Università degli Studi di Brescia - Facoltà di Ingegneria

Corso di Intelligenza Artificiale

PROPOSITIONAL SATISFIABILTY AND SAT SOLVER

Prof. Alfonso Gerevini

Catalin Roman

Matr. 036915

E-mail croman@neolt.it

Indice

Abstract

Il presente elaborato, si propone di fare un riassunto dello sviluppo degli algoritmi per la soddisfacibilità proposizionale, con un approfondimento su alcuni degli algoritmi che si sono evidenziati per le notevoli performance.

Nel Capitolo 1 sono definiti i concetti di ricerca sistematica e locale, problemi NP-completi e NP-hard, logica proposizionale e soddisfacibilità, forme normali, problemi con soddisfacimenti di vincoli.

Il Capitolo 2 si concentra sulle procedure complete per SAT ed argomenti come le eeuristiche, le strutture di dati e la loro realizzazione, backtracking ed apprendimento, risoluzione ed altri approcci.

Il Capitolo 3 evidenzia le principali caratteristiche di alcune procedure non complete come greedy, GSAT, random walk, Walksat, Novelty mentre nel Capitolo5 si fa un riassunto dei principali tipi di problemi affrontati dagli algoritmi SAT.

Cominciando con il Capitolo 6 si entra nello studio approfondito degli algoritmi che rappresentano il tema del presente elaborato.

Seguono le conclusioni nel Capitolo 10, link delle biblioteche di benchmark nel Capitolo 11 e la bibliografia nel Capitolo 12.

Per la realizzazione di questo elaborato sono stati utilizzati vari pubblicazioni tra cui si notano *The Search for Satisfaction* di Gent e Walsh [44], *Noise Strategies for Improving Local Search* di Selman, Kautz e Cohen [114], *Using CSP look-back techniques to solve exceptionally hard SAT instances* di Bayardo e Schrag [**4**], *Look-ahead versus look-back for satisfiability problems* di Li e Ambulagan [**89**], *Evidence for invariants in local search* di McAllester, Selman, e Kautz [**93**] e la tesi di dottorato di H. Hoos *Stochastic Local Search - Methods, Models, Applications* [**66**].

1 Introduzione

Negli ultimi anni, c'è stato un aumento notevole della ricerca AI nella soddisfacibilità proposizionale (SAT). Ci sono molti fattori dietro il crescente interesse in questo settore.

Un primo fattore è il miglioramento delle procedure di ricerca per SAT. Alcune nuove procedure di ricerca locale, come per esempio il GSAT, sono capaci di risolvere problemi SAT con migliaia di variabili. Nello stesso tempo, le implementazioni degli algoritmi di ricerca completa, come quello di Davis-Putnam, sono stati capaci di risolvere molti problemi matematici. Un altro fattore è l'identificazione dei problemi SAT difficili con la transizione di fase nella solubilità.

Un terzo fattore è la dimostrazione che possiamo spesso risolvere problemi del mondo reale codificandoli nel SAT. Inoltre, c'è stato anche un miglioramento della comprensione teoretica del SAT.

1.1 Problemi AI, logica, e ricerca

Il ruolo generale dell'AI è di realizzare un comportamento intelligente nelle macchine, idea che è stata anche alla base della costruzione dei primi computer. Da quando questa motivazione di base è strettamente correlata con questioni generali riguardanti l'intelligenza, l'apprendimento ed il lavoro della mente umana, i problemi dell'AI sono spesso, se non sempre, riferiti a discipline totalmente diverse da quelle riguardanti i computer, come psicologia, filosofia e neurobiologia. Alcuni dei problemi più rilevanti nell'AI sono le aree del riconoscimento vocale, la dimostrazione di problemi, i giochi, e la composizione musicale. Sfortunatamente, la maggior parte di questi compiti comportano problemi computazionali molto difficili e nonostante i successi impressionanti in molte aree dell'AI, mancano soluzioni soddisfacenti e fattibili a molti problemi dell'AI.

Come tutti i problemi della scienza dei computer, i problemi AI possono essere suddivisi in due grandi categorie: problemi di rappresentazione e problemi computazionali. Il primo tipo di problema comporta trovare una rappresentazione appropriata o formalizzazione di un problema da qualche dominio, mentre il secondo tipo consiste nel trovare gli algoritmi per risolvere questi problemi. Chiaramente, gli entrambi tipi di problemi si intrecciano, quanto tempo la risoluzione di un problema dal mondo reale richiede di trovare una rappresentazione appropriata e concepire un algoritmo che risolva il problema formalizzato. Inoltre, la costruzione e l'implementazione degli algoritmi risolutori di problemi dipendono spesso in modo critico dalla rappresentazione appropriata del problema.

Nell'AI, i problemi sono spesso rappresentati attraverso le logiche formali. Una logica formale L è data da un linguaggio formale che specifica la sintassi delle espressioni logiche, ed il mapping di queste espressioni in un sistema di costanti logiche che specificano la semantica di L. Esempi di logiche formali che sono spesso utilizzate e studiate nell'AI sono la logica proposizionale, la logica di primo ordine, del secondo ordine, la logica modale, la logica temporale.

In un sistema logico dato, le proposizioni che sono $True$ sono chiamate *teoremi*.

I teoremi possono essere dimostrati semanticamente, utilizzando il ragionamento formale fuori del sistema logico.

Alternativamente, il meccanismo sintattico di dimostrazione chiamato $calculi$ può essere utilizzato per stabilire la verità o la falsità delle proposizioni logiche.

Un calcolo logico è un sistema formale che consiste in un set di *assiomi* che sono proposizioni elementari vere, ed un set di *regole* che definiscono le trasformazioni che conservano la verità delle frasi. $Calculi$ è la base per l'automatizzazione della logica. Per

qualche sistema logico, ci sono proposizioni semanticamente vere che non possono essere dimostrate algoritmicamente. In queste logiche, il teorema è non risolvibile. La logica del primo ordine è di questo tipo, mentre per la logica proposizionale la validità delle proposizioni può essere decisa algoritmicamente.

Molti problemi dell'AI, particolarmente quelli che sorgono nel contesto delle logiche formali, sono problemi di decisione, dove le soluzioni sono caratterizzate da un set di condizioni che devono essere soddisfatte per risolvere il problema. Il risolutore di problemi classico è formulato al solito come un problema di decisione. Comunque, i problemi dai domini applicativi sono spesso piuttosto problemi di ottimizzazione che di decisione. I problemi di ottimizzazione possono essere considerati generalizzazioni dei problemi di decisione, dove le soluzioni sono valutate da una funzione obiettivo. Ci sono due tipi dei problemi di ottimizzazione, una versione di massimizzazione ed una di minimizzazione. Per questi problemi, lo scopo è di trovare soluzioni ottimali, per esempio soluzioni con valore massimo, rispettivamente minimo, della funzione obiettivo.

In molti casi nell'AI, i problemi di decisione e di ottimizzazione possono essere caratterizzati come problemi di ricerca, per esempio problemi che richiedono una ricerca attraverso un numero di soluzioni candidato o soluzioni parziali, mentre si tenta di trovare una soluzione valida, e forse ottimale, per le istanze del problema dato. Molti problemi di gioco, dimostrazioni di teoremi, possono essere formulate come problemi di ricerca. Notiamo che gli algoritmi per risolvere i problemi di ricerca non necessitano essere sempre degli algoritmi classici di ricerca.

Come menzionato prima, molti problemi dell'AI sono molto difficili computazionalmente. Usando la metafora della ricerca, di solito questo è riflesso dal fatto che per le istanze di un problema dato, il set delle soluzioni candidato o delle soluzioni parziali, che definiscono lo spazio in cui ha luogo la ricerca, è molto grande e non sono conosciuti metodi algoritmici per trovare soluzioni efficaci in questi enormi spazi di ricerca.

Una situazione tipica è il caso dei problemi dove la complessità temporale per trovare soluzioni cresce esponenzialmente con la dimensione del problema, mentre la decisione se una soluzione candidato è reale può essere fatta in tempo polinomiale.

1.2 Ricerca sistematica confrontata con la ricerca locale

I metodi di ricerca possono essere classificati in due grandi categorie:

- ricerca sistematica;
- ricerca locale.

Gli algoritmi di ricerca sistematica esaminano lo spazio di ricerca delle istanze di un problema in maniera sistematica, garantiscono di trovare una soluzione, oppure se nessuna soluzione esiste, questo fatto è determinato con certezza. Questa proprietà tipica degli algoritmi, basata sulla ricerca sistematica si chiama *completezza*.

D'altra parte la ricerca locale inizia la ricerca in qualche punto dello spazio di ricerca, e poi procede con movimenti iterativi, dalla posizione presente verso una posizione vicina (contigua), dove la decisione ad ogni passo è basata solamente sulla conoscenza locale. Anche se le decisioni locali sono fatte in un modo sistematico, gli algoritmi di ricerca tipicamente locali sono *incompleti*. Anche se le istanze del problema dato hanno una soluzione, loro non garantiscono di trovarla. I metodi di ricerca locale possono visitare la stessa posizione all'interno dello spazio di ricerca più di una volta. Esiste anche la possibilità di bloccarsi frequentemente da qualche parte nello spazio di ricerca, e questa posizione non possono abbandonarla senza meccanismi speciali, come un nuovo riavvio del processo di ricerca od un altro genere tipo di meccanismo.

Sembrerebbe che, dovuto alla loro *incompletezza*, gli algoritmi di ricerca locale sono generalmente inferiori a quelli sistematici. Ma, come vedremo più avanti, questo non è vero. Molti problemi sono impliciti di natura, si sa che sono risolvibili e quello che si richiede è di ottenere una soluzione, invece di decidere solamente l'esistenza di una soluzione. Questo è valido in particolare per i problemi di ottimizzazione, come il Problema del Commesso Viaggiatore (TSP) che consiste nel trovare una soluzione di sufficiente ottimalità, ma anche per problemi di decisione con sottovincoli che sono abbastanza comuni.

Secondo, in un tipico scenario applicativo, il tempo per trovare una soluzione è spesso limitato. Esempi per tali problemi di real-time, possono essere trovati in tutti i campi applicativi. Quasi ogni problema del mondo reale che comporta un'interazione con il mondo reale, ha vincoli di real-time. In queste situazioni, gli algoritmi sistematici devono essere spesso interrotti dopo che il tempo allocato è stato esaurito, e questo chiaramente li rende incompleti. Questo è particolarmente problematico per gli algoritmi sistematici di ottimizzazione che cercano attraverso gli spazi delle soluzioni parziali; se quelli di solito sono interrotti non è disponibile nessuna soluzione candidato, mentre nella stessa situazione, gli algoritmi di ricerca locale offrono tipicamente la migliore soluzione trovata finora[1]. Idealmente, gli algoritmi per problemi di real-time devono essere capaci di consegnare soluzioni ragionevolmente buone, in qualsiasi punto, durante la loro esecuzione. Per i problemi di ottimizzazione, questo significa che il run-time e la qualità della soluzione devono essere correlati positivamente, per i problemi di decisione si potrebbe immaginare di indovinare una soluzione quando accade un timeout, dove l'accuratezza dell'ipotesi dovrebbe aumentare con il run-time dell'algoritmo. Questa così detta proprietà di *any-time* dell'algoritmo, è di solito difficile da realizzare, ma in molte situazioni il modello di ricerca locale sembra andar meglio per concepire algoritmi *any-time* che modelli di ricerca sistematica.

Come una questione di fatto, gli algoritmi di ricerca sistematica e locale sono leggermente complementari nelle loro applicazioni. Un bell'esempio può essere trovato in [**78**], dove un algoritmo veloce di ricerca locale è utilizzato per trovare soluzioni per problemi di pianificazione, l'ottimalità di cui è dimostrata per mezzo di algoritmi sistematici. Diverse prospettive dello stesso problema possono in certi casi appellare ad algoritmi di ricerca locale, se sono richieste soluzioni buone in un tempo corto usando un calcolo parallelo e la conoscenza riguardo il dominio del problema è molto limitato. In altri casi, se sono richieste delle soluzioni esatte od ottimali, il vincolo del tempo è meno importante e possono essere sfruttate alcune conoscenze sul dominio di problema; in questo caso la ricerca sistematica sarà la scelta migliore. Finalmente, è evidente che per alcune classi di problemi, i metodi di ricerca locale e sistematica sono efficaci soprattutto per diverse sottoclassi di istanze.

Sfortunatamente, la questione generale su quando preferire il metodo di ricerca sistematica e quando quello locale rimane aperta.

Gli algoritmi di ricerca locale spesso si avvalgono pesantemente di meccanismi stocastici, come le euristiche. Questi algoritmi sono chiamati algoritmi di ricerca locale stocastica (SLS), e sono usati da molti anni nel contesto di problemi di ottimizzazione.

Fra gli algoritmi più rilevanti troviamo l'algoritmo di Lin-Kernighan per il problema del commesso viaggiatore, così come metodi generali come la ricerca tabu [**50**], e simulated annealing [**82**]. Ultimamente è diventato evidente che gli algoritmi stocastici di ricerca locale, possono essere applicati con successo nella risoluzione di problemi di decisione *NP-completi*, come il Graph Coloring Problem GCP [**59**], [**96**], [**97**] od il problema di soddisfacibilità nella logica proposizionale SAT [**116**], [**35**], [**56**].

---

[1] Si deve notare che, gli algoritmi di ricerca locale possono essere anche progettati in modo tale che possono cercare in uno spazio delle soluzioni parziali

Il secondo è particolarmente interessante per la ricerca nell'AI, in quanto non è soltanto uno dei problemi di base che sorgono nel contesto dell'approccio basato sulla logica, ma si può utilizzare anche nel contesto dell'approccio del risolutore generico di problemi dove problemi di base da altri domini, come Blocksworld Planning o Graph Coloring, sono risolti codificandoli nel SAT. In seguito si risolvono le istanze codificate usando tecniche stocastiche di ricerca locale. I recenti risultati della ricerca indicano che quest'approccio è vitale per almeno alcuni domini di problemi [**78**], [**68**] e di conseguenza oggi c'è un considerevole interesse, non solo nella comunità dell'AI, ma anche nelle questioni connesse a lui.

1.3 Risolutore di problemi con ricerca locale stocastica

L'interesse in algoritmi efficienti per problemi astratti e semplici, come il problema della soddisfacibilità nella logica proposizionale SAT, è basato sulla nozione di risolutore generale di problemi. Ci sono due versioni di quest'approccio indiretto, la più debole si basa sulla premessa che possono essere identificate delle tecniche generali per trattare con una serie larga di classi di problemi.

