Agenti logici: calcolo proposizionale

Paolo Mancarella

a.a. 2022/2023

Equivalenza logica

- Equivalenza logica:

  $\alpha \equiv \beta$ se e solo se $\alpha \models \beta$ e $\beta \models \alpha$

Esempi:

$A \wedge B \equiv B \wedge A$

$\neg(A \wedge B) \equiv \neg A \vee \neg B$

$(A \wedge (A \vee B)) \equiv A$

Equivalenze logiche (leggi)

$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha)$	commutatività di $\wedge$
$(\alpha \vee \beta) \equiv (\beta \vee \alpha)$	commutatività di $\vee$
$((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma))$	associatività di $\wedge$
$((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma))$	associatività di $\vee$
$\neg(\neg\alpha) \equiv \alpha$	eliminazione della doppia negazione
$(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha)$	contrapposizione
$(\alpha \Rightarrow \beta) \equiv (\neg\alpha \vee \beta)$	eliminazione dell'implicazione
$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$	eliminazione del bicondizionale
$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta)$	De Morgan
$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta)$	De Morgan
$(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)$	distributività di $\wedge$ su $\vee$
$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)$	distributività di $\vee$ su $\wedge$

Validità, soddisfacibilità

- α valida *sse* è vera in tutte le interpretazioni (detta anche tautologia)
  - Es. $A \vee \neg A$
- α soddisfacibile *sse* esiste una interpretazione in cui α è vera (ovvero se esiste un modello di α)
- Ne discende che
  - α è valida *sse* $\neg\alpha$ è insoddisfacibile
  - α è soddisfacibile *sse* $\neg\alpha$ non è valida

Inferenza per PROP

- *Model checking*
  - una forma di inferenza che fa riferimento alla definizione di conseguenza logica (si enumerano i possibili modelli)
  - Tecnica delle tabelle di verità
- Algoritmi per la *soddisfacibilità* (SAT)
  - $KB \models \alpha$ *sse* $(KB \wedge \neg\alpha)$ è insoddisfacibile (*Teorema di refutazione*)
  - La conseguenza logica può essere ricondotta a un problema SAT

Model checking

- Consiste nel:
  - Enumerare tutti i possibili modelli di KB
  - Verificare che in essi la formula α sia vera

$$M(KB) \subseteq M(\alpha)$$

L’algoritmo TV-Consegue?

- Quesito: α è conseguenza logica di KB? (KB $\models \alpha$?)
- Enumera tutte le possibili interpretazioni di KB
  - (*k* simboli, 2^k possibili interpretazioni)
- Per ciascuna interpretazione
  - Se non soddisfa KB, OK
  - Se soddisfa KB, si controlla che soddisfi anche α
- Se si trova anche solo una interpretazione che soddisfa KB e non α la risposta sarà NO.

Funzione TV-Consegue?

function TV-Consegue?(*KB*, α) // **returns** *true* oppure *false*
inputs: *KB*, la base di conoscenza, una formula della logica proposizionale
α, la query, una formula della logica proposizionale

simboli ← una lista dei simboli proposizionali contenuti in *KB* e α

return TV-Verifica-Tutto(*KB*, α, *simboli*, { })

function TV-Verifica-Tutto(*KB*, α, *simboli*, *modello*) //returns *true* oppure *false*

if Vuoto?(*simboli*) **then**
if PL-Vero?(*KB*, *modello*) **then return** PL-Vero?(α, *modello*)
else return *true* // quando *KB* è *false*, restituisce sempre *true*
else do
P ← Primo(*simboli*); *resto* ← Resto(*simboli*)
return TV-Verifica-Tutto(*KB*, α, *resto*, *modello* ← {*P* = *true*})
and
TV-Verifica-Tutto(*KB*, α, *resto*, *modello* ← {*P* = *false*})

Tabella di verità

$$(\neg a \vee b) \wedge (a \vee c) \models (a \vee c)$$

a	b	c	$\neg a \vee b$	$a \vee c$	$b \vee c$
T	T	T	T	T	
T	T	F	T	T	
T	F	T	F	T	
T	F	F	F	T	
F	T	T	T	T	
F	T	F	T	F	
F	F	T	T	T	
F	F	F	T	F	

Tabella di verità

$$(\neg a \vee b) \wedge (a \vee c) \models (a \vee c)$$

a	b	c	¬a ∨ b	a ∨ c	b ∨ c
T	T	T	T	T	
T	T	F	T	T	
F	T	T	T	T	
F	F	T	T	T	

Tabella di verità

$$(\neg a \vee b) \wedge (a \vee c) \models (a \vee c)$$

a	b	c	¬a ∨ b	a ∨ c	b ∨ c
T	T	T	T	T	T
T	T	F	T	T	T
F	T	T	T	T	T
F	F	T	T	T	T

Esempio di TV-Consegue?

