\}$, separabile in diverse maniere
  - un caso per ciascun valore: $A = 1, A = 2, A = 3, A = 4$
    - fa fare più strada con una sola divisione
  - due sottoinsiemi disgiunti: $A \in \{1, 2\}$ e $A \in \{3, 4\}$
    - taglia di più in meno passi (lavoro che non va rifatto per ogni valore)
  - ...

Schema di Algoritmo

Integrando l'approccio basato sulla consistenza in un algoritmo ricorsivo:

- semplificare il problema in input tramite `GAC( )`
- se non è risolto direttamente:
  - selezionare una variabile, con dominio almeno binario
  - partizionare il dominio ottenendo (2+) problemi semplificati (casi)
  - risolvere *ricorsivamente* tali problemi

```
procedure CSP_Solver(⟨Vs, dom, Cs⟩)
  // restituisce una soluzione al CSP oppure false
  return Solve2(⟨Vs, dom, Cs⟩, {⟨X, c⟩ | c ∈ Cs ∧ X ∈ scope(c)})

procedure Solve2(⟨Vs, dom, Cs⟩, to_do)
  dom_0 ← GAC2(⟨Vs, dom, Cs⟩, to_do)
  if ∃X: dom_0[X] = ∅ then
     return false
  else if ∀X: |dom_0[X]| = 1 then
     return soluzione con ∀X: X = x ∈ dom_0[X]
  else
     selezionare X tale che |dom_0[X]| > 1
     partizionare dom_0[X] in D_1 e D_2
     dom_1 ← copia di dom_0 con dom_1[X] = D_1
     dom_2 ← copia di dom_0 con dom_2[X] = D_2
     to_do ← {⟨Z, c'⟩ | {X, Z} ⊆ scope(c'), Z ≠ X}
     return Solve2(⟨Vs, dom_1, Cs⟩, to_do) or
            Solve2(⟨Vs, dom_2, Cs⟩, to_do)
```

Stesso algoritmo per avere *tutte* le soluzioni:

- dominio vuoto → insieme vuoto / $\perp$
- dominio con un solo valore → singoletto
- ritorno dalla ricorsione restituendo l'unione delle soluzioni dei 2 casi

Osservazione — spazio per algoritmi di ricerca su grafo ma contano le soluzioni

- e.g. DFS, con spazi finiti

Miglioramento possibile:

- se un'assegnazione rende il grafo *non connesso*,
  ogni componente può essere risolta *separatamente*
  - soluzione ottenuta ricombinando le soluzioni delle componenti
  - conteggio delle soluzioni efficiente
    - ad es., una componente con 100 soluzioni,
      altra con 20 → 2000 soluzioni totali

Eliminazione di Variabili

Eliminazione di Variabili

Eliminazione di variabili (*variable elimination*, VE):
tecnica che semplifica la rete dei vincoli rimuovendo variabili

Idea eliminare le variabili una alla volta ottenendo problemi semplificati e infine ricostruire le soluzioni dei problemi più complessi

- eliminando X si costruisce un *nuovo vincolo* sulle rimanenti che rifletta gli effetti di X
  - sostituisce tutti gli altri vincoli su X
    → rete semplificata (CSP ridotto)
- alla fine, ogni soluzione del CSP ridotto va *estesa* per ottenere una soluzione del CSP comprendente X

Eliminazione di una Variabile

Data X da eliminare:

1. considerate le relazioni di tutti i vincoli su X,
   sia $r_X(X, \bar{Y})$ il `join` di tali relazioni
   - $\bar{Y}$: ins. delle altre variabili nell'ambito di r_X

   influenza di X sulle altre *vicine* di X nel grafo dei vincoli
2. la *proiezione* di r_X su $\bar{Y}$ sostituisce tutte le relazioni in cui occorre X
3. si ottiene un CSP ridotto, senza X, da risolvere *ricorsivamente*:
   - al *ritorno*, tabelle-soluzioni per il CSP ridotto vanno estese, `join` con r_X, per aggiungere la colonna delle assegnazioni a X
   - *caso base*: resta una sola variabile → soluzione da restituire
     - = tabella con i valori del dominio consistenti con i suoi vincoli

Esempio — Si consideri un CSP su A, B, C di dominio $\{1, 2, 3, 4\}$; sia B da eliminare, inclusa nei vincoli: $A < B$ e $B < C$

- altre variabili possibili ma non coinvolte in tali vincoli
- per eliminare B, `join` tra le relazioni dei vincoli su B:

A	B
1	2
1	3
1	4
2	3
2	4
3	4

$\bowtie$

B	C
1	2
1	3
1	4
2	3
2	4
3	4

$=$

A	B	C
1	2	3
1	2	4
1	3	4
2	3	4

(..cont.)

- la sua proiezione su A e C induce una nuova relazione senza B:

A	C
1	3
1	4
2	4

  vincolo che sostituisce tutti quelli su B
  - contenendo tutte le info utili alla soluzione del resto della rete
- VE poi risolve il resto della rete semplificata
- Per avere una/tutte le soluzioni a partire dalla soluzione del CSP ridotto:
  - si memorizza la relazione del `join` su A, B, C per estendere la soluzione della rete ridotta includendo B

```
procedure VE_CSP(Vs, Cs)
  Input
     Vs: insieme di variabili
     Cs: insieme di vincoli su Vs
  Output
     relazione contenente tutte le assegnazioni consistenti alle
     variabili

  if |Vs| = 1 then
     return join di tutte le relazioni in Cs
  else
     Selezionare X ∈ Vs da eliminare
     Vs' ← Vs \ {X}
     Cs_X ← {c ∈ Cs | c coinvolge X}
     Sia R il join di tutti i vincoli in Cs_X
     Sia R' la proiezione di R sulle variabili di Vs'
     S ← VE_CSP(Vs', (Cs \ Cs_X) ∪ {R'})
     return R ⋈ S
```

Osservazioni

- caso *base*: rimane 1 variabile
  - una soluzione esiste se ci sono righe nelle relazioni finali
    - tutte su una sola variabile (insiemi di valori leciti) basta intersecarle
- caso *ricorsivo*: l'ordine di selezione delle variabili ha un impatto sull'efficienza
  - *al ritorno*: se bastasse una soluzione, restituire solo una tupla di $R \bowtie S$
    - è garantito che sia parte d'una soluzione
      - se un valore di R non avesse tuple, non ci sarebbero soluzioni con tale valore

Estensioni

VE può essere *combinato* con algoritmi basati su consistenza:

- usati per semplificare il problema quando si elimina una variabile
  - tabelle intermedie più piccole

Complessità ∢

L'efficienza dipende dall'*ordine* di selezione delle variabili

- struttura intermedia — variabili delle relazioni intermedie — dipendente solo dalla struttura del grafo dei vincoli
  - in generale, rete sparsa → VE efficiente

Treewidth del grafo:

- numero di variabili nella più grande relazione d'ordinamento (o *tree decomposition*)
- si considera la *minima* treewidth al variare degli ordinamenti

Complessità di VE:

- *esponenziale* rispetto alla treewidth
- *lineare* nel numero di variabili
  - esponenziale con gli algoritmi di ricerca precedenti

(..cont.)