La versione più forte, partendo dal presupposto che è possibile risolvere problemi trasformandoli in qualche dominio, per il quale è disponibile un risolutore universale. Le soluzioni che sono così ottenute sono poi trasformate indietro, nel dominio del problema originale, che fornisce le soluzioni desiderate per il problema.

Quest'approccio di risolutore generico di problemi conta su un numero di premesse:

- prima di tutto, il dominio in cui i problemi sono trasformati (il dominio intermedio) deve fornire sufficiente potere espressivo, in modo tale che un numero significativo di classi di problemi deve essere rappresentabile in questo dominio;
- in seguito, la complessità delle trasformazioni tra i domini dovrà essere sufficientemente bassa confrontata alla complessità per trovare una soluzione, altrimenti le spese generali di questo approccio indiretto potrebbe superare facilmente i suoi vantaggi.
- finalmente, la premessa più importante è l'esistenza di un risolutore sufficientemente generale nel dominio intermedio.

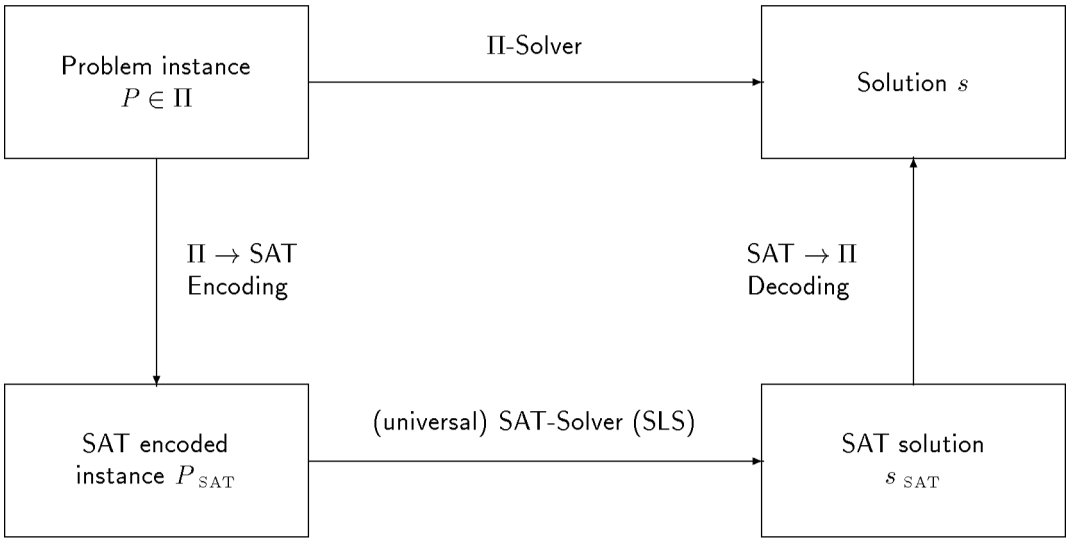

Figura 1.1: *Risolutore generico di problemi attraverso la trasformazione*

Si potrebbe argomentare che queste premesse sono critiche, nel senso che almeno per alcune di loro, c'è il dubbio su come possono essere incontrate in pratica. C'è un numero di domini che sono espressivi sufficientemente per rappresentare una larga varietà di classi di problemi. Per alcuni di questi, come il SAT ed il CSP, sono state trovate trasformazioni efficienti per molte classi di problemi, in modo tale che abbiamo indicazioni riguardo i costi addizionali della codifica / decodifica, che sono quasi trascurabili quando comparati ai costi per trovare le soluzioni.

Molto probabilmente ci sarà un collegamento stretto tra il problema da codificare e le prestazioni dell'algoritmo dato. Presumendo che si potrebbe trovare un algoritmo di dominio intermedio che raggiunge prestazioni ragionevoli su qualche codifica, per un'ampia serie di problemi, ancora non è chiaro se trovare codifiche appropriate è meno difficile che concepire algoritmi specializzati per i domini applicativi originali. Di conseguenza, il problema cruciale dell'approccio del risolutore generico di problemi è di trovare un algoritmo di dominio intermedio ed un numero di strategie di codifica abbastanza efficaci e semplici, che mostrino performance ragionevoli su un'ampia serie di problemi.

D'altra parte i vantaggi dell'approccio di risolutore generico sono ovvi. Prima di tutto, non c'è bisogno di nessun risolutore specializzato per il dominio del problema originale. Anche altri domini beneficiano dai miglioramenti dei risolutori di domini intermedi.

La versione debole dell'approccio di risolutore generico di problemi è ampiamente accettata nell'AI ed in molte tecniche per risolvere problemi, quali si sa di poter essere applicate con successo ad una varietà di domini di problemi. La variante forte, invece, è meno accettata. Ciononostante, i recenti risultati nel risolvere problemi codificati SAT, utilizzando tecniche di ricerca locale stocastica, danno alcune indicazioni sull'applicazione pratica della versione forte del risolutore.

Ci sono molte ragioni in relazione al SAT come un promettente dominio intermedio:

- Prima di tutto, SAT è *NP-completo*, significa che ogni problema *NP-completo* può essere ridotto polinomialmente al SAT. Così, esistono trasformazioni efficaci tra i problemi SAT e altri problemi di decisione *NP-completo*. Dato che la classe di problemi di decisione *NP-completo* copre una serie larga di problemi applicativi, SAT è abbastanza espressivo da agire come un dominio intermedio.
- Il secondo vantaggio del SAT è la sua semplicità concettuale. In quanto la sintassi, così come la semantica del SAT è molto semplice, specialmente quando limitato alle formule CNF, il SAT offre vantaggi sostanziali per ideare e valutare algoritmi.

Quindi, se il SAT sembra essere una scelta ragionevole per il dominio intermedio, perché non considerare gli algoritmi SLS come risolutori SAT?

Prima, come già menzionato, gli algoritmi SLS possono essere competitivi oppure anche superiori a quelli sistematici sia per i problemi SAT nativi, che per una larga serie di problemi da altri domini codificati nel SAT. Inoltre, la maggior parte degli algoritmi popolari SLS per il SAT sono relativamente facili da implementare perché sono concettualmente abbastanza semplici. Finalmente, la maggior parte di questi algoritmi SLS offre la facilità del confronto.

D'altra parte ci sono da menzionare gli svantaggi degli algoritmi SLS:

- Prima, gli algoritmi SLS tipici sono *incompleti* - non garantiscono di trovare una soluzione se esiste.
- Questi algoritmi sono estremamente *non deterministici*, che qualche volta è considerato un problema isolato (per esempio, quando sia affronta l'affidabilità e la riproducibilità dei risultati).

- Un inconveniente più importante è il fatto che questi algoritmi sono stati trovati estremamente difficili da analizzare e di conseguenza c'è poca comprensione teoretica del loro comportamento.

Ciononostante, i vantaggi, specialmente le performance superiori che possono essere raggiunte per tipi diversi di problemi SAT, sembrano superare tutti questi inconvenienti in modo tale che gli algoritmi SLS sono considerati ad essere fra i metodi più promettenti per risolvere problemi grandi e difficili di soddisfacibilità.

1.4 Problemi NP-completi e NP-hard

Oggi, la teoria della complessità computazionale è un campo con considerevole impatto su altre aree della scienza del computer. Nel contesto di questo lavoro, il campo primario di applicazione degli algoritmi di ricerca locale stocastica è rappresentato da una classe di problemi computazionalmente molto difficili, per la quale non si conoscono degli algoritmi efficienti. Inoltre, una grande maggioranza di esperti nella teoria della complessità pensa che sia impossibile l'esistenza di algoritmi efficienti per questi problemi.

La complessità di un algoritmo è definita in base al modello di macchina formale. Di solito, questi sono idealizzati, creati in un modo che facilitino il ragionamento formale riguardante il loro comportamento. Uno dei primi, e ancora forse il più evidente di questi modelli, è la macchina Turing. Per le macchine Turing e altre macchine formali, la complessità computazionale è definita in termini di requisiti di spazio e tempo per la computazione.

Di solito la teoria della complessità tratta con intere classi di problemi (generalmente set numerabile di istanze di problemi) invece di singole istanze. Per un algoritmo dato o modello di macchina, la complessità della computazione è caratterizzata dalla dipendenza funzionale fra la dimensione di una istanza e il tempo e lo spazio richiesti per risolvere quest'istanza. Per ragioni di trattabilità analitica, molti problemi sono formulati come problemi di decisione, e le complessità temporale e spaziale sono analizzate in termini del peggiore comportamento asintotico.

Data una definizione appropriata della complessità computazionale di un algoritmo per uno specifico problema, la complessità della classe di problemi può essere definita come la complessità del migliore algoritmo su questa classe di problemi. In quanto la complessità temporale rappresenta generalmente il fattore più restrittivo, le classi di problemi sono suddivise in classi di complessità riguardando il loro caso peggiore di complessità temporale.

Due particolarmente interessanti classi di complessità sono:

- la classe di problemi P, che possono essere risolte da una macchina deterministica in tempo polinomiale;
- la classe di problemi NP, che possono essere risolte da una macchina non deterministica in tempo polinomiale.

Chiaramente, ogni problema in P è anche inclusa in NP, fondamentalmente perché i calcoli deterministici possono essere emulati su macchina non deterministiche. Comunque, la domanda se anche $NP \subseteq P$, e di conseguenza $P = NP$, è uno dei problemi più rilevanti della scienza dei computer. Da quando molti dei problemi fondamentali sono in NP, ma possibilmente non in P (non è conosciuto nessun algoritmo deterministico in tempo polinomiale), questo così chiamato $P = NP\text{-}problem$ non è soltanto di interesse teoretico. Per questi problemi computazionalmente difficili, il miglior algoritmo conosciuto finora ha complessità temporale di tipo esponenziale. Perciò, con il crescere della dimensione dei problemi, le istanze dei problemi diventati rapidamente intrattabili, hanno poca efficacia sulla dimensione delle istanze del problema risolvibile in tempo ragionevole. Esempi ben

conosciuti di problemi *NP-hard* includono il problema della soddisfacibilità SAT, il problema Graph Coloring GCP, il problema del Commesso Viaggiatore TSP ed i problemi di scheduling.

Molti di questi difficili problemi NP possono essere tradotti in ogni altro in tempo polinomiale deterministico. Un problema che è alquanto difficile come alcun altro problema in NP (nel senso che ogni problema in NP può essere ridotto polinomialmente a lui) è chiamato *NP-hard*. Così, i problemi *NP-hard* in un certo senso possono essere considerati altrettanto difficili come ogni problema in NP.

I problemi *NP-hard* che sono contenuti in NP sono chiamati *NP-completi*; in un certo senso questi problemi sono i più difficili in NP.

Un risultato fondamentale della teoria della complessità indica che, per dimostrare che $NP = P$, è sufficiente trovare un algoritmo deterministico in tempo polinomiale per un singolo problema *NP-completo*. Questa è la conseguenza del fatto che tutti i problemi *NP-completi* possono essere codificati reciprocamente in tempo polinomiale. Oggi, la maggior parte degli scienziati pensa che $P \neq NP$; comunque, da lontano tutti gli sforzi per trovare una prova per quest'ineguaglianza sono stati senza successo, e ci sono alcune indicazioni che i metodi matematici odierni siano troppo deboli per risolvere questo problema fondamentale.

Molti problemi dai domini applicativi, come i problemi di scheduling e di pianificazione, sono in sostanza *NP-completi*, ed in generale non risolvibili efficientemente. Comunque, un problema essendo *NP-completo* o *NP-hard*, non vuol dire che è impossibile risolverlo efficientemente. Ci sono almeno in pratica, tre modi di trattare con questi problemi:

- tentare di trovare una sottoclasse relativa del problema che può essere risolta efficientemente;
- utilizzare algoritmi di approssimazione efficienti;
- utilizzare approcci stocastici.

Dobbiamo ricordare riguardo alla prima strategia, che *NP-hardness* è una proprietà di una intera classe di problemi P, mentre in pratica accade spesso solamente per le istanze di una certa sottoclasse $P' \subseteq P$. Chiaramente, in generale P' non potrà essere *NP-hard*, mentre per P potrebbe non esistere un algoritmo efficiente, ma è probabile che sia ancora possibile trovare un algoritmo efficiente per la sottoclasse P'.

Comunque, se il problema è di ottimizzazione e non può essere ristretto ad una sottoclasse efficientemente risolvibile, un'altra scelta potrebbe essere di accettare le approssimazioni di una soluzione invece di tentare di calcolare le soluzioni ottimali. Così, in molti casi la complessità computazionale del problema potrebbe essere ridotta sufficientemente per far diventare risolvibile il problema. In alcuni casi, consentendo un piccolo margine rispetto alla soluzione ottimale, si può risolvere deterministicamente il problema in tempo polinomiale. In altri casi, il problema di approssimazione rimane *NP-hard*, ma la riduzione dello sforzo computazionale è sufficiente per trovare soluzioni delle istanze del problema accettabili.

Comunque qualche volta non possono essere concepiti schemi di approssimazione efficienti, oppure il problema è un problema di decisione al quale non può essere applicato per niente la nozione di approssimazione. In questi casi, un'altra scelta è di concentrarsi su algoritmi probabilistici piuttosto che su quelli deterministici. Ad un primo sguardo, quest'idea ha una certa importanza: secondo la definizione della classe di complessità NP, almeno i problemi *NP-completi* possono essere risolti efficientemente da macchine non deterministiche. Certo, questo è di poco uso pratico, dato che per un algoritmo probabilistico questo vuol dire che, c'è in ogni caso un'opportunità piccola, di poter risolvere il problema dato in tempo polinomiale. In pratica, la probabilità di successo di tale algoritmo può essere

arbitrariamente piccola. Tuttavia, in numerose occasioni, sono stati trovati algoritmi probabilistici che sono stati più efficienti sui problemi *NP-completo* o *NP-hard* dei migliori metodi deterministici disponibili. In altri casi, i metodi probabilistici e deterministici si completano reciprocamente nel senso che, per certi tipi di istanze di problemi, sono stati trovati ad essere superiori sia l'uno che l'altro. Fra i problemi più importanti e conosciuti da questa categoria, sono i problemi della soddisfacibilità nella logica proposizionale SAT e con soddisfacimento di vincoli CSP.