KB: $(\neg a \vee b) \wedge (a \vee c)$ α: $(b \vee c)$ *simboli* = [a, b, c]

- TT-VERIFICA-TUTTO(KB, α, [a, b, c], { })
  - TT-VERIFICA-TUTTO(KB, α, [b, c], {a=T})
    - ✓ TT-VERIFICA-TUTTO(KB, α, [c], {a=T, b=T})
      - TT-VERIFICA-TUTTO(KB, α, [ ], {a=T, b=T, c=T}) OK
      - TT-VERIFICA-TUTTO(KB, α, [ ], {a=T, b=T, c=F}) OK
    - ✓ TT-VERIFICA-TUTTO(KB, α, [c], {a=T, b=F})
      - TT-VERIFICA-TUTTO(KB, α, [ ], {a=T, b=F, c=T}) OK
      - TT-VERIFICA-TUTTO(KB, α, [ ], {a=T, b=F, c=F}) OK
  - TT-VERIFICA-TUTTO(KB, α, [b, c], [a=F]) …

In questi due casi l'interpretazione è un modello di KB e di α
PL-Vero?(*KB*, *modello*)=true
PL-Vero?(α, *modello*)=true

In questi due casi l'interpretazione non è un modello di KB
PL-Vero?(*KB*, *modello*)=false

Solo alla fine, dopo avere provato tutti i possibili assegnamenti (interpretazioni) possiamo rispondere se $(b \vee c)$ è o meno conseguenza logica.

Algoritmi per la soddisfacibilità (SAT)

- Ne vediamo alcuni che usano KB in forma a clausole
- La forma a clausole è la forma normale congiuntiva (CNF): una congiunzione di disgiunzioni di letterali

  $(A \vee B) \wedge (\neg B \vee C \vee D) \wedge (\neg A \vee F)$

- Non è restrittiva: è sempre possibile ottenerla con trasformazioni che preservano l'equivalenza logica
- Può essere rappresentata come insiemi di letterali

  {A, B} {¬B, C, D} {¬A, F}

Trasformazione in forma a clausole

I passi sono:

1. Eliminazione della $\Leftrightarrow$: $(A \Leftrightarrow B) \equiv (A \Rightarrow B) \wedge (B \Rightarrow A)$
2. Eliminazione dell' $\Rightarrow$: $(A \Rightarrow B) \equiv (\neg A \vee B)$
3. Negazioni all'interno:

   $\neg(A \vee B) \equiv (\neg A \wedge \neg B)$

   $\neg(A \wedge B) \equiv (\neg A \vee \neg B)$
4. Distribuzione di $\vee$ su $\wedge$:

   $(A \vee (B \wedge C)) \equiv (A \vee B) \wedge (A \vee C)$

Esempio di trasformazione

C'è brezza in [1,1] *sse c'è un pozzo in* [1,2] *o* [2,1]

1. $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$
2. $(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$
3. $(\neg B_{1,1} \vee (P_{1,2} \vee P_{2,1})) \wedge (\neg (P_{1,2} \vee P_{2,1}) \vee B_{1,1})$
4. $(\neg B_{1,1} \vee (P_{1,2} \vee P_{2,1})) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$
5. $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$
6. $\{\neg B_{1,1}, P_{1,2}, P_{2,1}\}\ \{\neg P_{1,2}, B_{1,1}\}\ \{\neg P_{2,1}, B_{1,1}\}$

L’algoritmo DPLL per la soddisfacibilità

- DPLL: Davis, Putman, e poi Lovemann, Loveland
- Parte da una KB in forma a clausole
- È una enumerazione ricorsiva *in profondità* di tutte le possibili interpretazioni alla ricerca di un modello
- Tre miglioramenti rispetto a TT-Consegue:
  1. Terminazione anticipata
  2. Euristica dei simboli (o letterali) puri
  3. Euristica delle clausole unitarie

DPLL: terminazione anticipata

- Si può decidere sulla verità di una clausola anche con interpretazioni parziali: basta che un letterale sia vero
  - Se A è vero, lo sono anche {A, B} e {A, C} indipendentemente dai valori di B e C
- Se anche una sola clausola è falsa l'interpretazione non può essere un modello dell’insieme di clausole

DPLL: simboli puri

- *Simbolo puro*: un simbolo che appare con lo stesso segno in tutte le clausole

  Es. {A, ¬B} {¬B, ¬C} {C, A} A è puro, B anche
- Nel determinare se un simbolo è puro se ne possono trascurare le occorrenze in clausole già rese vere
- I simboli puri possono essere assegnati a *True* se il letterale è positivo, *False* se negativo.
- Non si eliminano modelli utili: se le clausole hanno un modello continuano ad averlo dopo questo assegnamento (non renderà mai falsa una clausola in cui compare il simbolo puro)

DPLL: clausole unitarie

- *Clausola unitaria*: una clausola con un solo letterale *non assegnato*

  Es. {B} è unitaria ma anche …

  {B, ¬C} è unitaria quando C = *True*

- Conviene assegnare prima valori al letterale in clausole unitarie. L'assegnamento è obbligato (*True* se positivo, *False* se negativo).