Trovare un ordine che minimizzi la treewidth minima è *NP-hard*,
ma ci sono *euristiche*:

- **min-factor**: selezionare la variabile che porta alla *relazione più piccola*
- **minimum deficiency** o **fill**: selezionare la variabile che aggiunge meno archi alla rete dei vincoli
  - *deficiency* di X: numero di coppie di variabili in una relazione con X che non sono in relazione fra loro
  - *idea*: rimuovere una variabile che non porti a una relazione grande purché non renda la rete più complicata
  - produce treewidth minori ma più difficile da calcolare

Ricerca Locale

Spazi molto grandi o *infiniti*?

- no ricerca sistematica dell'intero spazio
- metodi *mediamente efficienti* per trovare soluzioni
  - *senza garanzie*, anche quando esistono
  - utili quando si sa già che esistono (verosimilmente)

→ Metodi di **ricerca locale**

- Molte tecniche: campo comune tra *Ricerca Operativa* e *AI*
- Iniziano con un'assegnazione totale di un valore a ciascuna variabile
- Tentano di migliorare l'assegnazione iterativamente
  - passi di *miglioramento*
  - passi *casuali*
  - *ripartenze* con assegnazioni differenti

```
procedure Local_search(Vs,dom,Cs)
   Input
      Vs: insieme di variabili
      dom: funzione che restituisce il dominio di una variabile
      Cs: insieme di vincoli da soddisfare
   Output
      assegnazione totale che soddisfa i vincoli
   Local
      A array di valori indicizzato sulle variabili in Vs
      (assegnazione)
   repeat // try
      for each X ∈ Vs do
         A[X] ← valore casuale da dom(X)
      // walk
      while not stop_walk() and A non soddisfa Cs do
         Selezionare Y ∈ Vs e un valore w ∈ dom(Y)
         A[Y] ← w
      if A soddisfa Cs then
         return A
   until terminazione
```

- ogni iterazione della `repeat` rappresenta un **tentativo** (*try*)
  - primo `for each`: **inizializzazione** casuale di A
  - assegnazioni casuali successive: **ripartenza casuale** (*random restart*)
    - in alternativa, congetture più informate
      - euristiche o conoscenza pregressa, poi migliorata iterando
- ciclo `while`: **ricerca locale** (*walk*) nello spazio delle assegnazioni
  - **NB** seleziona un'assegnazione tra i possibili *successori* di A
    - differiscono per il valore assegnato a una sola variabile
  - stop quando: soluzione trovata o si avvera il criterio di `stop_walk()`
    - es. semplice: raggiunto numero max di cicli
- fermata *non garantita*: può divergere se non c'è soluzione
  - anche se ce ne fossero, potrebbe rimanere intrappolato in una regione
  - *completezza*: dipende dai criteri di selezione e di stop

Random Sampling

- `stop_walk()` sempre `true` → `while` mai eseguito
  - continua indefinitamente a provare assegnazioni casuali che possano soddisfare tutti i vincoli
- algoritmo *completo*
  - garantisce la soluzione se esiste
    - ma tempo richiesto non limitato!
    - tipicamente risulta molto lento
- efficienza dipende da:
  - (prodotto delle) dimensioni dei domini
  - numero di soluzioni

Random Walk

- `stop_walk()` sempre `false` → no ripartenze casuali
  - esce dal ciclo `while` solo quando trova una soluzione
  - nel ciclo seleziona casualmente una variabile e un valore da assegnarle
- algoritmo *completo*
  - passi più veloci rispetto al resampling di tutte le variabili
  - ma può richiedere più passi, in base alla distribuzione delle soluzioni
- quando le dimensioni dei domini delle variabili differiscono, si può
  - selezionare a caso una variabile, quindi un valore del suo dominio

    oppure
  - selezionare casualmente una coppia variabile-valore
    - favorisce la selezione di variabili con dominio più grande

Massimo Miglioramento Iterativo

ITERATIVE BEST IMPROVEMENT — ricerca locale che seleziona il *miglior successore* di un'assegnazione in termini di una **funzione di valutazione**

- funzione da minimizzare / massimizzare:
  GREEDY DESCENT / GREEDY ASCENT (noto anche come HILL CLIMBING)
  - basta implementare uno solo dei due obiettivi
    - per l'altro, sufficiente cambiare il segno della funzione
  - a *parità* di valore (*tie*) → scelta casuale
- funzione di valutazione tipica:
  - numero di **conflitti**, i.e. vincoli violati
    - 0 conflitti → soluzione
  - si può raffinare *pesando* i vincoli in maniera differenziata

OTTIMALITÀ

Si distinguono:

- **ottimi locali** assegnazioni non migliorabili da alcun successore
  - *minimi* / *massimo locali* da GREEDY DESCENT / ASCENT
- **ottimi globali** con valutazione massima fra tutte le assegnazioni
  - sempre anche ottimi locali

Minimizzando (una funzione de) il *numero di conflitti*:

- CSP *soddisfacibile* ← minimo globale con valore nullo
- CSP *non soddisfacibile* ← minimo globale con valore positivo

se si raggiunge un minimo locale con valutazione *positiva*,
NON è detto che sia globale (nel caso, CSP non soddisfacibile)

Esempio — ancora esempio precedente sulle consegne:

- se GREEDY DESCENT inizia da $\{A = 2, B = 2, C = 3, D = 2, E = 1\}$
  - 3 conflitti: $A \neq B$, $B \neq D$, $C < D$
- nel possibile successore $B = 4$
  - 1 conflitto: $C < D$
- possibile successore con conflitti minimi: $D = 4$
  - 2 conflitti
- poi, con il successore con $A = 4$
  - 2 conflitti
- infine, con $B = 2$
  - 0 conflitti $\longrightarrow$ *soluzione*

(..cont.)

trace:

A	B	C	D	E	val
2	2	3	2	1	3
2	4	3	2	1	1
2	4	3	4	1	2
4	4	3	4	1	2
4	2	3	4	1	0

con diverse inizializzazioni o diverse scelte in caso di pari valutazione possibili sequenze di assegnazioni e risultati differenti

Completezza

Osservazione — si considera il *miglior successore* anche quando questo *non ha* una migliore valutazione rispetto all'assegnazione corrente