1.5 Logica proposizionale e soddisfacibilità

1.5.1 Logica proposizionale

La logica proposizionale è basata su un linguaggio formale basato su un alfabeto che comprende i simboli proposizionali, le costanti logiche e gli operatori logici. Usando gli operatori logici, i simboli proposizionali e le costanti logiche sono combinati in formule proposizionali che rappresentano le espressioni proposizionali. Formalmente, la sintassi della logica proposizionale può essere definita dalla:

Definizione 1.1 (la sintassi della logica proposizionale)

> $S = V \cup C \cup O\{(,)\}$ è l'alfabeto della logica proposizionale con $V = \{x_i \mid i \in N\}$ che denota il set dei simboli proposizionali, $C = \{T, F\}$ il set di costanti logiche e $O = \{\neg, \wedge, \vee, \Leftrightarrow, \quad\}$ il set di connettivi logici.

Il set di formule proposizionali è caratterizzato dalla seguente definizione induttiva:

- le costanti logiche T e F sono formule proposizionali;
- ogni simbolo proposizionale P o Q è una formula proposizionale;
- se P è una formula proposizionale, allora anche $\neg P$ è una formula proposizionale;
- se P e Q sono formule proposizionali, allora anche $(P \wedge Q)$ e $(P \vee Q)$ sono formule proposizionali.

Gli assegnamenti sono mappature dei simboli proposizionali nella tavola di verità. Usando le interpretazioni standard dei connettivi logici sulle costanti logiche, le assegnazioni possono essere usate per valutare le formule proposizionali. Così, noi possiamo definire la semantica della logica proposizionale:

Definizione 1.2 (la semantica della logica proposizionale)

> Il set di simboli $Var(P)$ della formula P è definito come il set di tutte i simboli che appaiono in P.
>
> L'assegnazione del simbolo della formula P è una mappatura $B : Var(P) \mapsto C$ del set di simboli di *P* nelle costanti logiche. Il set completo delle assegnazioni di *P* è denotato da *Assign(P)*.

Le costanti logiche T e F sono note anche come $verum$ e $falsum$. Le tavole di verità dei connettivi logici negazione $\neg$, congiunzione $\wedge$ e disgiunzione $\vee$ sono intuitivi, in seguito saranno presentate soltanto le tavole di verità dell'implicazione e l'equivalenza.

P	Q	$P \Rightarrow Q$	$P \Leftrightarrow Q$
F	F	T	T
F	T	T	F
T	F	F	F
T	T	T	T

Figura 1.2: *Le tavole di verità dei connettivi logici implicazione e equivalenza*

Dovuto al fatto che il set di simboli di una formula proposizionale è sempre limitato, anche il set completo delle assegnazioni per una formula data è limitato. Più precisamente, per una formula che contiene n simboli ci sono esattamente 2^n assegnamenti di simboli.

1.5.2 Soddisfacibilità

Consideriamo il seguente problema diplomatico proposto da Brian Hayes [**58**]:

"Lei è capo del protocollo per il ballo all'ambasciata. Il principe vi consiglia di invitare Perù o di escludere Qatar. La regina vi chiede di invitare o Qatar o Romania o entrambi. Il re, in un umore dispettoso, vuole offendere sia Romania che Perù, oppure entrambi. Esiste un elenco degli ospiti che soddisferà i capricci dell'intera famiglia reale?".

Vedremo in seguito come questo problema può essere rappresentato come un problema di soddisfacibilità proposizionale. Definiamo la soddisfacibilità come segue:

Definizione 1.3 (Soddisfacibilità & Validità)

> Un assegnazione della variabile B è un $modello\ della\ formula\ F$, esattamente se $Val_B(F)=T$; in questo caso diciamo che $B\ soddisfa\ F$.
>
> Una formula *F* è *valida*, se è soddisfatta da tutte le sue assegnazioni delle variabili. Le formule valide sono chiamate anche *formule analitiche* o *tautologie*.
>
> Se per una formula *F* esiste almeno un modello, *F* è *soddisfacibile*.

Adesso possiamo definire formalmente, il problema della soddisfacibilità SAT e il problema di validità VAL per la logica proposizionale.

Definizione 1.4 (i problemi SAT e VAL)

> Per il problema di soddisfacibilità SAT nella logica proposizionale, lo scopo è di decidere per una formula data F, se è o no soddisfacibile.
>
> Per il problema di validità (VAL) nella logica proposizionale, lo scopo è di decidere per una formula data *F*, se è o no valida.

Per SAT, ci sono due versioni del problema, la versione di decisione e la versione di scoperta di modelli. Nella versione di decisione, è richiesta solamente una decisione di si/no per la soddisfacibilità della formula data, mentre per la versione di scoperta di modelli, nel caso che la formula è soddisfacibile, deve essere trovato un modello.

Notiamo che c'è una semplice relazione tra SAT e VAL: per ogni formula F, F è valida se e soltanto se $\neg F$ non è soddisfacibile, una formula è valida esattamente se la sua negazione non ha un modello. Per questo motivo, le procedure di decisione per SAT possono essere utilizzate per decidere VAL e viceversa. Comunque, questo non è valido per le procedure incomplete, come gli algoritmi SLS per SAT.

1.5.3 Forme normali

Spesso, i problemi come SAT e VAL sono studiati per classi sintatticamente ristrette di formule. Imponendo delle restrizioni sintattiche di solito sono facilitati gli studi teoretici e possono essere anche molto utili per semplificare la progettazione dell'algoritmo. Le forme normali sono formule sintatticamente ristrette in modo tale che per una formula arbitraria F c'è sempre almeno una formula in forma normale F'. Così, ogni forma normale induce una sottoclasse di formule proposizionali che è altrettanto potente come la logica proposizionale. Poiché per certe classi di formule sintatticamente ristrette non è provato se loro sono o non sono forme normali, useremo il termine *forma normale* in un senso più generale, coprendo sottoclassi della logica proposizionale che è caratterizzata da restrizioni sintattiche. Alcune delle forme normali comunemente utilizzate sono classificate dalle seguenti definizioni.

Definizione 1.5 (forme normali)

Un letterale è un simbolo proposizionale (chiamato letterale positivo) o la sua negazione (chiamato letterale negativo). Formule dalla forma sintattica $c_1 \wedge c_2 \wedge \ldots \wedge c_m$ si chiamano *congiunzioni*, mentre le formule dalla forma $d_1 \vee d_2 \vee \ldots \vee d_n$ si chiamano *disgiunzioni*.

Una formula proposizionale F è in *forma normale congiuntiva* (CNF), se è una congiunzione su disgiunzioni di letterali. Le disgiunzioni state chiamate *clausole*. Una formula CNF F è in kCNF, se tutte le clausole di F contengono esattamente k letterali.

Una formula proposizionale F è in *forma normale disgiuntiva* (DNF), se è una disgiunzione di congiunzioni di letterali. In questo caso, le congiunzioni sono chiamate *clausole*. Una formula DNF F è in kDNF, se tutte le clausole di F contengono esattamente k letterali.

Una formula CNF F è *Horn*, se ogni clausola in F contiene al massimo una variabile non negata.

La clausola che contiene solo un letterale è chiamata clausola unità.

La clausola che non contiene alcun letterale è chiamato la clausola vuota ed è interpretata come *False*.

Possiamo anche, mettere dei vincoli sulla dimensione delle clausole nel problema. Per esempio, kSAT è la classe di problemi di decisione nella quale tutte le clausole sono di lunghezza k. kSAT è *NP-completo* per qualsiasi $k \leq 3$ ma è polinomiale per $k = 2$ [**31**]. Il problema di ottimizzazione Max-SAT è collegato strettamente al problema di decisione SAT.

Dato un set di clausole, qual'è il numero massimo di clausole che possono essere soddisfatte?

Con alcune semplici modifiche, molte delle procedure complete e di approssimazione per SAT possono essere adattate per risolvere i problemi di Max-SAT.

Ritornando all'esempio considerato precedentemente [**1.5.2**], possiamo esprimere i tre vincoli tramite una formula proposizionale:

$$(P \vee \neg Q) \,\&\, (Q \vee R) \,\&\, (\neg R \vee \neg P)$$

dove P, Q e R sono variabili booleani che sono *True* se e solamente se noi invitiamo, rispettivamente, Perù, Qatar e Romania. Una soluzione al problema è un assegnamento soddisfacente, un assegnamento dei valori di verità alle variabili booleane che soddisfano la formula proposizionale. In questo caso, ci sono solo due assegnazioni soddisfacenti (delle otto

possibili). Queste assegnano a P e Q il valore $True$ e ad R il valore $False$, oppure assegniamo $False$ a P e Q e $True$ a R.

Ovvero, o possiamo invitare gli entrambi Perù e Qatar e offendere Romania, o possiamo invitare solo la Romania e offendere sia Perù che Qatar. La soddisfacibilità proposizionale è un problema molto semplice e rappresenta un elemento importante nella teoria della complessità computazionale. È stato mostrato che la soddisfacibilità proposizionale è stato il primo problema ad essere *NP-completo* [**13**]. Nonostante un quarto di secolo di sforzi, esso resta una questione aperta anche se nel caso peggiore, problemi di questo tipo possono essere risolti in tempo polinomiale. Come i migliori algoritmi completi sono esponenziali, generalmente *NP-completezza* si considera marcare il confine tra la trattabilità e l'intrattabilità. Ciononostante, la ricerca negli ultimi anni ha mostrato che spesso i problemi di soddisfacibilità possono essere efficientemente risolti in pratica.

1.6 Problemi con soddisfacimento di vincoli (CSP)

Un *problema con soddisfacimento di vincoli* (CSP) è definito da un set di variabili, ognuna delle quali può prendere un numero di *valori* da qualche dominio ed un insieme di *vincoli* che riguardano una o più variabili. Possono essere definiti vari tipi di CSP in base al fatto che i domini delle variabili sono di tipo discreto o continuo, finiti o infiniti.

Definizione 1.6 (CSP discreto e limitato)

> Un problema con soddisfacimento di vincoli può essere definito come un tripla $P=(V,D,C)$, dove $V=\{V_1,V_2,\ldots,V_n\}$ è un set limitato di n variabili, $D=\{D_1,D_2,\ldots,D_n\}$ un set di domini e $C=\{C_1,C_2,\ldots,C_k\}$ un set finito di vincoli.
>
> P è un CSP *discreto finito* se tutti i domini D_i sono distinti e finiti.
>
> L'*assegnazione della variabile* per P è un vettore $A=\{d_1,d_2,\ldots,d_n\}$ che contiene esattamente un valore $d_i \in D_i$ per ogni variabile V_i.

Certamente, per questo tipo di CSP il set di tutti gli assegnamenti possibili è limitato. In modo simile al SAT, la classe CSP discreto e finito è *NP-completo*. Questo può essere provato facilmente cosi come c'è un stretto collegamento tra il SAT ed il CSP discreto e finito: il SAT sulle formule CNF corrisponde direttamente ad un CSP discreto finito dove tutti i domini contengono solamente i valori booleani T e F, ed ogni vincolo contiene esattamente tutti gli assegnamenti soddisfacenti di una particolare clausola. Viceversa, ogni CSP discreto finito può essere trasformato direttamente in una istanza SAT, dove ogni variabile proposizionale rappresenta un particolare assegnamento di una variabile CSP e ogni vincolo è rappresentata da un numero di clausole. Notiamo che generalmente questo tipo di codifica diretta non produce una formula CNF, ma una formula in CDNF, quale può essere poi trasformata in CNF.

Anche se CSP sembra essere più adeguato del SAT per la rappresentazione di problemi combinatoriali difficili, il SAT è un dominio sintatticamente più semplice, e perciò progettare ed analizzare algoritmi tende ad essere più facile per il SAT che per il CSP. Nello stesso tempo, possono essere generalizzati direttamente in CSP più algoritmi SLS per il SAT.

Di conseguenza, i concetti e gli algoritmi per i risolutori SAT sono più avanzati in molti particolari dello sviluppo corrispondente in CSP.

1.7 Problemi random

Esistono due modelli popolari per i problemi random di soddisfacibilità:

- il modello con clausole di lunghezza fissa;

- il modello a probabilità costante.

Nel modello con clausole di lunghezza fissa (spesso chiamato il modello kSAT random), generiamo problemi con N variabili e L clausole come segue: per ogni clausola è selezionato un sottoinsieme random di dimensione k delle N variabili, e ogni variabile è fatta positiva o negativa con probabilità $1/2$. Così, tutte le clausole hanno la stessa dimensione.

Nel modello a probabilità costante, generiamo ognuna delle L clausole del problema, includendo con probabilità p ognuno dei $2N$ letterali possibili. Dopo l'inclusione delle clausole vuote od unità i problemi diventano più facili, cosi, molti studi utilizzano una versione del modello a probabilità costante proposto in [**62**] in cui, se una clausola generata contiene solamente uno o nessun letterale, è scartata ed è generata un'altra clausola al suo posto. Noi chiamiamo questo il $modello\ CP$.

2 Procedure complete

2.1 La procedura di Davis-Putnam

Nonostante la sua età e semplicità, la procedura Davis-Putnam rimane una delle migliori procedure complete per la soddisfacibilità [**22**]. La procedura di Davis-Putman originale [**20**] si basa sulla regola di risoluzione che elimina le variabili una ad una, in seguito aggiungendo tutte le soluzioni possibili alla serie di clausole. Sfortunatamente, in generale questa procedura richiede spazio esponenziale. Perciò, Davis, Logemann e Loveland hanno sostituito rapidamente la regola di risoluzione con una split rule che divide il problema in due sottoproblemi più piccoli [**19**]. Nella letteratura, questa procedura è chiamata imprecisamente la $procedura\ Davis\text{-}Putnam$ o DP.

procedure $DP(\Sigma)$
SAT **if** Σ empty then return satisfiable
EMPTY **if** Σ contains an empty clause **then return** unsatisfiable
TAUTOLOGY **if** Σ contains a tautologous clause c **then return** $DP(\Sigma - \{c\})$
UNIT PROPAGATION **if** Σ contains a unit clause $\{l\}$
then return $DP(\Sigma$ simplified by assigning l to $True)$
PURE LITERAL DELETION **if** Σ contains a literal l but not the negation of l
then return $DP(\Sigma$ simplified by assigning l to $True)$
SPLIT **if** $DP(\Sigma$ simplified by assigning l to $True)$ is satisfiable **then return** satisfiable
else return $DP(\Sigma$ simplified by assigning l to $True)$

***Figura 2.1**: La procedura Davis-Putman.*

Dopo l'applicazione della regola di suddivisione (split rule), semplifichiamo il set delle clausole cancellando ogni clausola che contiene il letterale l su $True$, spesso chiamata sussunzione di unità (unit subsumtion) e cancellando la negazione di l tutte le volte che si presenta nelle clausole rimanenti, spesso chiamata unit resolution. Notiamo che la regola sussunzione di unità può essere ignorata se la procedura DP è terminata quando ad ogni variabile è stato assegnato il valore $True$.