L’algoritmo DPLL

function DPLL-Soddisfacibile?(*s*) **returns** *true* oppure *false*
inputs: *s*, una formula della logica proposizionale
clausole ← l’insieme di clausole nella rappresentazione CNF di *s*
simboli ← una lista di tutti i simboli proposizionali in *s*
return DPLL(*clausole*, *simboli*, { })

function DPLL(*clausole*, *simboli*, *modello*) returns *true* oppure *false*
if ogni clausola in *clausole* è vera in *modello* **then return** *true*
if qualche clausola in *clausole* è falsa in *modello* **then return** *false*
P, *valore* ← Trova-Simbolo-Puro(*simboli*, *clausole*, *modello*)
if *P* è diverso da null **then return** DPLL(*clausole*, *simboli* – *P*, *modello* ← {*P* = *valore*})
P, *valore* ← Trova-Clausola-Unitaria(*clausole*, *modello*)
if *P* è diverso da null **then return** DPLL(*clausole*, *simboli* – *P*, *modello* ← {*P* = *valore*})
P ← Primo(*simboli*); *resto* ← Resto(*simboli*)
return DPLL(*clausole*, *resto*, *modello* ← {*P* = *true*}) **or**
DPLL(*clausole*, *resto*, *modello* ← {*P* = *false*})

DPLL: esempio

KB $\{\neg B_{1,1}, P_{1,2}, P_{2,1}\}\ \{\neg P_{1,2}, B_{1,1}\}\ \{\neg P_{2,1}, B_{1,1}\}\ \{\neg B_{1,1}\} \models \{\neg P_{1,2}\}$?

Aggiungiamo $\{P_{1,2}\}$ e vediamo se l'insieme è insoddisfacibile

SAT($\{\neg B_{1,1}, P_{1,2}, P_{2,1}\}\ \{\neg P_{1,2}, B_{1,1}\}\ \{\neg P_{2,1}, B_{1,1}\}\ \{\neg B_{1,1}\}\ \{P_{1,2}\}$) ?

1 2 3 4 5

1. La 5 è unitaria; $P_{1,2}$ = *True*; la prima clausola e la 5 sono soddisfatte

   $\{\neg B_{1,1}, P_{1,2}, P_{2,1}\}\ \{\neg P_{1,2}, B_{1,1}\}\ \{\neg P_{2,1}, B_{1,1}\}\ \{\neg B_{1,1}\}\ \{P_{1,2}\}$

2. $P_{2,1}$ è un simbolo puro; $P_{2,1}$ = *False*; 3 è soddisfatta

   $\{\neg B_{1,1}, P_{1,2}, P_{2,1}\}\ \{\neg P_{1,2}, B_{1,1}\}\ \{\neg P_{2,1}, B_{1,1}\}\ \{\neg B_{1,1}\}\ \{P_{1,2}\}$

   …

Non esistono modelli quindi $\neg P_{1,2}$ è conseguenza logica della KB

Miglioramenti di DPLL

- DPLL è completo e termina sempre
- Alcuni miglioramenti:
  - Analisi di componenti (sotto-problemi indipendenti): se le variabili possono essere suddivise in sotto-insiemi disgiunti (senza simboli in comune)
  - Ordinamento di variabili e valori: scegliere la variabile che compare in più clausole
  - Backtracking intelligente e altre ottimizzazioni …

Metodi locali per SAT: formulazione

- Gli ‘stati’ sono interpretazioni, assegnamenti completi
- L’obiettivo è un assegnamento che soddisfa tutte le clausole (un modello)
- Si parte da un assegnamento casuale
- Ad ogni passo si cambia il valore di un simbolo proposizionale (*flip*)
- Gli stati sono valutati contando il numero di clausole soddisfatte (più sono meglio è)

Metodi locali per SAT: algoritmi

- Ci sono molti massimi locali per sfuggire ai quali serve introdurre perturbazioni casuali
  - Hill climbing con riavvio casuale
  - Simulated Annealing
- Molta sperimentazione per trovare il miglior compromesso tra il grado di “avidità” e casualità
- WALK-SAT è uno degli algoritmi più semplici ed efficaci

WalkSAT

- WalkSAT ad ogni passo
  - Sceglie a caso una clausola non ancora soddisfatta
  - Individua un simbolo da modificare (*flip*), scegliendo con probabilità *p* (di solito 0,5) tra:
    - ✓ Scegliere un simbolo a caso (passo casuale, *random walk*)
    - ✓ Scegliere quello che rende più clausole soddisfatte (passo di ottimizzazione)
- Si arrende dopo un certo numero di *flip* predefinito

WalkSat: l'algoritmo

function WalkSAT(clausole, p, max_flips) **returns** un modello o fallimento
inputs: *clausole*, un insieme di clausole della logica proposizionale
p, la probabilità di effettuare una "camminata casuale", tipicamente intorno a 0,5
max_flips, numero massimo di inversioni di valore prima di abbandonare

modello ← un assegnamento casuale di valori di verità ai simboli in clausole

for i = 1 **to** max_flips **do**
  if modello soddisfa clausole **then return** modello
  clausola ← una clausola, falsa in modello, scelta casualmente nell'insieme clausole
  if Random(0, 1) =< p **then** inverti il valore in modello di un simbolo scelto casualmente in clausola
  else inverti il valore di verità del simbolo in clausole che massimizza il numero di clausole soddisfatte

return fallimento

WalkSAT: un esempio

Come euristica usiamo il numero di clausole soddisfatte (da massimizzare)