- e.g. minor numero di conflitti

Caso possibile: ottimi locali che risultano successori *reciproci*

- l'algoritmo continua a passare da uno all'altro
- non potrà trovare una soluzione → *incompletezza*

Obiettivo: evitare minimi locali che non siano anche globali

Idea: usare la *casualità* per *evitare* tali minimi

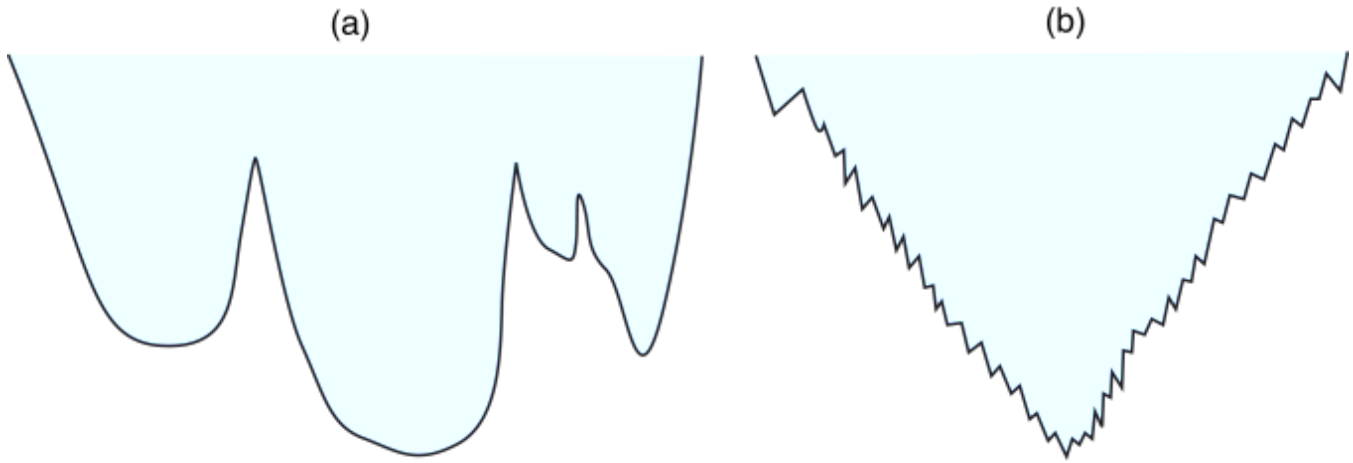
1. RANDOM RESTART — valori scelti a caso per tutte le variabili:
   - mossa casuale *globale* (più costosa)
   - per poter ripartire da regioni anche completamente diverse dello spazio
2. RANDOM WALK — mosse casuali alternate a passi di ottimizzazione:
   - mossa casuale *locale*
   - GREEDY DESCENT / ASCENT permettono passi in direzione opposta per sfuggire a minimi / massimi locali

Integrando *massimo miglioramento iterativo* + mosse *casuali*:

→ **Ricerca Locale Stocastica**

Esempio — *Spazio 2D*

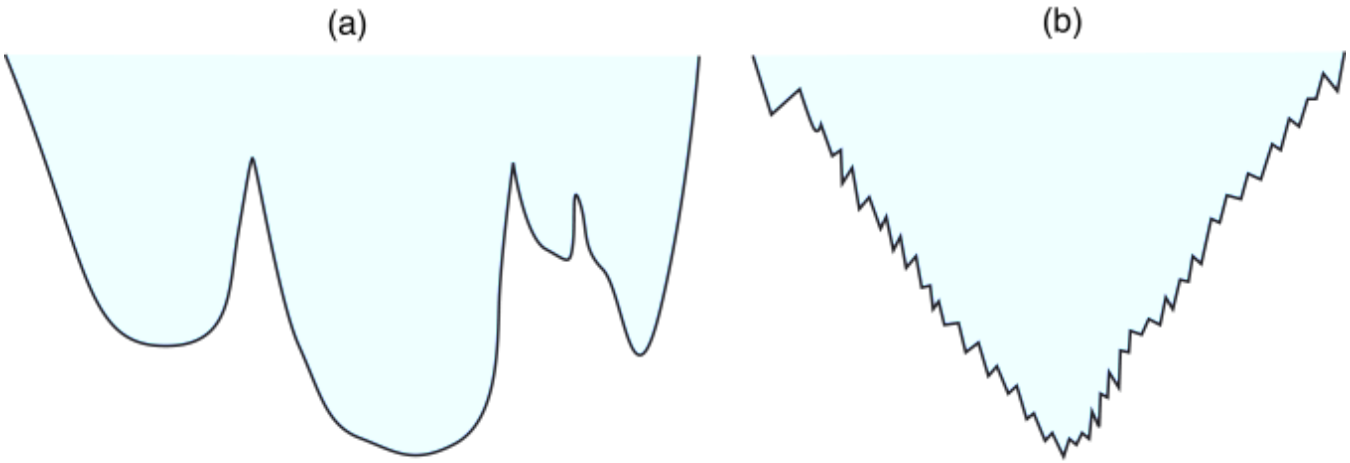
- *successore* tramite piccolo passo dall'attuale posizione
  - verso sinistra o destra



- spazio di ricerca (a)
  - GREEDY DESCENT può trovare facilmente minimi locali
    - serve un RANDOM RESTART che porti nella parte centrale (più profonda) nella quale si converge rapidamente verso uno globale
  - RANDOM WALK non funzionerebbe bene
    - richiederebbe molti piccoli passi casuali per uscire da un minimo locale

(..cont.)

- spazio di ricerca (b)
  - RANDOM RESTART rimane bloccato a cercare tra numerosi minimi locali
  - RANDOM WALK con GREEDY DESCENT potrebbe evitare tali minimi locali:
    - pochi passi casuali spesso sufficienti



Possibili spazi con caratteristiche diverse in regioni diverse

Varianti della Ricerca Locale

Molte altre varianti possibili nella *scelta del successore* + *casualità*

Successori e Domini

- domini *piccoli*: tutti i valori possono essere scelti per i successori
- domini *estesi*: si considerano solo alcuni valori (risparmiando tempo)
  - spesso i *più vicini*
  - possibili metodi più sofisticatati di selezione

TABU SEARCH

TABU SEARCH evita la modifica di assegnazioni introdotte di recente

idea memorizzare variabili modificate negli ultimi t passi (**tenure**)
da considerare *non selezionabili*

- evita i cicli dopo poche assegnazioni
- t parametro da ottimizzare
  - *piccolo*: lista delle variabili modificate di recente
  - *grande*: si memorizza per ogni variabile il passo relativo all'ultimo cambiamento

Passo di Massimo Miglioramento

Metodo che seleziona una coppia variabile-valore che porta al *miglioramento* di valutazione *massimale*