A causa del suo costo, la pure literal rule è spesso cancellata. Il modo simile alle tautologie che possono essere eliminate solamente all'inizio della ricerca, la regola tautologica è spesso cancellata. Nessuna delle due è richiesta per la correttezza o la completezza della procedura. Neanche la regola unit propagation è necessaria per la correttezza o la completezza della procedura; cosi come le split rule ed empty rule raggiungono lo stesso effetto. Comunque, la regola unit propagation contribuisce tantissimo all'efficienza della procedura Davis-Putnam. Per i problemi Horn-SAT, la regola unit propagation è la base per determinare la soddisfacibilità di una procedura completa.

Hooker ha mostrato come adattare la procedura di Davis-Putnam per determinare l'aumento della soddisfacibilità in modo efficiente [**64**]. Dato un set di clausole soddisfacenti Σ, il problema è di decidere la soddisfacibilità di $\cup\{C\}$ per qualche nuova clausola C.

2.2 Euristiche di ramificazione

L'algoritmo Davis-Putnam è non deterministico, poiché possiamo scegliere il letterale su cui ramificare. Una euristica di ramificazione conosciuta e conveniente è quella di Mom. Questa sceglie il letterale che si presenta più spesso nelle clausole di dimensione minima. I collegamenti di solito sono rotti con un ordine statico o casuale. In alcuni casi, calcolando il letterale che si presenta più spesso è considerato troppo costoso e si ramifica semplicemente sul primo letterale della clausola di dimensione minore.

L'euristica di Jeroslow-Wang [**68**] valuta il contributo che ogni letterale è probabile a far soddisfare il set di clausole. Ciascun letterale è punteggiato come segue: per ogni clausola c nella quale appare il letterale, è sommato $2^{-|c|}$ al punteggio del letterale, dove $|c|$ rappresenta il numero di letterali in c. Poi ad ogni letterale con il punteggio più alto è applicata la split rule.

Hooker e Vinay hanno investigato la motivazione riguardo la funzione punteggio utilizzata nell'euristica di Jeroslow-Wang [**65**]. Loro propongono l'*ipotesi di soddisfazione*, che è la migliore per dividerla in sottoproblemi che hanno più probabilità ad essere soddisfacibili, ma rifiutano questa in favore dell'*ipotesi di semplificazione*, che è la migliore per dividerla in semplici sottoproblemi con clausole più corte e più semplici dopo la unit propagation. L'ipotesi di semplificazione suggerisce una regola Jeroslow-Wang *two-sided* che funziona meglio della regola Jeroslow-Wang originale.

Una delle migliori realizzazioni correnti della procedura di Davis-Putnam, SATZ utilizza un'euristica basata sulla massimizzazione del numero di unit propagation [**88**]. Questa aggiunge un ulteriore appoggio all'ipotesi di semplificazione di Hooker e Vinay.

2.3 Strutture di dati e la loro realizzazione

Le performance della procedura di Davis-Putnam dipendono in modo critico dalla cura nella realizzazione. Crawford e Auton hanno presentato un algoritmo che fa unit resolution ed unit subsumtion in tempo lineare [**15**]. Per ogni variabile, ci sono liste di clausole che contengono variabili positive e negative. Ogni clausola ha un counter che contiene il numero di letterali ancora stati assegnati, ed un flag *inattivo* se la clausola è stata sussunta. Quando una variabile è assegnata su *True*, il counter delle clausole attive è decrementato quando la variabile è negativa, e quando la variabile è positiva quello delle clausole attive è messo su *inattivo*. Quando una variabile è assegnata su *False*, cambi analoghi sono compiuti. Zhang e Stickel mostrano come migliorare leggermente questo compiendo unit resolution in tempo lineare ammortizzato, ed unit subsumtion in tempo ammortizzato costante [**132**]. Questa modifica porta ad un miglioramento notevole dell'algoritmo di Crawford e Auton su vari problemi SAT.

Gli alberi di discriminazione (spesso chiamati *tries*) si sono dimostrati utili per una rapida indicizzazione in una varietà di aree di ragionamento automatizzato come la riscrittura dei vocaboli e la dimostrazione dei teoremi. Zhang e Stickel hanno comparato il loro utilizzo nella procedura di Davis-Putnam con i metodi a base di liste descritti sopra [**131**]. In una procedura basata su alberi di discriminazione, il set di clausole è immagazzinato come una struttura di dati ad albero i cui margini sono etichettati dai letterali, e ogni percorso rappresenta un set di letterali in una delle clausole. Unit resolution può essere compiuta poi da un'operazione di merging. I risultati sperimentali e teoretici di Zhang e Stickel suggeriscono uno schema ibrido, in cui le clausole sono rappresentate come alberi di discriminazione ma sono utilizzate liste per identificare rapidamente clausole nelle quali la variabile si presenta positiva e negativa. Questo schema è implementato nella procedura SATO [**128**].

2.4 Problemi non clausolari

La descrizione precedente della procedura di Davis-Putnam ha ammesso l'ipotesi che i problemi siano nella forma CNF. Questo non è essenziale. Per esempio, Otten ha usato una rappresentazione a matrice per estendere la procedura di Davis-Putnam alle formule non clausolari [**100**]. Esperimenti su problemi non clausolari hanno mostrato che questa procedura Davis-Putnam non clausolare esegue il suo compito meglio di una procedura Davis-Putnam standard usando la traduzione diretta nella forma clausolare (che è esponenziale nel peggiore dei casi), così come la traduzione definizionale (che è quadratica, ma introduce variabili addizionali).

2.5 Backtracking intelligente ed apprendimento

La procedura di Davis-Putnam standard compie un backtracking cronologico, esplorando completamente un ramo dell'albero di ricerca prima del backtracking e poi esplorando l'altro. Possiamo migliorare su questo adattando alcune delle migliori tecniche sviluppate dalla comunità constraint satisfaction come conflict-directed backjumping e nogood learning. Il conflict-directed backjumping percorre all'indietro l'albero di ricerca verso la causa del fallimento, ignorando gli assegnamenti irrilevanti delle variabili. Il nogood learning memorizza le cause dei fallimenti per prevenire simili errori che potrebbero essere fatti in altri rami. Bayardo e Schrag hanno descritto come gli entrambi meccanismi possono essere implementati all'interno della procedura di Davis-Putnam [**4**]. Loro hanno mostrato che, con questi miglioramenti (specialmente nella relevance-bounded learning), la procedura di Davis-Putnam può funzionare così bene o meglio delle migliori procedure di ricerca locale, come il GSAT ed il WalkSAT, su una larga varietà di problemi benchmark.

2.6 Errori iniziali

Un problema con una procedura completa come Davis-Putnam è che un errore iniziale può essere molto costoso. In [**38**], Gent e Walsh hanno identificato i problemi SAT che erano tipicamente facili ma occasionalmente potevano far fallire la procedura di Davis-Putnam. Per esempio, su un problema CP, la prima divisione ha portato immediatamente ad un sottoproblema non soddisfacibile che ha rilevato 350 milioni rami da risolvere. Nel tornare indietro verso la radice dell'albero di ricerca, è stata trovata un'istanza soddisfacente in un altro ramo senza nessuna ricerca. Gomes, Selman e Kautz hanno mostrato che una strategia di randomization e riavvii rapidi spesso possono contrastare efficacemente gli errori iniziali [**54**]. Meseguer e Walsh hanno mostrato che altre modifiche della strategia di ricerca depth-first come la ricerca limited discrepancy e la ricerca interleaved depth-first possono contrastano questi errori iniziali [**95**].

2.7 Parallelization

Le split rule non deterministiche suggeriscono un meccanismo semplice per parallelizzare la procedura di Davis-Putnam. Piuttosto che esplorare un ramo, tornare indietro, e poi esplorare l'altro, possiamo esplorare i due rami in parallelo. Comunque, dobbiamo decidere quando esplorare rami in sequenza e quando esplorarli in parallelo. Dobbiamo bilanciare anche il carico sui diversi microprocessori quanto tempo l'esplorazione di un ramo può terminare prima dell'esplorazione dell'altro. Zhang, Bonacina e Hsiang hanno perfezionato una procedura distribuita master-slave Davis-Putnam e utilizzata per risolvere problemi di esistenza dei quasigroup [**130**].

2.8 Limiti superiori

Sono stati dedotti vari limiti superiori per la performance della procedura Davis-Putnam e per altre procedure complete sui problemi 3SAT. Il peggiore caso di complessità temporale del Davis-Putnam è $O(1.696^N)$, ed una piccola modifica migliora questo portandolo ad $O(1.618^N)$ [**14**]. Rodosek ha presentato un algoritmo che arriva a $O(1.476^N)$ prendendo decisioni binarie che eliminano almeno due variabili [**110**]. Beigel ed Eppstein hanno presentato un algoritmo che riduce il problema 3SAT in un problema 2SAT, e poi tenta di estendere la soluzione di questo al problema originale [**7**]. Nel caso peggiore, questo algoritmo arriva a $O(1.381^L)$. Comunque, quanto tempo il numero delle clausole L di solito è molto più grande del numero delle variabili N, spesso questo non rappresenta un miglioramento. Effettivamente, $O(1.476^N)$ il limite dell'algoritmo di Rodosek è migliore per $L/N=1.206$.

2.9 Risoluzione

Van Gelder e Tsuji hanno migliorato la procedura di Davis-Putnam con una regola 2-closure che utilizza una versione limitata di risoluzione per generare tutti le soluzioni possibili con 2 o meno letterali [**124**]. Quest'algoritmo è stato molto competitivo con i problemi 3SAT random molto difficili e di analisi di circuit-fault. Più recentemente, Dechter e Rish hanno difeso la procedura di Davis-Putnam originale [**21**]. Loro hanno mostrato che una versione di questa procedura che loro chiamano *directional resolution*, lavora bene su varie classi trattabili come i problemi 2SAT così come su certi tipi di problemi strutturati. Loro hanno proposto anche una procedura approssimata, chiamata *bounded directional resolution*, che limita la dimensione delle soluzioni per ottenere lo spazio di complessità di tipo polinomiale. Questa procedura di approssimazione potrebbe essere utile come un passo di pre-processing per la procedura DP.

2.10 Tecniche OR

Sono state utilizzate varie tecniche *OR* per determinare la soddisfacibilità di un set di clausole. Per esempio, Hooker ha sviluppato un algoritmo *cutting plane* per risolvere i problemi SAT [**61**]. Hooker aveva combinato questo con un algoritmo *branch-and-bound* per ottenere un algoritmo *branch-and-cut* che mostra qualche promessa [**63**]. Kamath e al. hanno avuto successo anche con un metodo del punto interno [**75**].

2.11 Approcci logici

Sono state fatte molte ricerche sulle tecniche SAT da quelli che implementano dimostratori di teoremi del primo ordine. Quanto tempo il SAT è una sottoclasse della logica del primo ordine, alcuni dimostratori di teoremi dovevano contenere inevitabilmente qualche risolutore SAT, anche se si era concentrato poco sulla sua costruzione. Alcuni di questi dimostratori, per esempio la risoluzione basata su Otter [**94**], ha avuto un rilevante successo e deve essere trattata insieme alla logica proposizionale. Spesso la linea tra proposizionale ed il ragionamento del primo ordine è confusa, per esempio in sistemi che permettono formule quantificate solamente su domini finiti.

Un approccio attraente per trattare con la logica proposizionale è il metodo con *analitic tableaux* di Smullyan [**119**]. Questo ha il vantaggio di non necessitare ingressi nelle formule clausali, ed è veramente facile per il ragionamento umano su piccole formule. Una tecnica relativa è il KE calcolus che qualche volta può migliorare in modo esponenziale rispetto a tableaux [**18**]. Crawford e Auton [**15**] riferiscono che, avendo cominciato con un calcolo di tableaux, il loro programma *Tableau* potrebbe essere considerato come una delle migliori realizzazioni di Davis-Putnam.

3 Procedure di approssimazione

3.1 Procedura di semi-decisione

Papadimitriou ha proposto una procedura di semidecisione per il 2SAT, che modifica ripetutamente l'assegnamento di verità per una variabile in una clausola non soddisfacibile [**102**]. Questa soddisfa la clausola ma può fare diventare non soddisfacibili altre clausole. Utilizzando un argomento basato sulla rovina del giocatore d'azzardo, Papadimitriou dimostra che, se il problema è poi soddisfacibile questa procedura troverà un assegnamento soddisfacente in $O(N^2)$ cambiamenti, con una probabilità che si avvicina ad 1.

procedure $2SAT(\Sigma)$
    $T := random(\Sigma)$; random truth assignment
    repeat until T satisfies Σ
        v:=$variable$ in an unsatisfied clause
        $T := T$ with truth assignment for v flipped
    end

Figura 3.1: *La procedura di semidecisione di Papadimitriou per 2-SAT*

Esiste anche una procedura di decisione per i problemi Horn-SAT, che modifica le variabili nelle clausole non soddisfacibili e che impiega al massimo N cambiamenti [**103**].

Questa procedura ha due fasi. Nella prima fase, comincia con un assegnamento che assegna $False$ ad ogni variabile. Allora soddisfa le clausole Horn che contengono un letterale positivo, modificando il valore assegnato ai letterali positivi. Nella seconda fase, esamina le rimanenti clausole negative per determinare se hanno un assegnamento soddisfacente.

procedure $HornSAT(\Sigma)$
    $T := all\text{-}false(N)$; all N vars set to $False$
    for $v := positive\ literal$ in Horn clause **do**
        $T := T$ with truth assignment for v flipped
    end
    if T satisfies Σ **return** "satisfiable"
        else "unsatisfiable"

Figura 3.2: *La procedura di decisione di Papadimitriou per Horn-Sat*

3.2 Algoritmo greedy semplice

Koutsoupias e Papadimitriou hanno presentato una procedura di approssimazione semplice per la soddisfacibilità, che in seguito hanno chiamato algoritmo $greedy$ (goloso) [**85**]. In questa procedura, si sceglie a caso un assegnamento di verità. Poi si sceglie una variabile tale che modificando i valori di verità aumenta il numero di clausole soddisfatte. Non sono permesse le mosse laterali. Ripetiamo finché nessun miglioramento è possibile. Se questa è una soluzione, consideriamo successo, altrimenti abbandoniamo. Koutsoupias e Papadimitriou hanno dimostrato che questa procedura troverà un assegnamento soddisfacente almeno per tutti i problemi soddisfacibili con $O(N^2)$ clausole.