Rosso: passo casuale

Verde: passo di ottimizzazione

$\{\neg B_{1,1}, P_{1,2}, P_{2,1}\}$ (1) $\{\neg P_{1,2}, B_{1,1}\}$ (2) $\{\neg P_{2,1}, B_{1,1}\}$ (3) $\{\neg B_{1,1}\}$ (4)

$[B_{1,1}=F, P_{1,2}=T, P_{2,1}=T]$ 2, 3 F; scelgo 2; a caso flip $B_{1,1}$

$[B_{1,1}=T, P_{1,2}=T, P_{2,1}=T]$ 4 F; scelgo 4; flip $B_{1,1}$ unica scelta

$[B_{1,1}=F, P_{1,2}=T, P_{2,1}=T]$ 2, 3 F; scelgo 2; a caso flip $P_{1,2}$

$[B_{1,1}=F, P_{1,2}=F, P_{2,1}=T]$ 3 F; scelgo 3; ottimizzazione: flip $P_{2,1}$[#4]; flip $B_{1,1}$[#3]

$[B_{1,1}=F, P_{1,2}=F, P_{2,1}=F]$ ho trovato un modello!!!

Analisi di WalkSAT

- Se *max-flips* = ∞ e l'insieme di clausole è soddisfacibile prima o poi termina.
- Ma se vogliamo un algoritmo che termina, questo è necessariamente incompleto
- Va bene per cercare un modello, sapendo che c'è, ma se è insoddisfacibile non termina
- Non può essere usato per verificare l'insoddisfacibilità

Problemi SAT difficili

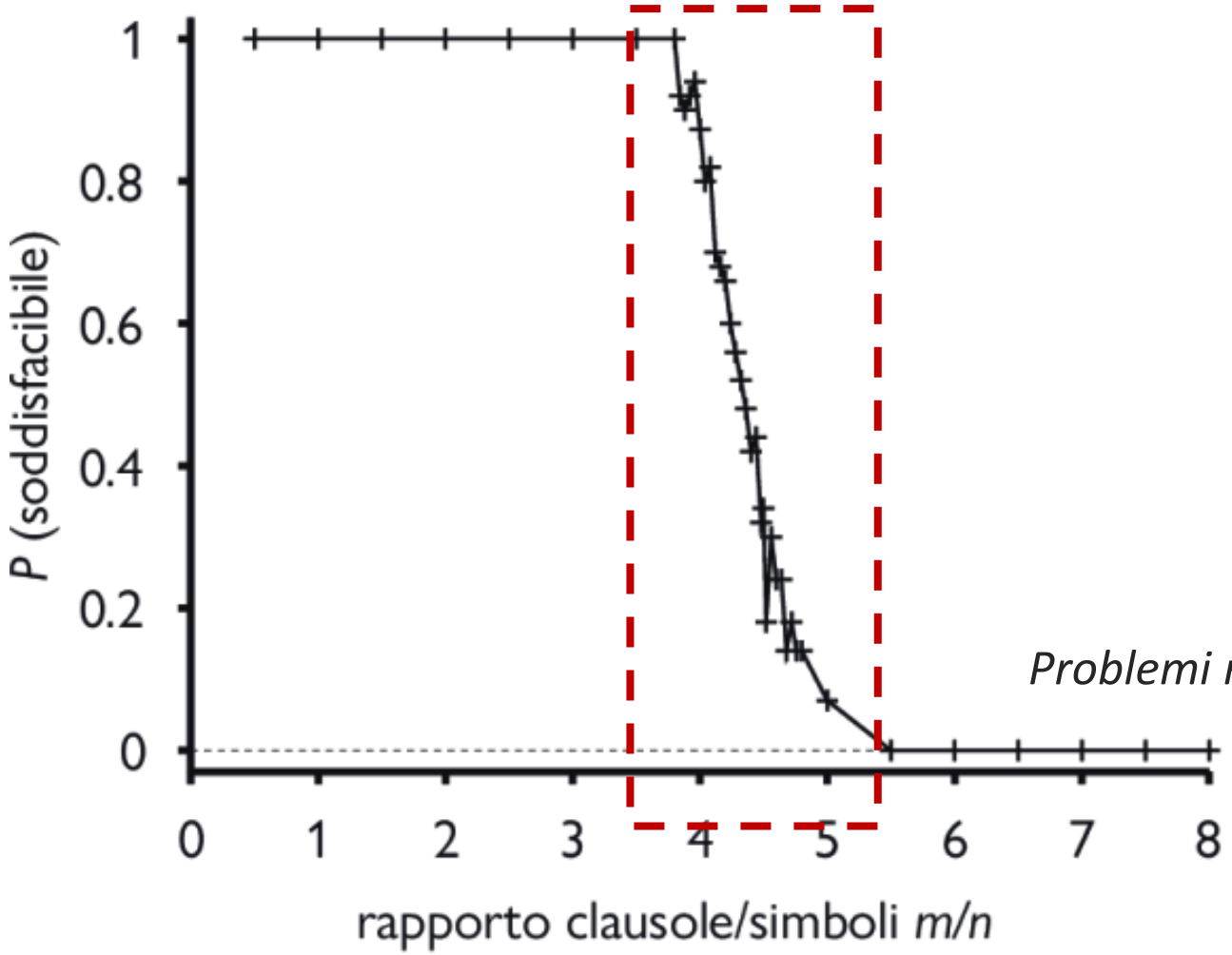
- Se un problema ha molte soluzioni (problema sotto-vincolato) è più probabile che WalkSAT ne trovi una in tempi brevi
  - Esempio: 16 soluzioni su 32; un assegnamento ha il 50% di probabilità di essere giusto: 2 passi in media!
- Istanza di 3–SAT (clausole con tre letterali)