- più coppie → scelta casuale

Implementazione ingenua
data l'assegnazione totale corrente:

- per ogni X e ogni $v \in dom(X)$ diverso da quello corrente, confrontare l'assegnazione corrente con quella in cui $X = v$
- selezionare una coppia di *massimo miglioramento*
  - anche in caso di *differenze negative* (peggioramenti)
  - variabili senza vincoli possono essere tralasciate
  - un passo → $O(ndr)$ valutazioni
    - n numero di variabili
    - d cardinalità max dei domini
    - r numero di vincoli per variabile

Implementazione alternativa
con *coda con priorità* di coppie pesate variabile-valore

- per ogni X e ogni $v \in dom(X)$ non presenti nell'assegnazione corrente, in coda $\langle X, v \rangle$ con *peso* w
  - miglioramento dell'assegnazione con $X = v$ rispetto a quella corrente
    - dipende dai valori assegnati a X e suoi vicini nella rete dei vincoli, non da quelli assegnati alle altre
- a ogni iterata si seleziona una *coppia-successore* di massimo miglioramento
  - *peso minimale*
- nuova assegnazione → ricalcolo-pesi → riordinamento della coda
  - ma <u>solo</u> per coppie con variabili in vincoli il cui soddisfacimento è mutato

Complessità ∢

- dimensione della coda $n(d-1)$
  - n numero variabili
  - d dim. medie del dominio
- inserimento/rimozione: $O(\log(nd))$
- l'algoritmo rimuove un solo elemento dalla coda, ne aggiunge un altro e aggiorna i pesi di rk variabili al più
  - r numero di vincoli per variabili
  - k numero di variabili per vincolo
- complessità di un passo: $O(rkd\log(nd))$
  - molto tempo speso nel gestire le strutture dati

Scelta a Due Fasi

Idea:

1. selezione della variabile
2. selezione del valore

Si gestisce una *coda* con priorità di variabili
con peso pari al *numero dei conflitti* in cui sono coinvolte

- ad ogni passo:
  - si seleziona X che partecipa a *più conflitti*
  - si cambia il valore assegnato:
    - minimizzando il numero di conflitti

      oppure casualmente
  - ricalcolo-pesi per variabili in vincoli il cui soddisfacimento è mutato

Complessità ∢

- ogni passo: $O(rk \log n)$
  - meno lavoro rispetto all'algoritmo basato su coppie variabile-valore
- *compromesso*:
  - più passi nell'unità di tempo
  - meno costosi ma meno migliorativi

Algoritmo Any Conflict

Idea: si sceglie una **variabile conflittuale** (i.e. partecipa a conflitti)

A ogni iterata:

- si seleziona casualmente una *variabile conflittuale*
  - non necessariamente quella con più conflitti
- quindi le si assegna, in alternativa:
  - un valore che *minimizzi* il numero di conflitti

    oppure
  - un valore casuale

Varianti

Criterio di selezione casuale della variabile:

1. si sceglie prima un conflitto, poi una variabile coinvolta
2. scelta di una variabile conflittuale

Differenze — probabilità di selezione di una variabile

1. dipende dal numero di conflitti in cui è coinvolta
2. stessa probabilità per tutte

Complessità ◁

- gestione di strutture dati per la rapida selezione casuale di una variabile
  - *variante 1*: *insieme di conflitti* da cui selezionare un elemento casuale
    - es. con un albero binario di ricerca
    - complessità di un passo: $O(r \log c)$
      - caso pessimo: r vincoli da aggiungere/rimuovere dall'ins. dei conflitti

Simulated Annealing

Metafora dalla *termodinamica* | *metallugia*:

- *alta* temperatura → maggiore casualità / plasticità
- *bassa* temperatura → minore casualità / maggiore durezza

Metodo senza strutture dati ausiliari:

- seleziona casualmente una variabile e un valore del suo dominio
- quindi *accetta* / *rigetta* la nuova assegnazione risultante

SIMULATED ANNEALING [3] riduce lentamente la *temperatura*:

- alle *alte temperature* si comporta come **RANDOM WALK**
  - consente di saltare i minimi locali e
    trovare regioni con bassi valori dell'euristica
    - passi *peggiorativi* più probabili ad alte temperature
- alle *basse temperature* si comporta come il **GREEDY DESCENT**
  - porta direttamente verso i minimi (locali)

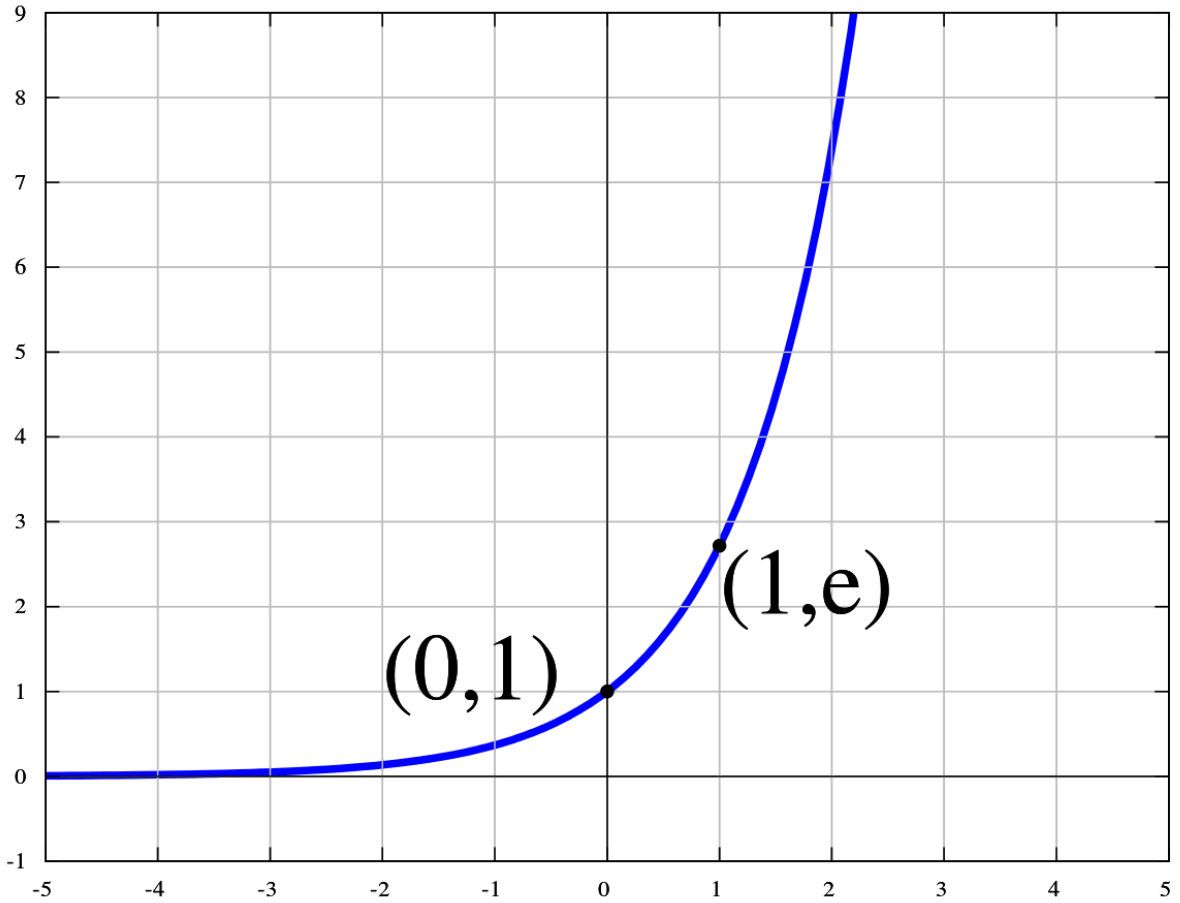
A ogni passo, data l'assegnazione *corrente* A:

- si sceglie a caso una variabile e un valore
  ottenendo una *nuova* assegnazione A'
- se A' non peggiora l'euristica
  - sostituisce l'assegnazione corrente
  - altrimenti lo fa con una *probabilità*
    che dipende dalla temperatura e dal peggioramento che comporta

Temperatura $T \in \mathbb{R}_+$

- $h(A)$ *euristica* da minimizzare
  - (tipicamente) numero dei conflitti
- se $h(A') \leq h(A)$, si accetta direttamente: $A \leftarrow A'$
- altrimenti, si può accettare con *probabilità*

$$e^{-(h(A')-h(A))/T}$$



distribuzione di Gibbs / Boltzmann

- se $h(A') > h(A)$ allora esponente negativo
- tendendo $h(A') - h(A) \to 0$, più probabile accettare A'
  - alte temperature: esponente $\to 0$ e probabilità $\to 1$
  - al decrescere della temperatura: esponente $\to -\infty$ e probabilità $\to 0$

Esempio — Probabilità di accettazione di passi peggiorativi a diverse temperature e differenze $k = h(A') - h(A)$:

Temperatura	$k = 1$	$k = 2$	$k = 3$
10	0.9	0.82	0.74
1	0.37	0.14	0.05
0.25	0.018	0.0003	0.000006
0.1	0.00005	$0.2 \cdot 10^{-8}$	$0.9 \cdot 10^{-13}$

Programma di annealing

specifica come ridurre la temperatura al progredire della ricerca

- *raffreddamento geometrico*: molto usato
  - ad es. si parte da $T = 10$ e si moltiplica per 0.99 ad ogni passo
    - arrivando a 0.07 dopo 500 passi

Osservazioni

- temperature ***alte*** (e.g. $T = 10$):
  - si tende ad accettare passi che peggiorano di poco
    - leggera preferenza rispetto a passi che migliorano
- temperature ***ridotte*** (e.g. $T = 1$):
  - passi peggiorativi accettati molto meno di frequente
- temperature ***basse*** (e.g. $T = 0.1$):
  - passi peggiorativi accettati molto raramente

(sola *lettura*)

Valutazione empirica comparativa basata sulla **run-time distribution**

- distribuzione *cumulativa*:
  quante volte il problema è stato risolto entro un dato numero di passi
  - o in un lasso di tempo

Random Restart o ripartenza casuale

- permette di migliorare le prestazioni di un *algoritmo casuale debole*: ha successo in pochi casi specifici
  - p probabilità di successo d'una singola esecuzione
- per stimarne le prestazioni di una *sequenza* di n esecuzioni *indipendenti*, probabilità di successo:

$$1 - (1 - p)^n$$

  - n esecuzioni necessarie al ritrovamento di una soluzione
  - fallisce, con probabilità $(1 - p)^n$, solo se falliscono *tutti* i tentativi

Esempio — algoritmo con $p = 0.5$

- ripetuto per 5 volte, trova una soluzione circa il 96.9% delle volte
- ripetuto 10 volte: 99.9%

Se $p = 0.1$

- ripetuto 10 volte: 65% di percentuale di successo
- ripetuto 44 volte: 99%

Osservazioni

RANDOM RESTART *costoso* se sono coinvolte molte variabili

- nel PARTIAL RESTART si fanno assegnazioni solo ad *alcune* variabili, per spostarsi verso un'altra regione
  - e.g. una data *percentuale* di esse (es. 10%)
- tentativi/esecuzioni *non indipendenti* → analisi teorica più complessa

Algoritmi Basati su Popolazioni

Algoritmi Basati su Popolazioni

Metodi che gestiscono **popolazioni** (insiemi) di **individui** (assegnazioni):

- *beam search*: i migliori k
  - *beam search stocastica*: numero variabile aleatoriamente
- algoritmi *genetici*: i migliori k ai fini riproduttivi

Beam Search

Simile al miglioramento iterativo

- conserva fino a k assegnazioni anziché una sola
- *successo* quando viene trovata una soddisfacente
- a ogni passo, si selezionano i migliori k *successori*
  - anche meno qualora non ce ne fossero a sufficienza
  - selezione casuale in caso di parità
- si itera con il nuovo set di k assegnazioni

Osservazioni

- utile in caso di *memoria limitata*
  - k selezionato in base alla memoria disponibile
- possibili *varianti*:
  - impiegare più tempo nel cercare i migliori k
  - impiegare meno tempo puntando ad approssimazioni (stime)

Beam Search Stocastica

- Si selezionano k individui *casualmente*
  - favorendo (maggiore probabilità) quelli con valutazione migliore
- Probabilità di scelta in funzione dell'euristica
  - selezione di un individuo A con probabilità proporzionale a

$$e^{-h(A)/T}$$

distribuzioni di Gibbs / Boltzmann

- $h(A)$ funzione di valutazione
- T temperatura

Osservazione: tende a consentire più *diversità* nella popolazione

- h riflette l'*adattamento*/*fitness*
  - (come in *biologia*) un individuo più *adatto* ha più probabilità di passare i propri geni a future generazioni
    - *riproduzione asessuata*:
      - un individuo produce una prole leggermente *mutata*
  - si adotta un principio di *sopravvivenza dei più adattabili*
- stessi individui selezionabili casualmente anche più volte

Analoga metafora biologica:

- assegnazione = patrimonio genetico d'un individuo
- nuovi individui della popolazione dalla *combinazione* di *coppie* di individui-*genitori* della generazione precedente

Crossover: operazione che prevede:

- *selezione* di una coppia di individui
- *generazione* della loro prole
  - copiando parte delle assegnazioni alle variabili da un genitore e il resto dall'altro

NB operazione *aggiuntiva* rispetto alla mutazione

Si gestisce una *popolazione* di k individui (con k pari)

Fino a trovare una soluzione:

- A ogni iterata, *generazione* di k *nuovi* individui
  - Selezione casuale di coppie:
    - favorendo la selezione degli individui più adatti
    - la probabilità dipende dall'incremento di fitness e dalla temperatura
  - Per ogni coppia, si opera un *crossover*
  - Si fanno *mutare* casualmente alcuni (pochissimi) valori,
    per alcune variabili scelte a caso
    - RANDOM WALK
  - Si passa alla successiva generazione

```
procedure Algoritmo_Genetico(Vs,dom,Cs,S,k)
  Input
     Vs: insieme di variabili
     dom: dominio della variabile come funzione
     Cs: insieme di vincoli da soddisfare
     S: programma di raffreddamento della temperatura
     k: dim. popolazione - intero pari
  Output
     assegnazione totale che soddisfano i vincoli
  Locali
     Pop: insieme di assegnazioni
     T: real
```

```
Pop ← k assegnazioni totali casuali
T inizializzato secondo S
repeat
    if A ∈ Pop soddisfa tutti i vincoli in Cs then
        return A
    Npop ← ∅
    repeat k/2 volte
        A1 ← Random_selection(Pop, T)
        A2 ← Random_selection(Pop, T)
        N1, N2 ← Crossover(A1, A2)
        Npop ← Npop ∪ {mutate(N1), mutate(N2)}
    Pop ← Npop
    T viene aggiornato secondo S
until terminazione
```

procedure Random_selection(Pop, T)
  selezionare A da Pop con probabilità proporzionale a $e^{-h(A)/T}$
  return A
procedure Crossover(A_1, A_2)
  selezionare casualmente un intero i, $1 \leq i < |Vs|$
  $N_1 \leftarrow \{(X_j = v_j) \in A_1 \mid j \leq i\} \cup \{(X_j = v_j) \in A_2 \mid j > i\}$
  $N_2 \leftarrow \{(X_j = v_j) \in A_2 \mid j \leq i\} \cup \{(X_j = v_j) \in A_1 \mid j > i\}$
  return N_1, N_2

Crossover

- **crossover uniforme**: considera due individui *genitori* e genera due *figli*
  - nei figli, casualmente per ogni variabile, valore copiato da uno dei genitori
- **one-point crossover**, metodo molto comune: assume variabili *ordinate*
  - seleziona casualmente un *indice* i
  - produce un figlio selezionando i valori per le variabili fino a i da un genitore e per le successive ($> i$) dall'altro
    - per l'altro figlio si agisce in modo *complementare*
  - *efficacia* dipendente dall'ordinamento scelto:
    - in fase di progettazione dell'algoritmo

Esempio — Nell'esempio precedente,
funzione di costo basata sul numero di conflitti

- $A = 2, B = 2, C = 3, D = 1, E = 1$ → costo: 4
  - basso grazie a $E = 1$
  - un discendente che erediti $E = 1$ tenderà ad avere un costo più basso
    - sopravvivenza più probabile
- Altri individui con valutazione bassa
  - $A = 4, B = 2, C = 3, D = 4, E = 4$ → costo: 4
    - a causa delle assegnazioni su A, B, C, D
    - i figli che preservano questa proprietà saranno più adatti rispetto agli altri, candidandosi alla sopravvivenza
- Combinando questi individui, tra i discendenti:
  - alcuni erediteranno cattive proprietà e non saranno scelti per tramandarle
  - altri quelle buone e avranno maggiori probabilità di sopravvivenza

Ottimizzazione

Problema di ottimizzazione — trovare i *migliori* mondi possibili

Dati:

- un insieme di *variabili* con un *dominio* associato
- una **funzione-obiettivo** dalle assegnazioni totali a $\mathbb{R}$
- un **criterio di ottimalità**
  - tipicamente minimizzare/massimizzare la funzione-obiettivo

Trovare: un'assegnazione totale **ottimale** per il criterio adottato

Problema di ottimizzazione vincolato comprende anche *vincoli rigidi* che specificano le assegnazioni possibili ammissibili

- *Obiettivo*: assegnazione ottimale che soddisfi i vincoli rigidi

---

Vasta letteratura scientifica

- molte tecniche:
  - es. *programmazione lineare* con variabili reali e funzione-obiettivo lineare e diseguaglianze lineari come vincoli
- se il problema da risolvere rientra nelle *categorie classiche*, meglio usare algoritmi specifici
  - anche dopo qualche trasformazione

Problema di ottimizzazione di vincoli:
funzione-obiettivo fattorizzata in un insieme di **vincoli flessibili**

- vincoli con un *ambito* di variabili
- funzioni di **costo**:
  - dai domini delle variabili a $\mathbb{R}$
- criterio di ottimalità:
  - tipicamente *minimizzazione della somma* dei costi dei vincoli flessibili

Esempio — Caso precedente ma con preferenze sui tempi invece dei vincoli rigidi: costi associati alle *combinazioni* di valori (tempi)

- scopo: trovare una *disciplina* con somma totale dei costi minimale
- date A, C, D ed E con dominio $\{1, 2\}$ e B con dominio $\{1, 2, 3\}$
- vincoli flessibili:

c_1 :

A	B	$costo$
1	1	5
1	2	2
1	3	2
2	1	0
2	2	4
2	3	3

c_2 :

B	C	$costo$
1	1	5
1	2	2
2	1	0
2	2	4
3	1	2
3	2	0

c_3 :

B	D	$costo$
1	1	3
1	2	0
2	1	2
2	2	2
3	1	2
3	2	4

- nel seguito si considereranno anche $c_4(C, E)$ e $c_5(D, E)$

Somma *puntuale* di vincoli flessibili → vincolo flessibile con:

- ***ambito***: unione degli ambiti
- ***costo*** di un'assegnazione alle variabili nell'ambito:
  somma dei costi delle assegnazioni nei vincoli flessibili

Esempio — Dati $c_1(A, B)$ e $c_2(B, C)$ dell'es. precedente:

- $c_1 + c_2$ funzione con ambito $\{A, B, C\}$, dato da

$c_1 + c_2$:

A	B	C	$costo$
1	1	1	10
1	1	2	7
1	2	1	2
⋮	⋮	⋮	⋮

  - e.g. $(c_1 + c_2)(A = 1, B = 1, C = 2)$
    $= c_1(A = 1, B = 1) + c_2(B = 1, C = 2) = 5 + 2 = 7$