Anche Gent ha proposto un algoritmo greedy altrettanto semplice e che mostra performance ancora migliori. In quest'algoritmo, si assegna semplicemente una variabile secondo la polarità che ha avuto più spesso. Questo troverà un assegnamento soddisfacente per almeno tutti i problemi soddisfacibili con $O(Nlog(N))$ clausole [**32**].

```
procedure greedy(Σ)
        T:=random(Σ); random truth assignment
        repeat until no improvement possibile
                T:=T with a truth assignment flipped that increases
                        number of satisfied clauses
        end
        if T satisfies Σ return "satisfiable"
                else "no satisfying assignment found"
```

Figura 3.3: *L'algoritmo greedy di Koutsoupias e Papadimitriou*

```
procedure greedy(Σ)
        T:=nil; no truth assignment
        for v:=1 to N
                if v occurs more often positively in Σ then T:=T∪{v/True}
                        else T:=T∪{v/False}
        end
        if T satisfies Σ return "satisfiable"
                else "no satisfying assignment found"
```

Figura 3.4: *L'algoritmo greedy di Gent*

3.3 GSAT

La procedura GSAT [**116**] aggiunge riavvii e mosse laterali all'algoritmo greedy di Koutsoupias e Papadimitriou. Nonostante la sua semplicità, GSAT e le sue varianti hanno buone performance con una larga varietà di problemi di soddisfacibilità. GSAT comincia con un assegnamento di valori di verità alle variabili generate random, e sale modificando l'assegnamento della variabile che determina il maggior incremento del numero totale di clausole non soddisfatte.

```
procedure GSAT(Σ)
        for i:=1 to Max-tries
                T:=random(Σ); random truth assignment
                for j:=1 to Max-flips
                        if T satisfies Σ then return T
                        else T:=T with truth assignment flipped to maximize number of satisfied clauses
                end
        end
        return "no satisfying assignment found"
```

Figura 3.5: *L'algoritmo di ricerca locale GSAT*

Se non esiste nessun assegnamento della variabile che può essere modificato per aumentare il risultato, è modificata una variabile che non cambia il risultato. Senza tali modifiche laterali, le performance di GSAT diminuiscono.

Le procedure di ricerca locale, come il GSAT, necessitano che l'implementazione sia fatta con qualche cura. Per ogni iterazione, è modificata soltanto una variabile, e si calcola il numero di clausole soddisfate dopo che la variabile è stata cambiata. Per N variabili e L clausole, il lavoro richiesto è $O(NL)$. Al raggiungimento del numero massimo d'iterazioni, fissato dalla variabile $Max\text{-}flips$, il GSAT ricomincia con un nuovo assegnamento random.

Sebastiani ha descritto come modificare il GSAT in modo che si possa usare con formule non clausali [**46**]. Lui ha mostrato come calcolare il numero di clausole che potrebbero essere soddisfatte, se le formule fossero convertite in CNF senza costruire la conversione di CNF stessa. Questo calcolo impiega tempo lineare con la dimensione della formula in ingresso.

3.4 Procedura Gu

Nello stesso momento quando è stato presentato il GSAT, Gu ha proposto varie famiglie di procedure di ricerca locale per il SAT [**55**]. Queste procedure presentano alcune somiglianze con il GSAT. Comunque, loro hanno una struttura di controllo diversa che li permette di muoversi lateralmente quando sono possibili mosse in salita. Non è stato fatto ancora nessun paragone diretto tra le procedure di Gu e la procedura GSAT con le sue versioni.

```
procedure SAT1.0(Σ)
    T := random(Σ); random truth assignment
    while T does not satisfy Σ do
        for i := 1 to number of variables
            if T with v_i flipped does not decrease score then
                T := T with v_i flipped
        if local then flip some variables at random
    end
```

Figura 3.6: *La famiglia di procedure SAT 1.0 introdotte da Gu*

3.5 Pesi

Una modifica della procedura GSAT, che può essere molto utile su una certa classe di problemi, consiste nei *pesi delle clausole*, introdotte in [**112**] e [**99**]. Per ogni clausola è associato un peso che è variato durante la ricerca per focalizzare l'attenzione sulle clausole più difficili. La funzione risultato è la somma dei pesi delle clausole soddisfatte. Selman e Kautz hanno cambiato originalmente solamente i pesi alla fine di ogni prova. Frank ha mostrato che GSAT e HSAT, possono essere migliorate se i pesi sono cambiati dopo ogni modifica [**25**].

3.6 Random walk

Il successo della procedura di semidecisione di Papadimitriou per il 2SAT e la sua procedura di decisione per il Horn-SAT, illustra i benefici nel focalizzarsi sulle variabili in clausole non soddisfatte (le chiameremo *variabili non soddisfatte*).

Una versione del GSAT, chiamata GSAT con *random walk* [**114**] focalizza alcuni dei suoi sforzi di ricerca sulle variabili non soddisfatte. Il GSAT con random walk modifica con probabilità p una variabile in una clausola non soddisfatta, altrimenti sale. Modificare una variabile non soddisfatta può far decrescere il numero delle clausole soddisfatte. Ciononostante, il random walk migliora notevolmente le performance del GSAT. Sembra una strategia migliore modificare una variabile non soddisfatta, invece di modificare una variabile random con probabilità p. In tutti gli assegnamenti soddisfacenti, almeno una delle variabili in una clausola non soddisfatta deve avere il valore di verità opposto. Perciò, dobbiamo modificare almeno una di loro che portano ad un assegnamento soddisfacente.

Siccome la ricerca del GSAT è governata dalle variabili di questo set di clausole non soddisfatte, modificare una variabile non soddisfatta introduce diversità nella ricerca. In [**41**], Gent e al. hanno mostrato che, uno dei più grandi benefici nell'aggiungere random walk è la riduzione della sensibilità delle performance al parametro $Max\text{-}Flips$.

3.7 WalkSAT

La procedura WalkSAT presentata in [**114**] prende l'idea di random walk e la inserisce come componente centrale dell'algoritmo. Essenzialmente, il WalkSAT aggiunge i riavvii ed un'euristica per scegliere tra le variabili non soddisfatte alla procedura di decisione di Papadmitriou per 2SAT.

WalkSAT sceglie a caso una clausola non soddisfatta, e poi da questa sceglie una variabile non soddisfatta per modificarla usando un'euristica greedy oppure una random. La scelta esatta è come segue: se una variabile può essere modificata per soddisfare la clausola senza spezzare nessun'altra clausola, allora la variabile è modificata, altrimenti con la probabilità p modifichiamo la variabile che spezza il minor numero di clausole, oppure modifichiamo una variabile a caso. Notiamo che, diversamente dal GSAT, la funzione risultato non considera il numero di clausole soddisfatte, ma solamente il numero di clausole che attualmente sono soddisfatte e diventeranno non soddisfatte.

```
procedure WalkSAT(Σ)
    for i := 1 to Max-tries
        T := random(Σ); random truth assignment
        for j := 1 to Max-flips
            if T satisfies Σ then return T
            else pick an unsatisfied clause
                v := var in clause chosen either greedly or at random
                T := T with truth assignment to v flipped
        end
    end
    return "no satisfying assignment found"
```

Figura 3.7: *La procedura di ricerca locale WalkSat*

3.8 Novelty

Forse ispirato dalle buone performance del WalkSAT e HSAT, l'algoritmo Novelty [**93**] combina insieme una focalizzazione sulle clausole non soddisfatte con un meccanismo simile al HSAT per variabili modificate che non sono state modificate recentemente. L'algoritmo Novelty ritorna alla funzione di punteggio del GSAT anche il numero delle clausole soddisfatte. L'algoritmo Novelty sceglie in modo random una clausola non soddisfatta. Poi modifica la variabile che presenta il punteggio più alto e che non è l'ultima variabile per essere modificata nella clausola. Se c'è, poi con probabilità p, Novelty modifica la variabile col prossimo punteggio più alto, e con probabilità $1\text{-}p$, la variabile con il punteggio maggiore. L'algoritmo R-Novelty, introdotto sempre in [**93**], tiene conto della differenza tra i due risultati maggiori. Questo fa la procedura interamente deterministica per $p \in \{0,\ 0.5,\ 1\}$. Per inibire i loop, la variabile non soddisfatta è selezionata in modo random ogni 100 cambiamenti.

3.9 Simulated annealing

Simulated annealing non si dimostra così popolare come i metodi di ricerca locale, per esempio il GSAT. I confronti tra i metodi di ricerca locale come il GSAT ed il simulated annealing dipingono un quadro piuttosto confuso. Per esempio, Selman e Kautz non sono riusciti trovare uno schedule d'annealing che funzioni meglio del GSAT [**113**].

D'altra parte, Spears ha presentato dei risultati che suggeriscono che simulated annealing funzioni meglio del GSAT sui problemi 3SAT random difficili [**121**]. Ad oggi non abbiamo nessuna notizia su qualche confronto tra il simulated annealing ed i metodi di ricerca locale come il WalkSAT ed il Novelty, che tendono a funzionare meglio del GSAT su queste classi di problemi.

3.10 Ricerca tabu

Mazure, Sais e Gregoire hanno mostrato che la ricerca tabu funziona bene sui problemi SAT. Loro hanno proposto il TSAT, un algoritmo di ricerca tabu per il SAT [**92**]. Questo algoritmo tiene una lista FIFO di lunghezza fissa delle variabili modificate. Sui problemi

*3*SAT random difficili, hanno trovato che la lunghezza ottimale di questa lista tabu aumenta linearmente con N. Hanno mostrato che con tale lista il TSAT era molto competitiva con il WalkSAT.

3.11 Metodi ibridi

C'è stato qualche interesse sui metodi ibridi che combinano insieme le migliori caratteristiche dei metodi approssimati e quelli completi per il SAT. In metodi di ricerca locale possono attraversare rapidamente lo spazio di ricerca. D'altra parte, i metodi completi tendono a seguire traiettorie molto più limitate. Per questo, loro sono compensati da regole propagazione di vincoli come l'unit propagation. Perciò, Zhang e Zhang hanno proposto un metodo ibrido per il SAT che combina insieme un metodo di ricerca locale basato su GSAT ed una procedura sistematica basata sulla procedura di Davis-Putnam [**133**]. Loro riportano alcuni promettenti risultati su certi problemi di esistenza di quasigroup.

3.12 Altri approcci

Per risolvere i problemi SAT sono state utilizzate anche le reti neurali e gli algoritmi genetici. Un'eccezione è presentata in [**120**] dove Spears propone un algoritmo di rete Hopfield per risolvere i problemi SAT. La rete è basata su alberi di parsing AND/OR del problema SAT. Spears riporta dei risultati favorevoli sui problemi *3*SAT random difficili confrontati con il GSAT. Spears e de Jong hanno proposto anche un semplice algoritmo genetico per risolvere i problemi SAT [**72**].

Comunque, sono stati riportati dei risultati insufficienti per determinare se quest'approccio è competitivo con i metodi di ricerca locale come il GSAT.

Hogg e Williams hanno proposto algoritmi quantum per risolvere i problemi SAT. Per esempio, Hogg ha proposto un algoritmo quantum semplice per la soddisfacibilità ed il CSP [**60**]. Lui ha osservato una phase transition sui problemi *3*SAT random come vista nei metodi computazionali classici.

Finalmente, Lipton ha mostrato problemi SAT che utilizzano metodi di calcolo basato su DNA [**90**].

4 Comportamento di phase transition

Negli ultimi anni c'è stato un interessamento considerevole nella *phase transition* in tanti problemi *NP-completi* [**11**]. I problemi che sono molto sopravincolati sono irrisolvibili e di solito è facile determinare questo. Problemi che sono molto sottovincolati sono risolvibili e di solito è facile indovinare una delle tante soluzioni. Una phase transition tende a presentarsi in mezzo quando i problemi sono *criticamente vincolati* ed è difficile determinare se sono o no risolvibili. Tale comportamento è stato studiato intensivamente nel SAT negli ultimi cinque anni.

4.1 Phase transition nel kSAT random

Per il *2*SAT random, si è dimostrato che la phase transition si presenta a $L/N=1$ [**12**], [**53**]. Per il *3*SAT random, è stato dimostrato sperimentalmente che la phase transition si presenta intorno a $L/M=4.3$ [**15**], [**96**]. I limiti teoretici hanno messo la phase transition per *3*SAT random nell'intervallo $3.003 < L/N < 4.598$ [**28**], [**84**]. Per il *4*SAT random, è stato dimostrato sperimentalmente che la phase transition si presenta per $L/N = 9.8$ [**40**].

Per *k* grande, la phase transition per il *k*SAT random si presenta per il valore di L/N vicino a $-1/log_2(1-1/2^k)$ [**82**]. Un recente risultato di Friedgut, dimostra che l'ampiezza della phase transition nel modello *3*SAT random si restringe all'aumento della dimensione dei problemi [**25**]. Kamath e al. hanno dimostrato che alla transizione del *3*SAT random, la distribuzione nel numero di assegnamenti soddisfacenti è molto distorsionato [**73**]. Un numero esponenzialmente piccolo di problemi ha un numero esponenzialmente grande di assegnamenti soddisfacenti. Questo rende difficile il calcolo della posizione della transizione.

Una tecnica prestata dalla meccanica statistica chiamata rappresentazione in scala di dimensione limitata [**1**] modella la forma della transizione della soddisfacibilità [**82**], [**115**]. Intorno al valore critico di L/N, la misurazione in scala di dimensione limitata predice che proprietà macroscopiche come la probabilità di soddisfacibilità sono quasi sempre indistinguibili.

La misurazione in scala dei costi della ricerca può essere accuratamente modellata utilizzando questa tecnica [**33**].