  $\{\neg D, \neg B, C\}\ \{B, \neg A, \neg C\}\ \{\neg C, \neg B, E\}\ \{E, \neg D, B\}\ \{B, E, \neg C\}$

- È significativo il rapporto *m/n* dove *m* è il numero di clausole (vincoli) e *n* il numero di simboli. Nell'esempio: 5/5=1
- Più grande il rapporto, più vincolato è il problema
- Le regine sono 'facili 'perché il problema è sotto-vincolato

Probabilità di soddisfacibilità in funzione di m/n

Problemi poco vincolati

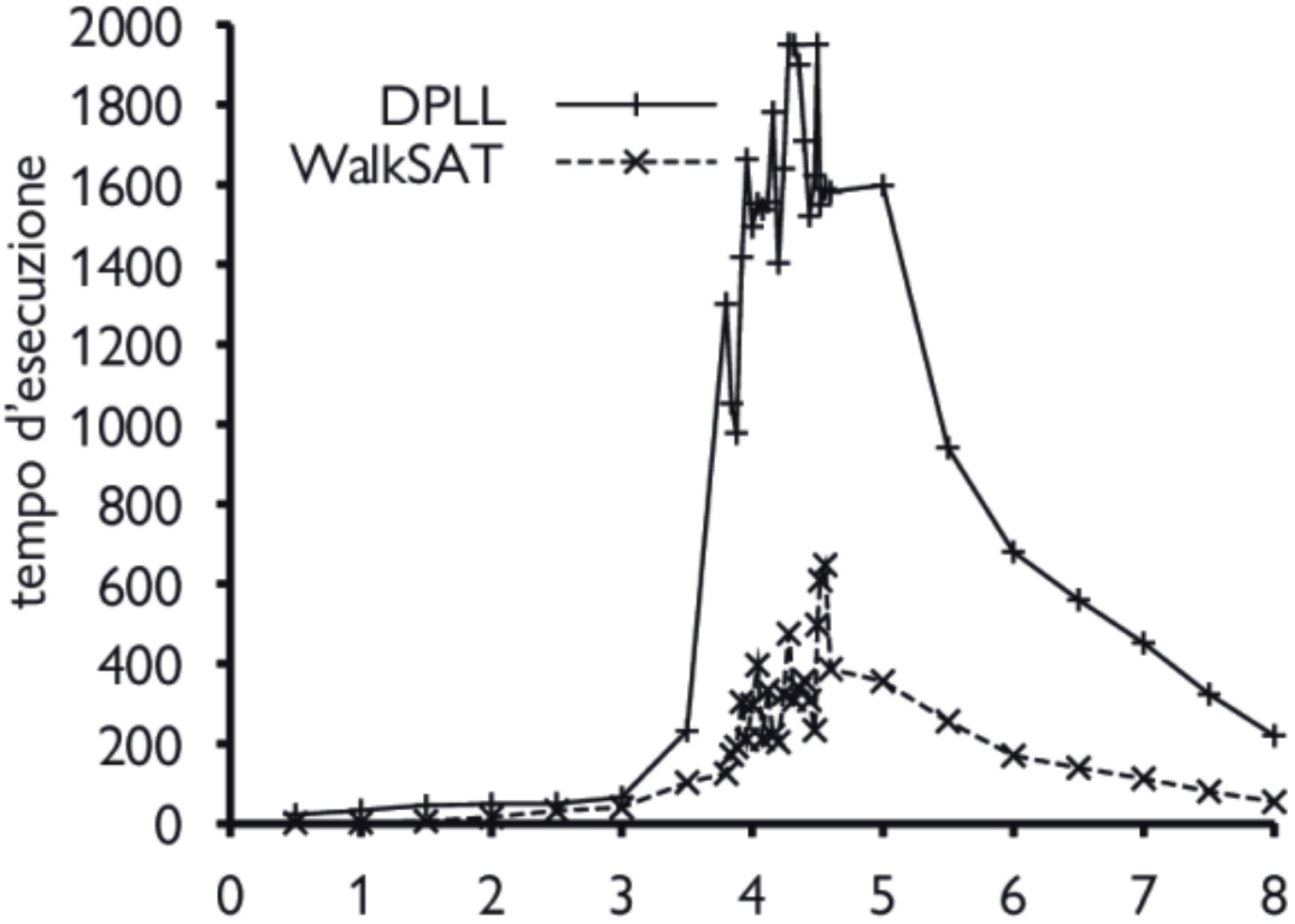


Problemi molto vincolati

m (n. clausole) varia
n (n. simboli) = 50
3 letterali per clausola

media su 100
problemi generati a caso

Confronto tra DPLL e WalkSAT



Confronto su problemi soddisfacibili, ma difficili

Inferenza come deduzione

- Un altro modo per decidere se KB $\models$ A è usare un sistema di deduzione
  - Denotiamo con KB $\vdash$ A il fatto che A è derivabile (deducibile) da KB
  - La deduzione avviene specificando delle regole di inferenza.
- In un sistema di inferenza le regole …
  - dovrebbero derivare solo formule che sono conseguenza logica
  - dovrebbero derivare tutte le formule che sono conseguenza logica

Correttezza e completezza

- Correttezza: Se KB $\vdash \alpha$ allora KB $\models \alpha$

  Tutto ciò che è derivabile è conseguenza logica. Le regole preservano la verità

- Completezza: Se KB $\models \alpha$ allora KB $\vdash \alpha$

  Tutto ciò che è conseguenza logica è ottenibile tramite il sistema deduttivo.

Alcune regole di inferenza per Prop

- Le regole sono schemi deduttivi del tipo:

$$\frac{\alpha \Rightarrow \beta, \quad \alpha}{\beta}$$

Modus ponens

$$\frac{\alpha \wedge \beta}{\alpha}$$

Eliminazione dell'AND

$$\frac{\alpha \Leftrightarrow \beta}{(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)}$$

$$\frac{(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)}{\alpha \Leftrightarrow \beta}$$

Eliminazione e introduzione della doppia implicazione

Una rappresentazione per il Wumpus

R_1: $\neg P_{1,1}$ — *non ci sono pozzi in* [1, 1]

C'è brezza nelle caselle adiacenti ai pozzi*:*

R_2: $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$

R_3: $B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{1,2} \vee P_{2,1})$

Percezioni:

R_4: $\neg B_{1,1}$ — *non c'è brezza in* [1, 1]