Soddisfacimento di Vincoli e Ottimizzazione

Differenza — Rispetto ai CSP, problemi che presentano un'ulteriore difficoltà:

- sapere quando un'assegnazione è una soluzione
  - CSP: basta controllare se l'assegnazione soddisfa tutti i vincoli (hard)
  - ottimizzazione: solo per *confronto* con altre assegnazioni

---

- Vincoli rigidi come quelli flessibili ma costo *infinito* in caso di violazione
  - costo finito → nessuna violazione
- Alternativa: costo *elevato* associato alla violazione di vincolo rigido
  - maggiore della somma di tutti i vincoli soft
- Ottimizzazione per trovare una soluzione con
  - il minor numero di vincoli rigidi violati
    e, tra questi, quelli di costo minimo

Metodi per CSP adattabili a problemi di ottimizzazione:

- GENERATE-AND-TEST:
  - somma dei vincoli e selezione assegnazione con il minimo costo
    - solo per problemi semplici
- algoritmo di *consistenza* generalizzato ⊀
- separazione di domini ⊀
- *eliminazione* di variabili

Eliminazione di Variabili per l'Ottimizzazione

Eliminazione di una variabile alla volta, e.g. X:

- sia R l'insieme dei vincoli che coinvolgono X
- c_T, vincolo somma dei vincoli in R con relazione T
- c_N nuovo vincolo con relazione N e ambito ridotto $V = scope(T) \setminus \{X\}$
  - per ogni valore delle variabili in V:
    - selezionare un valore di X che minimizzi T
  - c_N sostituisce i vincoli in R
- si ha un problema ridotto con (meno variabili e) nuovo insieme di vincoli, da risolvere *ricorsivamente*
  - soluzione S del problema ridotto: assegnazione su V
    - quindi $c_{T(S)}$ vincolo con rel. T sotto S, è in funzione di X
    - valore ottimale per X ottenuto scegliendo un valore che porti al minimo valore per $c_{T(S)}$

```
procedure VE_SC(Vs, Cs)
  Input
     Vs: insieme di variabili
     Cs: insieme di vincoli flessibili
  Output
     assegnazione ottimale per Vs
  if Vs contiene un solo elemento o Cs contiene un solo
  vincolo then
     sia c_C la somma dei vincoli in Cs
     return assegnazione con il minimo costo in c_C
  else
     selezionare X ∈ Vs secondo un ordine di eliminazione
     R ← {C ∈ Cs | X ∈ scope(C)}
     sia c_T la somma dei vincoli in R
     c_N ← min_X c_T
     S ← VE_SC(Vs \ {X}, Cs \ R ∪ {c_N})
     X_ott ← argmin_X c_T(S)
     return S ∪ {X = X_ott}
```

Ordine di eliminazione:

- dato a priori
- calcolato via via
  - ad es., usando euristiche viste per VE_CSP

Si può implementare VE_SC senza memorizzare c_T e costruendo solo una rappresentazione estensionale di c_N

Esempio — Tornando a un esempio precedente

- A è solo in $c_1(A, B)$; eliminandola:

$$c_6(B) = \arg\min_A c_1(A, B) :$$

B	$Costo$
1	0
2	2
3	2

$c_1(A, B)$ sostituito con $c_6(B)$

(..cont.)

- B compare in $c_2(B, C)$, $c_3(B, D)$ e $c_6(B)$ la cui somma è:

$c_2(B, C) + c_3(B, D) + c_6(B)$:

B	C	D	$costo$
1	1	1	8
1	1	2	5
⋮	⋮	⋮	⋮
2	1	1	4
2	1	2	4
⋮	⋮	⋮	⋮
3	1	1	6
3	1	2	8

(..cont.)

- Vengono quindi sostituiti da $c_7(C, D) = \text{argmin}_B(c_2(B, C) + c_3(B, D) + c_6(B))$:

C	D	$costo$
1	1	4
1	2	4
$\vdots$	$\vdots$	$\vdots$

- Restano $c_4(C, E)$, $C_5(D, E)$ e $c_7(C, D)$ da ottimizzare ricorsivamente
- Supponendo che la chiamata ricorsiva restituisca la soluzione $C = 1, D = 2, E = 2$, un valore ottimale per B è quello relativo al minimo di $c_2(B, C = 1) + c_3(B, D = 2) + c_6(B) \longrightarrow B = 2$
- Da $c_1(A, B)$, il valore di A che minimizza $c_1(A, B = 2)$ è $A = 1$
- Quindi la soluzione ottimale sarà $A = 1, B = 2, C = 1, D = 2, E = 2$ dal costo pari a 4

Complessità ∢

dipende dalla struttura del grafo dei vincoli (come per i CSP)

- grafi *sparsi* → piccoli vincoli intermedi, negli algoritmi VE come `VE_SC`
- grafi densamente *connessi* → vincoli intermedi più grandi

Ricerca Locale per l'Ottimizzazione

Ricerca locale su problemi di ottimizzazione:
minimizzare la funzione-obiettivo
(invece di cercare generiche soluzioni)

- algoritmi di ricerca locale (anche con ripartenze casuali)
  - mantenendo e infine restituendo la *migliore assegnazione trovata*

Vincoli Misti

- vincoli *rigidi* → soluzione senza conflitti
- vincoli *flessibili* → difficile determinare se un'assegnazione totale trovata sia la miglior soluzione secondo il criterio di ottimalità
  - **ottimo locale**: assegnazione non peggiore di tutti i *possibili successori*
  - **ottimo globale**: assegnazione non peggiore di *tutte* le assegnazioni
    - senza ricerca sistematica non si può sapere se la migliore trovata localmente sia ottimo globale o se ne esista altrove una migliore
- con *vincoli misti* può essere necessario consentire la violazione di quelli rigidi pur di arrivare a una soluzione ottimale
  - adottando costi di violazione alti ma *finiti*

Domini Continui: Gradiente

Ricerca locale più complicata:
come definire il *successore* di un'assegnazione?