4.2 Il costo della ricerca kSAT random

Per i metodi completi come quello di Davis-Putnam, è stato osservato un massimo del costo medio della ricerca, nella phase transition del *3*SAT random dove approssimativamente 50% di problemi sono soddisfacibili [**96**]. Crawford e Auton hanno riportato una semplice crescita esponenziale, per la loro procedura di Davis-Putnam, nella classe di problemi *3*SAT random a $l/n = 4.2$ (nella phase transition) e a $l/n = 10$ (nella regione sopravincolata) [**16**]. Loro hanno registrato anche una crescita approssimativamente lineare per $l/n = 1, 2$ e 3 nella regione sottovincolata [**16**]. Gent e Walsh hanno riportato un comportamento simile nel modello a probabilità costante [**39**], [**40**].

L'analisi teoretica della procedura di Davis-Putnam è stata ristretta alla classe più facile di problemi a probabilità costante. Yugami ha sviluppato un modello teoretico approssimativo per il caso medio di complessità della procedura Davis-Putnam attraverso la phase transition del *3*SAT random [**127**]. Mentre i risultati confermano gli esperimenti, la derivazione è complessa e produce solamente equazioni ricorsive.

Per i metodi di ricerca locale, Gent e Walsh hanno riportato una possibile crescita cubica nel costo di ricerca per GSAT nella phase transition per il *3*SAT random [**36**]. Parkes e Walser anche hanno registrato una crescita sotto esponenziale per varie versioni della

procedura GSAT nella phase transition del *3*SAT random [**105**]. Gent e al. hanno esteso questi risultati al costo del modello di ricerca attraverso l'intera phase transition del *3*SAT random [**34**]. Loro non sono stati capaci di mettere d'accordo i loro dati per il GSAT ad un modello di ricerca a crescita esponenziale, invece hanno utilizzato un semplice modello di legge potenza con due parametri che cresce non più veloce di n^4.

Recentemente, ci sono stati analisi più dettagliate della distribuzione del costo di ricerca. Frost e Rish hanno proposto le distribuzioni modelling discrete run-time con funzioni di probabilità continue [**29**]. Loro hanno mostrato che sui problemi *3*SAT random fortemente non soddisfacibili, la distribuzione run-time della procedura Davis-Putnam può essere modellata da una funzione con densità lognormale. Anche Hoos e Stutzle hanno modellato le distribuzioni run-time, concentrandosi sulle singole istanze di problemi per ridurre le discordanze [**66**], [**67**]. Loro hanno mostrato che le distribuzioni di run-time per algoritmi come il WalkSAT possono essere modellate spesso con una distribuzione esponenziale.

Queste confermano i risultati sperimentali [**41**] in cui si suggerisce che riavvii random offrono poco o nessun vantaggio con gli algoritmi che compiono random walk, così come gli algoritmi che tendono di non bloccarsi in un minimo locale.

4.3 Il modello a probabilità costante

Sono stati mostrati vari risultati teoretici per il modello a probabilità costante senza la cancellazione delle clausole unit o vuote. Per esempio, Purdom e Brown [**108**] mostrano che il tempo medio di funzionamento è polinomiale se qualsiasi del seguente è asintoticamente:

- $L \leq MlnN$ per qualche M costante;
- $L \geq e^{\varepsilon N}$ per qualche ε costante;
- $p \geq \varepsilon$;
- $p \leq Mln(N/N)^{3/2}$.

Comunque, è importante notare che questi risultati non coprono tutte le funzioni possibili di p e L, e non nella copertura limite della phase transition tra soddisfacibile e non soddisfacibile.

La maggior parte degli studi sperimentali ha utilizzato il modello CP, in cui le clausole unit e vuote sono cancellate per prevenire i problemi banalmente irrisolvibili. Se la lunghezza attesa della clausola è tenuta costante variando poi p come $1/n$, allora la solubilità della phase transition è approssimativamente uguale al valore costante L/N [**40**]. Mentre i problemi dal modello CP sono tipicamente molto più facili dei problemi *k*SAT random della stessa dimensione [**96**], i problemi occasionali nella regione risolvibile $easy$ possono essere molto più difficili per la procedura Davis-Putnam dei problemi più difficili della regione $hard$ del modello *k*SAT random [**39**], [**40**]. Tali problemi avvengono in una regione dove constraint propagation è meno efficace, la profondità della ricerca è massima e l'euristica di ramificazione può commettere errori occasionali molto costosi [**42**].

5 Applicazioni

Se un problema è *NP-completo*, possiamo in teoria tradurlo nel SAT usando una codifica che è polinomialmente limitata in dimensione. Le ricerche negli ultimi dieci anni hanno mostrato che quest'approccio è spesso notevolmente efficace in pratica.

5.1 Graph Coloring

I problemi di Graph Coloring possono essere codificati facilmente in SAT. Per codificare il problema di $k$coloring del grafo con n nodi, utilizziamo kn variabili, con v_{ij} vero se il nodo i prende il colore j. È possibile una codifica più compatta che usi $n\log_2 k$ variabili in cui v_{ij} è $True$ se il nodo i ha il colore con il bit j impostato su $True$. In più, possiamo scegliere di non specificare i vincoli che ogni nodo prende solamente un colore.

Se ad un nodo sono dati due o più colori, possiamo semplicemente scegliere uno a caso. Questo è specialmente utile per i metodi di ricerca locale. Anche Selman e al. hanno riportato che GSAT è competitivo con procedure specializzate di Graph Coloring [**116**]. Un'eccezione è Hoos, che ha guardato all'effetto delle diverse codifiche di problemi Graph Coloring nel SAT, su algoritmi di ricerca locale come WalkSAT [**66**]. Un'altra eccezione è la soluzione dei problemi d'esistenza dei quasigroup, che possono essere visti come un tipo specializzato di problemi Graph Coloring.

5.2 Hardware

Problemi come la verifica e la diagnosi dei difetti nell'hardware sono stati obiettivi popolari per la loro codifica nel SAT. Tali problemi sono rappresentati molto spesso nel SAT in modo molto diretto e naturale, specialmente se il modello di circuito ignora i problemi di tempi.

Il segnale in un circuito può essere modellato come variabile booleana che è $True$ se il segnale è alto e $False$ altrimenti. Le porte logiche come NAND sono poi modellate dalla congiunzione logica con lo stesso nome. Per esempio, Larrabee ha generato alcuni problemi SAT basati su una varietà di difetti nei circuiti digitali sincroni incluso *single stuck-at* e *bridge-faults* [**86**]. Come un altro esempio, Kamath, Karmarkar, Ramakrishnan e Resende hanno proposto dei benchmark che codificano i problemi di sintesi di circuiti nel SAT [**75**]. Mentre questi problemi di sintesi di circuiti sono usati in un numero di studi sui metodi di ricerca locale ([**112**], [**114**], [**116**]), loro competono poco con il metodo completo Davis-Putnam.

5.3 Pianificazione

Kautz e Selman hanno avuto successo nel risolvere i problemi di pianificazione, codificandoli nel SAT e utilizzando, sia metodi di ricerca locale come GSAT, che procedure complete come quella di Davis-Putnam con le strategie di randomization e di riavvio [**54**], [**77**], [**78**]. In generale, la pianificazione è un problema più complesso del SAT. Senza restrizioni sugli operatori, la pianificazione è indecidibile. Per codificare i problemi di pianificazione in SAT, si devono fare restrizioni supplementari. La proposta di Kautz e Selman è stata di limitare la lunghezza del piano. Per conservare la completezza questo limite può essere aumentato iterativamente.

Un problema di pianificazione può essere rappresentato come un set di assiomi, per il quale qualsiasi assegnamento soddisfacente corrisponde ad un piano valido. Nel linguaggio di pianificazione ogni predicato è aumentato con un indice di extra time. Per esempio, il predicato $move(x,y,z,i)$ è $True$ se trasportiamo x da y a z all'istante di tempo i.

Kautz e Selman hanno creato a mano le loro codifiche originali con informazioni supplementari di dominio che sono non esprimibili nel linguaggio azione STRIPS. Ernst, Millstein e Weld hanno proposto il sistema di pianificazione MEDIC nel quale la codifica è automatica e si avvicina all'efficienza delle codifiche manuali di Kautz e Selman [**24**]. Kautz e Selman hanno investigato anche un modo diverso di codificare i problemi di pianificazione nel SAT, basato sulla rappresentazione della pianificazione, utilizzato nel rivoluzionario sistema Graphplan [**79**].

5.4 Scheduling

Crawford e Baker hanno codificato problemi di scheduling delle officine meccaniche in SAT [**17**]. Il problema di scheduling delle officine meccaniche consiste nello scheduling di un set di lavori sottoposti ad una serie di vincoli delle risorse, le date d'inizio e di scadenza, ed ordinare in sequenza i vincoli. Crawford e Baker hanno rifiutato una codifica troppo semplice di questo problema, in cui la variabile booleana $v_i t$ è $True$ se l'operazione i parte al momento t ed è dato uno spazio di ricerca più grande del necessario. Invece, loro propongono una codifica in cui la variabile booleana $v_i j$ è $True$ se l'operazione i parte prima dell'operazione j.

6 Walksat

Gli algoritmi di ricerca locale sono stati applicati con successo in molti problemi di ottimizzazione. Generalmente, gli algoritmi di ricerca locale trovano soluzioni buone ma non ottimali, e si credeva che tali algoritmi non potessero essere appropriati per la soddisfacibilità, dove l'obiettivo è di trovare un assegnamento che soddisfa tutte le clausole (se esiste tale assegnamento).

La ricerca locale ha dimostrato di essere efficace nel trovare assegnamenti completamente soddisfacenti per i problemi CNF [**3.4**], [**3.3**]. Questi metodi offrono prestazioni migliori rispetto ai più conosciuti algoritmi di ricerca sistematica su certe classi di problemi grandi di soddisfacibilità. Per esempio, GSAT, un algoritmo di ricerca locale a riavvio casuale, può risolvere 3CNF computazionalmente difficili generate random con 2000 variabili, mentre i più veloci algoritmi di ricerca sistematica non possono gestire istanze dalla stessa distribuzione con più di 400 variabile [**9**], [**22**].

L'algoritmo GSAT di base realizza una ricerca locale attraverso lo spazio degli assegnamenti dei valori di verità, cominciando con un assegnamento generato random, e poi ripetutamente modificando l'assegnamento di una variabile che minimizza il numero totale di clausole non soddisfatte. Come con alcuni problemi combinatoriali, i minimi locali nello spazio di ricerca sono problematici nelle applicazioni dei metodi di ricerca locale.

Un minimo locale è definito come uno stato il cui vicinato locale non include uno stato che è rigorosamente migliore. L'approccio standard nell'ottimizzazione combinatoriale nel terminare la ricerca quando è raggiunto un minimo locale [**101**] non funziona bene per la verifica della soddisfacibilità booleana, quanto tempo interessano solamente gli ottimi globali. In [**116**] è stato mostrato che continuando semplicemente a cercare facendo mosse laterali non migliorative, aumenta di molto la percentuale del successo dell'algoritmo[2].

La serie degli stati esplorati in sequenza attraverso mosse laterali nello spazio di ricerca sono chiamate *altopiani*. La ricerca attraverso gli altopiani domina spesso la ricerca GSAT. Per un'analisi dettagliata, si può vedere Gent e Walsh [**35**].

Il successo del GSAT è determinato dalla sua abilità di muoversi tra gli altopiani successivamente bassi. La ricerca fallisce se GSAT non può trovare modo di uscire dagli altopiani, sia perché tali transizioni dagli altopiani sono rare, sia perché sono inesistenti. Quando capita questo, si può semplicemente riavviare la ricerca con un nuovo assegnamento iniziale random.

Ci sono anche altri meccanismi per uscire dai minimi locali o altopiani che consistono nel fare occasionalmente mosse in salita. Da notare fra tali approcci l'utilizzo del simulated annealing [**82**], dove un parametro formale *temperatura*, controlla la probabilità che l'algoritmo di ricerca locale faccia una mossa in salita.

Nel 1993 Selman e Kautz [**112**] hanno proposto un altro meccanismo per introdurre questo tipo di mosse in salita. La strategia è basata sulla combinazione del *random walk* sulle variabili che appaiono nelle clausole non soddisfatte, con la ricerca locale greedy. La strategia può essere vista come un modo di introdurre del *noise* in una maniera molto focalizzata, perturbando solamente quelle variabili critiche alle clausole non soddisfatte rimanenti.

Selman e al. hanno presentato dati sperimentali dettagliati che comparano le strategie random walk, simulated annealing, random noise, e GSAT di base sulle formule random

[2] Minton ed al. (1990) hanno incontrato un fenomeno simile nella loro applicazione di ricerca locale nel risolvere problemi grandi di scheduling.

computazionalmente difficili. Per ottenere le migliori performance per ogni procedura hanno calibrato i valori dei vari parametri.

Hanno considerato particolarmente i problemi di pianificazione [**77**] e di sintesi dei circuiti [**73**], [**75**].

6.1 Ricerca locale per verificare la soddisfacibilità

GSAT [**116**] effettua una ricerca locale greedy per assegnamenti soddisfacenti di una serie di clausole proposizionali [**1.5.1**], [**1.5.2**], [**1.5.3**]. La procedura comincia con un assegnamento di verità casuale. Poi si modifica l'assegnamento della variabile che minimizza il numero totale di clausole non soddisfatte. Da notare che il maggior decremento può essere zero (mosse laterali) o negativo (mosse in salita). I cambiamenti sono ripetuti fino quando viene trovato un assegnamento soddisfacente o si arriva al numero massimo di cambiamenti preimpostati $MAX\text{-}FLIPS$. Questo processo, se necessario, è ripetuto fino a $MAX\text{-}TRIES$ volte.

In [**116**] è stato mostrato che GSAT migliora notevolmente in confronto alle procedure di ricerca backtracking, come la procedura Davis-Putnam, su varie classi di formule, incluso le formule difficili generate random e le codifiche SAT di problemi Graph Coloring [**69**].

Come menzionato sopra, i minimi locali nello spazio di ricerca dei problemi combinatoriali sono i principali ostacoli nell'applicazione dei metodi di ricerca locale. L'utilizzo da parte del GSAT delle mosse laterali non elimina completamente questo problema, perché l'algoritmo può ancora bloccarsi sugli altopiani [***altopiani***].

Perciò, è utile assumere dei meccanismi per uscire dai minimi locali o altopiani facendo mosse in salita (modifiche che aumentano il numero di clausole non soddisfatte). Vedremo in seguito due meccanismi per fare tali mosse[3].