R_5: $B_{2,1}$ — *c'è brezza in* [2, 1]

$KB = \{R_1, R_2, R_3, R_4, R_5\}$ $\quad KB \models \neg P_{1,2}$?

Dimostrazione

R_6: $(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$ (R_2, $\Leftrightarrow$E)

R_7: $(P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1}$ (R_6, $\wedge$E)

R_8: $\neg B_{1,1} \Rightarrow \neg(P_{1,2} \vee P_{2,1})$ (R_7, *contrapposizione*)

R_9: $\neg(P_{1,2} \vee P_{2,1})$ (R_4 e R_8, *Modus Ponens*)

R_{10} : $\neg P_{1,2} \wedge \neg P_{2,1}$ (R_9, De Morgan)

R_{11} : $\neg P_{1,2}$ (R_{10}, $\wedge$E)

Dimostrazione come ricerca

- *Problema*: come decidere ad ogni passo qual è la regola di inferenza da applicare? … e a quali premesse? Come evitare l'esplosione combinatoria?
- Problema di esplorazione di uno spazio di stati
- Una *procedura di dimostrazione* definisce:
  - la direzione della ricerca
  - la strategia di ricerca

Direzione della ricerca

- Nella dimostrazione di teoremi conviene procedere all'indietro. Con una applicazione *in avanti* delle regole di inferenza non controllata:

  Da A, B: $A \wedge B$ $A \wedge (A \wedge B)$ … $A \wedge (A \wedge (A \wedge B))$

- Meglio *all'indietro*
  - se si vuole dimostrare $A \wedge B$, si cerchi di dimostrare A e poi B
  - se si vuole dimostrare $A \Rightarrow B$, si assuma A e si cerchi di dimostrare B
  - …

Strategia di ricerca

- Completezza
  - Le regole della deduzione naturale sono un insieme di regole di inferenza completo
  - Se l'algoritmo di ricerca è completo siamo a posto
- Efficienza
  - La complessità è alta: è un problema decidibile ma NP-completo

Regola di risoluzione per PROP

- Meno regole sono meglio è, senza rinunciare alla completezza.
- Un'unica regola: la regola di risoluzione (presuppone forma a clausole)

$$\frac{\{P, Q\} \quad \{\neg P, R\}}{\{Q, R\}}$$

$$\frac{(P \vee Q) \wedge (\neg P \vee R)}{(Q \vee R)}$$

- Corretta? Basta pensare ai modelli
- Preferita la notazione insiemistica: gli eventuali duplicati si eliminano

La regola di risoluzione in generale

$$\frac{\{l_1, l_2, \ldots l_i, \ldots l_k\} \quad \{m_1, m_2, \ldots m_j, \ldots m_n\}}{\{l_1, l_2, \ldots l_{i-1}, l_{i+1}, \ldots l_k\, m_1\, m_2, \ldots m_{j-1}, m_{j+1}, \ldots m_n\}}$$