Algoritmo di **discesa del gradiente**, Gradient Descent

Idea minimizzare la funzione di valutazione h (purché *continua* e *differenziabile*)

- come in un percorso in discesa, passi nelle direzioni più ripide
- *successore* di un'assegnazione:
  passo proporzionale alla *pendenza* di h, quindi alla sua *derivata*
  - ma in discesa (segno negativo)

Ricerca di massimi tramite **risalita di gradiente**, Gradient Ascent

Caso Monodimensionale

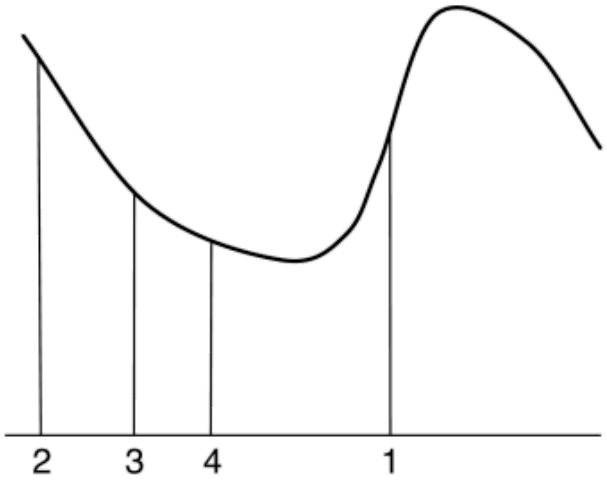
Se a X è assegnato $v \in \mathbb{R}$, valore successivo:

$$v - \eta \cdot \frac{dh}{dX}(v)$$

- η, misura del *passo*, determina la *rapidità* della discesa
  - troppo grande, può superare il minimo
  - troppo piccolo, progresso lento
- derivata valutata in v:

$$\lim_{\epsilon \to 0} \frac{h(X = v + \epsilon) - h(X = v)}{\epsilon}$$

Esempio — Ricerca di minimo locale in una funzione di una variabile



Discesa di gradiente

0. inizia in posizione (1)
1. derivata positiva (grande valore) → a sinistra sulla pos. (2)
2. derivata negativa prossima a zero → passo più breve verso destra (3)
3. derivata negativa ancor più vicina a zero
   → passo ancor più breve verso destra (4) …
   - avvicinandosi al minimo, la pendenza tende a zero: passi più piccoli

Discesa di Gradiente

Passi in tutte le dimensioni, proporzionali a ogni *derivata parziale*

- variabili $\langle X_1, \ldots, X_n \rangle$
- assegnazione $\vec{v} = \langle v_1, \ldots, v_n \rangle$
- *successore* ottenuto muovendosi in ogni direzione in proporzione alla relativa pendenza di h:

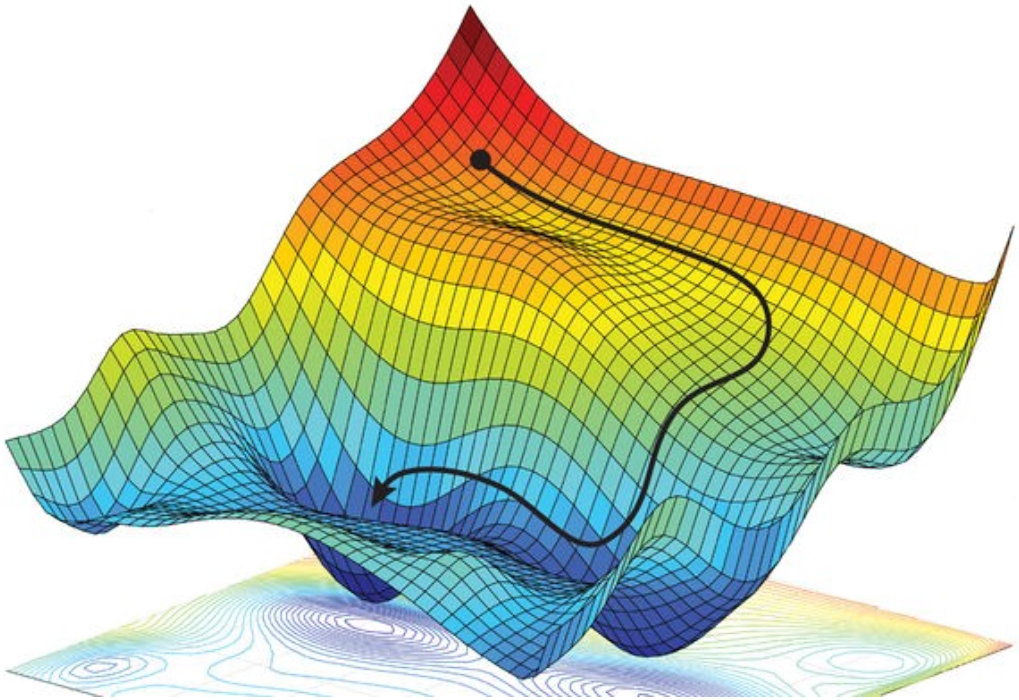
$$v_i \leftarrow v_i - \eta \cdot \frac{\partial h}{\partial X_i}(\vec{v}) \qquad \forall i = 1, \ldots, n$$

nuovo valore per X_i

  - $\frac{\partial h}{\partial X_i}$ *derivata parziale*, funzione di $X_1, \ldots, X_n$:

$$\frac{\partial h}{\partial X_i}(\vec{v}) = \lim_{\epsilon \to 0} \frac{h(v_1, \ldots, v_i + \epsilon, \ldots, v_n) - h(v_1, \ldots, v_i, \ldots, v_n)}{\epsilon}$$

    - meglio se calcolabile *analiticamente*
    - altrimenti *stimata* per piccoli valori di ϵ



Discesa di gradiente — caso multidimensionale
Da: https://towardsdatascience.com

Utilità

Uso discesa di gradiente:

- nell'apprendimento: valore dei parametri di un modello
  - migliaia/milioni di parametri da ottimizzare

Varianti: molte

- ad es., η non costante
  - ricerca binaria per cercare un valore ottimale

Osservazioni

- per funzioni regolari (*smooth*) con un minimo, la discesa di gradiente converge a un minimo locale se il passo è sufficientemente piccolo
  - passo troppo grande → possibile che diverga
  - passo troppo piccolo → lento
- minimo locale unico → trovato min. globale
- più minimi locali, non tutti globali:
  necessaria una ricerca per trovare il minimo globale
  - ad es. con RANDOM RESTART o RANDOM WALK
- garantito il minimo globale solo avendo attraversato l'intero spazio di ricerca

Riferimenti

[1] D. Poole, A. Mackworth: *Artificial Intelligence: Foundations of Computational Agents*. Cambridge University Press [Ch.4]
[2] D. Poole, A. Mackworth, R. Goebel: *Computational Intelligence: A Logical Approach*. Oxford University Press
[3] S. J. Russell, P. Norvig: *Artificial Intelligence* Pearson. 4rd Ed. - cfr. anche ed. Italiana [Cap.6,4]

Link

[AIPython] sezione *Reasoning with Constraints* https://artint.info/AIPython/
[Gradiente] wikipedia di funzioni vettoriali
[CSP] wikipedia (in inglese)
[CP] Programmazione a Vincoli wikipedia
[GradientDescent] Discesa del Gradiente (steepest descent) wikipedia
[Optimization] Portale (in inglese)

Note

[⪡] consigliata la lettura
[versione] 12/10/2022, 11:25:11

Figure tratte da [1] salvo diversa indicazione