6.2 Simulated annealing

Simulated annealing introduce le mosse in salita nella ricerca locale usando un modello $noise$ basato sulla meccanica statistica [**82**]. Selman e al. hanno utilizzato l'algoritmo definito in Johnson e al. [**77**]: inizio con un assegnamento di verità generato random. Si sceglie ripetutamente una variabile random, e si calcola δ, la modifica del numero di clausole non soddisfatte quando è modificata quella variabile. Se $\delta \leq 0$ (mosse discendenti o laterali), si fa la modifica. Altrimenti, si modifica la variabile con probabilità $e^{-\delta/T}$, dove T è un parametro formale chiamato $temperatura$. La temperatura potrebbe essere tenuta costante[4], oppure decrementata lentamente da una temperatura alta verso vicino zero secondo uno scheduling di raffreddamento. Spesso si utilizzano scheduling geometrici, nei quali la temperatura è diminuita ripetutamente moltiplicandola con un fattore costante < 1.

Dato uno scheduling di raffreddamento finito, simulated annealing non garantisce che trovi un ottimo globale, ovvero un assegnamento che soddisfa tutte le clausole. Perciò, Selman e al. nei loro esperimenti hanno utilizzato avvii random multipli, e calcolato il numero medio di riavvii R, necessari prima di trovare una soluzione.

L'algoritmo GSAT di base è molto simile all'annealing alla temperatura zero, ma differisce nel senso che GSAT impiega naturalmente riavvii e fa sempre una mossa in discesa se è

---

[3] se l'unica mossa possibile per il GSAT è ascendente, esso farà tale mossa, ma tale mossa *obbligata* in salita è molto rara, e non è efficace per uscire dai minimi locali.

[4] questa forma di annealing corrisponde all'algoritmo di Metropoli (Jerrum 1992).

disponibile. Il valore R per GSAT è semplicemente il numero medio di tentativi richiesti per trovare una soluzione.

6.3 Strategia random walk

Selman e Kautz [**112**] hanno presentato diverse estensioni della procedura GSAT di base. Una di queste estensioni combina la strategia di random walk con la ricerca locale greedy. Più precisamente, loro hanno proposto la seguente strategia random walk combinata:

- scegliere con probabilità p una variabile che si presenta in alcune clausole non soddisfatte e cambiare il suo assegnamento di verità.
- seguire con probabilità $1\text{-}p$ lo schema GSAT standard, facendo la migliore mossa locale possibile.

Notiamo che le mosse $walk$ possono essere in salita.

Una versione più semplice della strategia random walk consiste nel non restringere la selezione della variabile scelta random, alla serie di variabili che appaiono nelle clausole non soddisfatte. Selman e al. hanno chiamato questa modifica $random\ noise$.

Si nota che random walk differisce sia dal simulated annealing che dal random noise, nel senso che le mosse in salita random walk sono collegate da vicino alle clausole non soddisfatte.

6.4 Risultati sperimentali

Selman e al. hanno comparato gli algoritmi GSAT di base, simulated annealing, random walk, e random noise su una serie di test, che includono sia i problemi CNF generati in modo random, che le codifiche booleane di altri problemi combinatoriali. I risultati sono mostrati nelle Tabelle 6.1, 6.2 e 6.3. Per ogni strategia sono dati il tempo medio in secondi impiegato per trovare un assegnamento soddisfacente[5], il numero medio di cambiamenti (flips) richiesto e il numero medio di riavvii R necessario prima di trovare una soluzione.

Per ogni strategia, Selman e al. hanno utilizzato almeno 100 riavvii random (il parametro $MAX\text{-}TRIES$ impostato in GSAT) per ogni istanza di problema; se sono stati necessari più di 20 riavvii prima di trovare una soluzione, la procedura è stata riavviata 1000 volte. Un "*****" nelle tabelle, indica che non è stata trovata nessuna soluzione dopo più di 10 ore di funzionamento, o di aver riavviato più di 1000 volte la procedura.

I parametri per ogni metodo sono stati variati in un range di valori, e sono stati inclusi nelle tabelle solamente i risultati delle calibrazioni migliori. Selman e al. per il GSAT di base hanno variato anche i valori $MAX\text{-}FLIPS$ e $MAX\text{-}TRIES$. Per il GSAT con random walk, gli autori hanno variato anche la probabilità p con cui è fatta una mossa non greedy, ed in modo simile per GSAT con random noise.

In tutti gli esperimenti di Selman e al., il valore ottimale di p è stato trovato tra 0.5 e 0.6. Per il simulated annealing con temperatura costante, gli autori hanno variato la temperatura T da 5 a 0 in passi da 0.05 (per $T=5$ le mosse in salita sono accettate con probabilità maggiore a 0.8).

[5] Gli algoritmi sono stati implementati in C e girano su un SGI Challenge con un processore MIPS R4400 a 70 MHz.

Per le formule random, le performance migliori sono state trovate per $T = 0.2$. Le formule di pianificazione hanno richiesto una temperatura più alta $T = 0.5$, mentre la sintesi di circuiti booleani è stata risolta più rapidamente ad una temperatura bassa $T = 0.15$.

Gli autori hanno sperimentato anche con vari scheduling geometrici di raffreddamento, e non hanno trovato alcuno scheduling che sia meglio del miglior scheduling a temperatura costante.

Selman e al. non sono riusciti a migliorare significativamente il numero medio di riavvii necessario prima di trovare una soluzione con scheduling di raffreddamento molto lenti, nonostante l'effetto sul tempo d'esecuzione. Una spiegazione possibile per questo è che quasi tutto il lavoro per risolvere i problemi CNF consiste nel soddisfare poche delle ultime clausole non soddisfate. Questo corrisponde alla parte a bassa temperatura dello scheduling geometrico, dove la temperatura ha poche variazioni.

6.5 Formule random difficili

Le istanze random delle formule CNF sono spesso state utilizzate nella valutazione delle procedure di soddisfacibilità perché possono essere facilmente generate e manca qualsiasi struttura fondamentale *nascosta*, spesso presente nelle istanze fatte a mano. Sfortunatamente, a meno di grande cura nello specificare i parametri della distribuzione random, i problemi così creati possono essere banali da risolvere. Mitchell e al. [**98**] hanno mostrato come i problemi random computazionalmente difficili possono essere generati usando il modello con clausole di lunghezza fissa.

Consideriamo che N sia il numero di variabili, K il numero di letterali per ogni clausola, e L il numero di clausole. Ogni istanza è ottenuta generando L clausole casuali, ognuna contenendo K letterali. I K letterali sono generati selezionando in modo random K variabili, e ognuna delle variabili è negata con una probabilità di 50%. La difficoltà di queste formule dipende in modo critico dal rapporto tra N e L. Le formule più difficili sono intorno alla regione dove le formule generate random hanno una possibilità d'essere soddisfacibili di 50%. Per le formule 3CNF ($K=3$) gli esperimenti mostrano che questo è il caso per $L \approx 4.3N$. Per N più grande, il rapporto critico per il punto 50% converge a 4.25. Selman e al. hanno esaminato gli algoritmi intorno al punto 4.3 su formule con dimensioni comprese tra 100 a 2000 variabili; i risultati sono presentati nella Tabella 6.1.

method	#vars #clauses	100 200	100 500	100 700	300 600	300 800	300 1500	300 2000	500 5000
zloc	best	0.4	10.8	21.0	2.8	8.3	35.4	64.4	233.8
zjohn1	mean	2.3	14.9	28.2	4.7	10.6	45.3	74.1	268.8
zjohn2	mean	1.4	13.5	26.9	2.7	9.0	44.1	76.5	257.4
anneal	best	0	5.6	15.1	0.7	3.1	22.6	47.6	215.7
zsamd	best	0	3.7	13.4	0.5	1.2	10.6	34.0	174.6
	mean	0.3	5.1	14.7	2.4	4.3	15.3	39	182.8
GSAT+	best	0	2.8	12.9	0	0	7.6	31.8	161.2
	mean	0	2.9	12.9	0	0	8.1	34.9	163.6

Tabella 6.1: *Comparazione delle strategie noise su istanze 3CNF random difficili*

Per le formule più piccole (100 variabili), si osserva una piccola differenza nei run-times. Con l'aumento del numero delle variabili, la strategia random walk domina significativamente sugli altri approcci. Le strategie random noise e simulated annealing migliorano anche rispetto a GSAT di base, ma nessuno di questi metodi trova soluzioni per le tre formule più

grandi[6]. Le performance del GSAT con walk sono notevoli, specialmente considerando il fatto che i più veloci metodi correnti di ricerca sistematica non possono risolvere le istanze 3CNF random con più di 400 variabili [**22**].

Le colonne marcate con $flips$ danno il numero medio di cambiamenti richiesti per trovare un assegnamento. Nell'algoritmo simulated annealing un $flips$ è un cambio attuale dell'assegnamento di verità. Comparando il numero di cambiamenti richiesti dalle varie strategie, si arriva alla stessa conclusione sulla relativa efficienza dei metodi.

Finalmente, viene considerato il numero medio di riavvii R necessari prima di trovare una soluzione.

GSAT di base si è facilmente bloccato sugli altopiani, e richiede molti riavvii random, in particolare per le formule più grandi. D'altra parte, GSAT con walk garantisce in sostanza di trovare un assegnamento soddisfacente. Combinando random walk con mosse greedy sulle variabili nelle clausole non soddisfatte, permette di uscire quasi sempre dagli altopiani che hanno pochi o nessun stato da cui può essere fatta una mossa in discesa. Anche le altre due strategie danno un valore migliorato di R rispetto al GSAT di base.

6.6 Problemi di pianificazione

Selman e al. hanno considerato come un secondo esempio dell'efficacia delle varie strategie d'uscita, le codifiche dei problemi di pianificazione blocksworld [**77**]. L'analisi dei migliori assegnamenti trovati quando GSAT fallisce nel trovare un assegnamento soddisfacente, indica che le difficoltà sorgono dai minimi locali molto profondi. Per esempio, il problema di pianificazione conosciuto col nome di *Hanoi* corrisponde al puzzle familiare *torri di Hanoi*, che consiste nel muovere una pila di dischi tra tre pioli, non mettendo mai un disco più grande sopra uno più piccolo. Ci sono molti assegnamenti di verità che soddisfano quasi tutte le clausole che codificano questo problema, ma sono molto diversi dal corretto assegnamento soddisfacente; per esempio, un assegnamento simile può corrispondere a far slittare un disco fuori dal fondo della pila.

Dalla Tabella 6.2 si osserva che GSAT con random walk è nettamente superiore. Come in precedenza, GSAT di base non riesce a risolvere i problemi più grandi. GSAT con walk è approssimativamente 100 volte più veloce del simulated annealing sui due problemi più grandi, e più di 200 volte più veloce del random noise.

formula			GSAT									Simul. Ann.		
			basic			walk			noise					
id	vars	clauses	time	flips	R	time	flips	R	time	flips	R	time	flips	R
med.	273	2311	7.5	70652	125	0.4	3464	1.0	4.5	41325	1.1	4.5	12147	1.0
rev.	201	1382	3.7	41693	100	0.3	3026	1.0	2.7	29007	1.1	2.7	9758	1.0
hanoi	417	2559	*	*	*	39	334096	2.6	20017	16×10^6	100	3250	4.1×10^6	25
huge	937	14519	*	*	*	38	143956	1.1	9648	37×10^6	200	8302	4.4×10^6	13

Figura 6.2: *Comparazione delle strategie noise su codifiche SAT di problemi di pianificazione*

formula		Int.P.	GSAT									Simul. Ann.		
			basic			walk			noise					
id	vars	time	time	flips	$R.$	time	flips	$R.$	time	flips	$R.$	time	flips	$R.$
f16a1	1650	2039	58	709895	5	2	3371	1.1	375	1025454	6.7	12	98105	1.3
f16b1	1728	78	269	2870019	167	12	25529	1.0	1335	2872226	167	11	96612	1.4
f16c1	1580	758	2	12178	1.0	1	1545	1.0	5	14614	1.0	5	21222	1.0
f16d1	1230	1547	87	872219	7.1	3	5582	1.0	185	387491	1.0	4	25027	1.0
f16e1	1245	2156	1	2090	1.0	1	1468	1.0	1	3130	1.0	3	5867	1.0

Figura 6.3: *Comparazione delle strategie di noise sui problemi di sintesi di circuiti come studiati da Kamath*

6.7 Sintesi di circuiti

Kamath e al. [**73**] hanno sviluppato un set di codifiche SAT per i problemi di sintesi dei circuiti booleani per valutare una procedura di soddisfacibilità basata sulla programmazione di interi. Il compito da considerare era da ricavare un circuito logico partendo dal suo comportamento input-output. Selman et al. [**116**] hanno mostrato che GSAT di base è competitivo con il metodo di programmazione di interi. Nella Tabella 6.3 sono mostrati i risultati sperimentali di Selman e al. sulle cinque più difficili istanze considerate da Kamath. Dalla tabella si nota che le entrambe strategie random walk e simulated annealing migliorano in modo significativo in confronto a GSAT, con random walk che qualche volta funziona meglio di simulated annealing. Per fare dei paragoni, Selman e al. hanno incluso anche i tempi originali riportati da Kamath e al[7]. In questo caso, la strategia di random noise non porta ad un miglioramento sul GSAT di base. Infatti, il mixing nel random noise sembra peggiorare le performance del GSAT.

Gli esempi di Kamath e al. [**73**] sono stati ricavati dai circuiti booleani collegati in modo casuale. Quindi, anche se le codifiche di SAT contengono alcune strutture intricate di porte booleane, c'è ancora un aspetto casuale delle istanze del problema. In seguito, Kamath e al. [**75**] hanno generalizzato il loro approccio, per considerare i circuiti con uscite multiple. Utilizzando questa formulazione, si possono codificare i circuiti booleani utili nelle applicazioni pratiche. Alcuni esempi sono i circuiti del sommatore e del comparatore. Selman e al. hanno codificato il comportamento I/O di molti di questi circuiti e utilizzato GSAT con walk per risolverli.