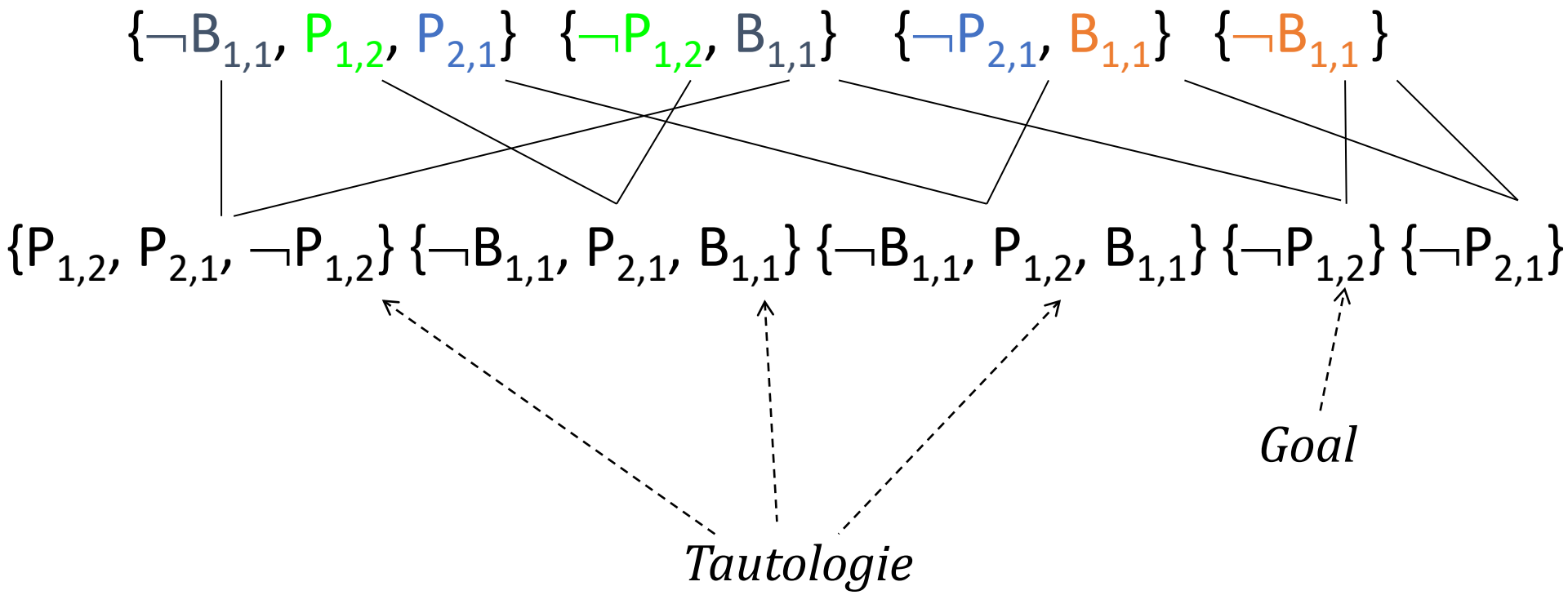
Gli l e m sono letterali, simboli di proposizione positivi o negativi;
l_i e m_j sono uguali e di segno opposto

Caso particolare

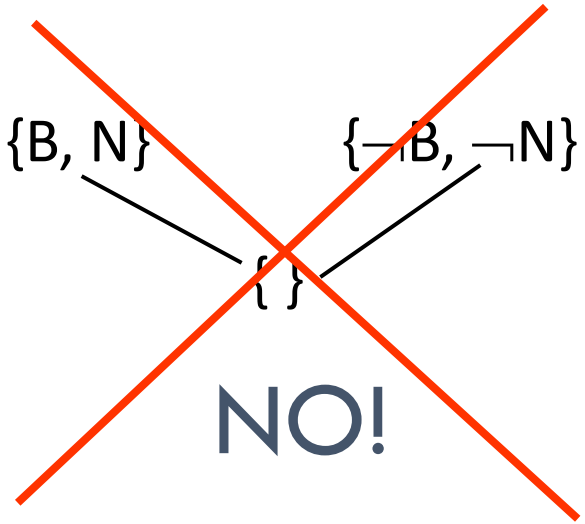
$$\frac{\{P\} \quad \{\neg P\}}{\{\,\}}$$

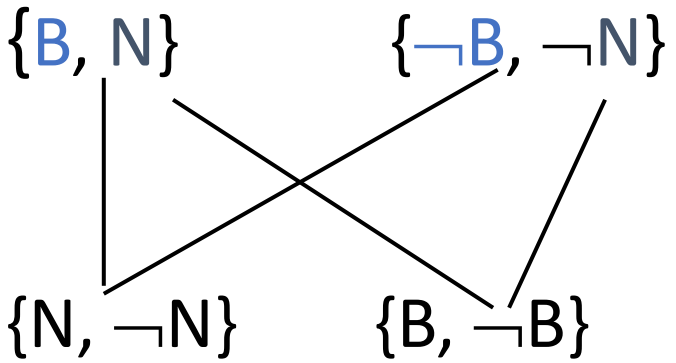
clausola vuota
contraddizione

Il grafo di risoluzione



Attenzione!





… e qui ci fermiamo

Un passo alla volta !!!

Ma siamo sicuri che basti una regola? È completo?

- Completezza: se KB $\models \alpha$ allora KB $\vdash_{res} \alpha$? Non sempre. Ci basta un contro-esempio.

  Es. KB $\models \{A, \neg A\}$ ma non è vero che KB $\vdash_{res} \{A, \neg A\}$

- Teorema di risoluzione [ground]:

  KB insoddisfacibile *sse* KB $\vdash_{res} \{\}$ *completezza*

- Teorema di refutazione offre un modo alternativo:

  KB $\models \alpha$ sse (KB $\cup \{\neg\alpha\}$) è insoddisfacibile

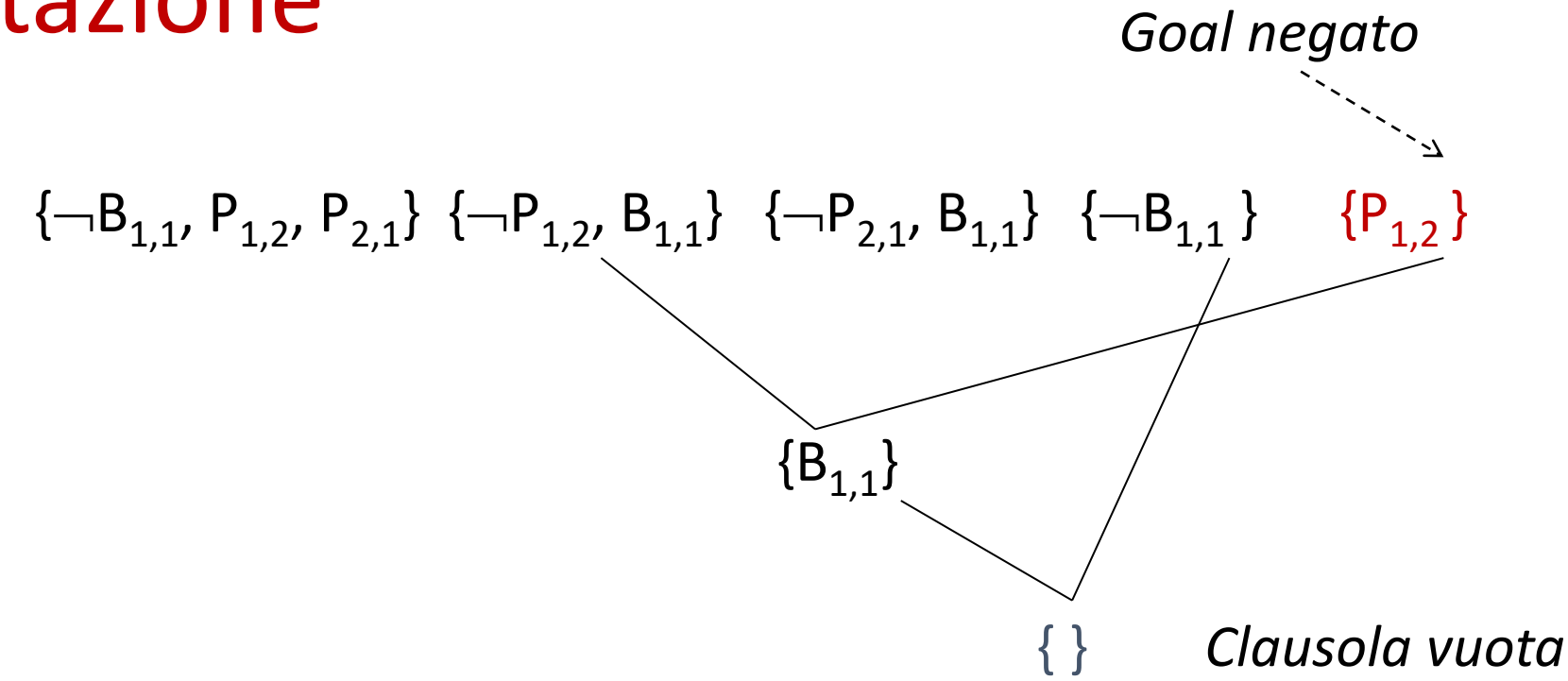
- Nell'esempio:

  KB $\cup$ *FC* $(\neg(A \vee \neg A))$ è insoddisfacibile? Sì, perché …

  KB $\cup \{A\} \cup \{\neg A\} \vdash_{res} \{\}$ in un passo

  Quindi possiamo concludere che KB $\models \{A, \neg A\}$

Refutazione



Conclusioni

- Abbiamo visto come gli agenti KB che usano PROP come linguaggio di rappresentazione possono decidere se KB $\models \alpha$
- Il problema è decidibile, ma intrattabile (NP) nel caso peggiore
- Esistono algoritmi efficienti e completi che consentono di affrontare problemi di grosse dimensioni
- I metodi locali sono particolarmente efficienti ma non completi

Riferimenti

- AIMA Cap 7
  - 7.1 – 7.6