La Tabella 6.4 mostra i risultati di Selman e al. $GSAT+w$ significa GSAT con walk, $p=0.5$. Il tipo di circuito e indicato nella tabella. Per esempio, ogni assegnamento soddisfacente per la formula $2bitadd\_11$ corrisponde ad un progetto per un sommatore a 2-bit che usa un PLA (Programmable Logic Array). Il suffisso 11 indica che il circuito è costretto ad utilizzare solamente 11 porte AND. Si vedrà dalla tabella che GSAT con walk può risolvere le istanze in tempi che vanno da meno di un secondo ad alcuni minuti.

formula			DP	GSAT+w	WSAT
id	vars	clauses	time	time	time
2bitadd_12	708	1702	*	0.081	0.013
2bitadd_11	649	1562	*	0.058	0.014
3bitadd_32	8704	32316	*	94.1	1.0
3bitadd_31	8432	31310	*	456.6	0.7
2bitcomp_12	300	730	23096	0.009	0.002
2bitcomp_5	125	310	1.4	0.009	0.001

Figura 6.4: *Confronto del metodo completo DP con le strategie di ricerca locale su problemi di sintesi di crcuiti (tempi in secondi)*

formula			DP	GSAT+w	WSAT
id	vars	clauses	time	time	time
ssa7552-038	1501	3575	7	129	2.3
ssa7552-158	1363	3034	*	90	2
ssa7552-159	1363	3032	*	14	0.8
ssa7552-160	1391	3126	*	18	1.5

Figura 6.5: *Confronto del DP con le strategie di ricerca locale su problemi di diagnosi di circuiti da Larabee (tempi in secondi)*

Selman e al. hanno incluso anche i tempi per la procedura Davis-Putnam, includendo anche una versione di questa procedura sviluppata da Crawford e Auton [**15**]. Al momento rispettivo questa procedura era uno dei metodi completi più veloci, ma è sorprendente vedere che risolve

[7] La procedura di soddisfacibilità di Kamath ed al. girava su una VAX 8700 con il codice scritto in FORTRAN e C.

solamente due degli esempi[8]. Con "*" è stato indicato che il metodo ha girato per 10 ore senza trovare un assegnamento. Le buone performance del GSAT con walk su questi problemi indicano che i metodi di ricerca locale possono lavorare bene su problemi strutturati che non contengono alcun componente casuale.

6.8 Diagnosi di circuiti

Larrabee [**86**] aveva proposto una traduzione del problema della generazione di campioni di test per i circuiti VLSI in un problema SAT. Selman e al. hanno comparato le performance del GSAT con walk e quello del DP su molte delle formule di Larrabee. I loro risultati sono mostrati nella Tabella 6.5. Si nota che il GSAT con walk gira nuovamente molto bene, specialmente se comparato alla ricerca sistematica DP. Questi risultati e quelli per la sintesi dei circuiti presentano particolare interesse perché riguardano le codifiche dei problemi con chiara applicazione pratica.

6.9 Modifiche della strategia random walk

Selman e al. hanno implementato un algoritmo che perfeziona la strategia random walk del GSAT con semplici ma significative modifiche. Questo algoritmo, chiamato WSAT (walksat), fa cambiamenti, scegliendo per prima in modo random una clausola che non è soddisfatta dal corrente assegnamento, e poi scegliendo (sia a caso che secondo un'euristica greedy) una variabile nella clausola da modificare. Così, mentre GSAT con walk può essere visto come aggiungere $walk$ ad un algoritmo greedy, WSAT può essere visto come aggiungere golosità come euristica al random walk. Sembra dipendere da particolari classi di problemi il fatto che WSAT migliori o no rispetto al GSAT con walk.

Un'inaspettata e interessante osservazione degli autori è che potrebbe essere una grande discrepanza tra il funzionamento del GSAT con 100% walk ($p=1.0$) e il funzionamento del WSAT, dove le variabili sono scelte a caso in una clausola non soddisfatta. Ad un primo sguardo, queste scelte sembrerebbero essere identiche. GSAT mantiene una lista (senza duplicati) delle variabili che appaiono nelle clausole non soddisfatte, e sceglie a caso da quella lista; così, ogni variabile che appare in una clausola non soddisfatta è scelta con uguale probabilità. WSAT utilizza un processo casuale a due passi come descritto sopra (scegliendo prima una clausola, e poi una variabile), che favorisce le variabili che appaiono in molte clausole non soddisfatte. Per molte classi di formule, la differenza sembra non essere significativa. Comunque, GSAT con 100% walk non risolve i problemi di diagnosi dei circuiti, mentre WSAT con scelta random può risolverli tutti.

6.10 Soddisfacibilità massima

Selman e al. hanno comparato le performance del GSAT con walk ai metodi studiati da Hansen e Jaumard [**57**] per MAX-SAT. I loro risultati appaiono nella Tabella 6.6.

Hansen e Jaumard hanno analizzato cinque algoritmi diversi per MAX-SAT, un algoritmo di ricerca locale base chiamato *zloc*, due algoritmi deterministici proposti da David Johnson [**69**] chiamato *zjohn1* e *zjohn2*, un approccio simulated annealing chiamato *anneal*, e il loro algoritmo "steepest ascent, mildest descent" "*zsamd*". L'ultimo è simile al GSAT di base con una forma di lista tabu [**50**].

Selman e al. hanno ripetuto i loro esperimenti che utilizzavano GSAT con walk. Le istanze dei problemi sono istanze 3SAT generate random. Per ogni dimensione del problema, sono

[8] esperimenti preliminari indicano che alcune di queste formule possono essere risolte anche combinando DP con avvii multipli che casualmente permutano le variabili. Dettagli appariranno nella versione intera.

stati generati in modo random 50 problemi, ed ogni problema è stato risolto 100 volte,

method	#vars	100	100	100	300	300	300	300	500
	#clauses	200	500	700	600	800	1500	2000	5000
zloc	best	0.4	10.8	21.0	2.8	8.3	35.4	64.4	233.8
zjohn1	mean	2.3	14.9	28.2	4.7	10.6	45.3	74.1	268.8
zjohn2	mean	1.4	13.5	26.9	2.7	9.0	44.1	76.5	257.4
anneal	best	0	5.6	15.1	0.7	3.1	22.6	47.6	215.7
zsamd	best	0	3.7	13.4	0.5	1.2	10.6	34.0	174.6
	mean	0.3	5.1	14.7	2.4	4.3	15.3	39	182.8
GSAT+	best	0	2.8	12.9	0	0	7.6	31.8	161.2
	mean	0	2.9	12.9	0	0	8.1	34.9	163.6

Figura 6.6: *Risultati sperimentali per MAX-3SAT. I dati per i primi cinque metodi sono da Hansen e Jaumard.*

utilizzando diversi assegnamenti iniziali random. Selman e al. hanno incluso nella tabella i valori dei numeri migliori e medi delle clausole non soddisfatte trovato durante le 100 prove. Per esempio, nelle istanze 3SAT con 500 variabile - 5000 clausole, i migliori assegnamenti trovati del GSAT con walk contenevano una media di 161.2 clausole non soddisfatte. Come possiamo notare dalla tabella, GSAT con walk ha trovato costantemente soluzioni migliori rispetto a qualsiasi altro metodo. Notiamo che c'è solamente una piccola differenza tra i valori migliori e quelli medi trovati dal GSAT con walk, indicazione che i valori migliori sono, infatti, vicini ad ottimo.

7 Relsat

Bayardo e Schrag hanno presentato una versione migliorata della procedura Davis-Putnam per la soddisfacibilità proposizionale SAT su esempi ricavati dai problemi del mondo reale nella pianificazione, scheduling, diagnosi e sintesi dei circuiti. Gli autori hanno mostrato che incorporando tecniche CSP look-back, specialmente la nuova tecnica dell'apprendimento relevance-bounded, trasformano molti problemi facili che altrimenti sarebbero oltre la portata del DP. Frequentemente loro fanno che l'algoritmo sistematico DP, lavori altrettanto bene o meglio degli algoritmi SAT stocastici come GSAT o WSAT.

Come gia mostrato, la soddisfacibilità proposizionale CNF SAT è uno specifico problema con soddisfacimento di vincoli CSP. Fino poco tempo fa ci sono state poche applicazioni delle tecniche CSP look-back negli algoritmi SAT. Nel lavoro precedente, Bayardo e Schrag [**4**] hanno dimostrato che la versione look-back-enhanced dell'algoritmo Tableau per le istanze *3*SAT [**16**] può risolvere facilmente molte istanze che senza il look-back sarebbero *eccezionalmente hard*, più difficili di altri esempi con le stesse caratteristiche di superficie.

Nella presentazione del Relsat, le istanze sono state generate artificialmente. Bayardo e Schrag hanno dimostrato l'utilità pratica delle tecniche CSP look-back utilizzando un nuovo algoritmo look-back-enhanced riferito a Tableau per risolvere istanze SAT grandi derivate da problemi di pianificazione del mondo reale, scheduling e diagnosi e sintesi dei circuiti. Kautz e Selman [**78**] avevano trovato Tableau inadeguato per risolvere molte istanze derivate dalla pianificazione e sono ricorsi ad utilizzare un algoritmo stocastico, WSAT anche noto come WalkSAT [**114**].

Data la solita struttura della ricerca backtrack per le soluzioni sistematiche del problema con soddisfacimento di vincoli CSP, le tecniche progettate per migliorare l'efficienza possono essere divise in due classi:

- tecniche di look-ahead che sfruttano informazioni riguardo lo spazio di ricerca rimanente;
- tecniche look-back che sfruttano le informazioni sulla ricerca che ha avuto già luogo.

La prima classe include l'euristica con ordinamento delle variabili e schemi *dynamic consistency enforcement* come il *forward checking*.

La seconda classe include schemi per il backjumping (noto anche come *intelligent backtracking*) ed apprendimento (noto anche come *nogood* o *constraint recording*).

Negli algoritmi CSP, sono popolare tecniche dalle entrambe classi; per esempio, una combinazione comune di tecniche è forward checking, conflict-directed backjumping, e un ordening heuristic preferring variables con i domini minori.

SAT è un genere specifico di CSP, in cui ogni variabile prende i valori *{True, False}*. Gli algoritmi sistematici più popolari per il SAT sono versioni della procedura Davis-Putnam [**19**]. In termini di CSP, DP è equivalente alla ricerca backtrack con forward checking e un ordering heuristic favoring unit-domained variables. Due moderne realizzazioni efficaci sono Tableau [**16**] e POSIT [**26**]; le entrambe sono estremamente ottimizzate ed includono euristiche con ordinamento di variabili selezionate attentamente.

Gli algoritmi di ricerca sistematica o globale attraversano lo spazio di ricerca sistematicamente per assicurarsi che nessuna parte rimanga inesplorata. Loro sono *completi*: avendo a disposizione abbastanza run-time, se esiste una soluzione loro la troveranno; se non esiste nessuna soluzione loro riporterà questo.

L'alternativa agli algoritmi sistematici per SAT sono gli algoritmi stocastici o algoritmi di ricerca locale come WSAT e GSAT [**116**], come abbiamo visto precedentemente. Gli algoritmi stocastici esplorano lo spazio di ricerca in modo casuale, facendo perturbazioni locali ad un assegnamento di lavoro senza memoria dove loro sono stati. Loro sono *incompleti*: non garantiscono di trovare una soluzione se esiste una; se non trovano nessuna soluzione, non possono concludere che non esiste nessuna.

Gli algoritmi stocastici migliorano rispetto a quelli sistematici su istanze soddisfacibili dalla regione phase transition dello spazio dei problemi casuali, come il 3SAT random. Le istanze in questa regione sono mediamente più difficili per diversi algoritmi; loro cominciano ad essere utilizzati frequentemente come benchmark per le performance degli algoritmi SAT. Nello stesso tempo è riconosciuto che loro hanno strutture fondamentali molto diverse dalle istanze SAT.

Gli algoritmi stocastici migliorano anche rispetto agli algoritmi sistematici come Tableau su alcuni problemi del mondo reale. Molti problemi di pianificazione codificati in SAT descritti da Kautz e Selman [**78**] sono irrealizzabili per Tableau (10 ore), ma sono risolti facilmente dal WSAT (circa 10 minuti). La versione DP look-back-enhanced di Bayardo e Schrag è competitiva con WSAT nell'identificare piani fattibili utilizzando le stesse istanze. Inoltre, DP look-back-enhanced dimostra l'inesistenza dei piani corti da 1 a 3 minuti su istanze che Tableau non riesce a risolvere in 10 ore; questo compito è impossibile per WSAT, dovuto alla sua incompletezza.

7.1 Descrizione dell'algoritmo

Per un CNF dato, Bayardo e Schrag hanno rappresentato un assegnamento come un set di letterali ognuno dei quali è soddisfatto. Un *nogood* è un assegnamento parziale che non soddisferà il CNF dato. La clausola $(a \wedge b \wedge \neg c)$ codifica il nogood $\{\neg a, \neg b, c\}$. Chiameremo tale clausola codificata nogood un *reason*. La risoluzione è l'operazione che consiste nel combinare due clausole in ingresso menzionando un letterale dato e la sua negazione, ricavando una clausola implicita che menziona oltre questi tutti gli altri letterali.

La procedura Davis-Putnam DP è rappresentata sotto in pseudo-codice. SAT è un problema di decisione, sebbene frequentemente noi siamo interessati ad esibire anche un assegnamento di verità soddisfacente σ, che è vuoto partendo con l'entrata iniziale al livello superiore fino al ricorrente, procedura DP call-by-value.

```
procedure DP(Σ, σ)
    UNIT-PROPAGATE(Σ, σ)
    if () in Σ then return
    if Σ = ∅ then exit-with(σ)
    α ← SELECT-BRANCH-VARIABLE(Σ)
    DP(Σ ∪ {(α)}, σ ∪ {α})
    DP(Σ ∪ {(¬α)}, σ ∪ {¬α})
    return
```

La formula CNF Σ e l'assegnamento di verità σ sono modificati in chiamate per nome nell'UNIT-PROPAGATE. Se Σ contiene una contraddizione, questo è fallimento ed è necessario il backtracking. Se tutte le sue clausole sono già state semplificate, allora l'assegnamento corrente soddisfa la formula CNF.

SELECT-BRANCH-VARIABLE è una funzione euristica che ritorna la prossima variabile al valore nell'elaborazione dell'albero di ricerca. Se nessun valore di verità funziona, anche questo è fallimento.

UNIT-PROPAGATE(Σ, σ)
while (exits ω in Σ **where** $\omega=(\lambda)$)
$\sigma \leftarrow \sigma \cup \{\lambda\}$
$\Sigma \leftarrow$ SIMPLIFY(Σ)

UNIT-PROPAGATE aggiunge il singolo letterale λ da una clausola unità ω al set di letterali σ, poi semplifica la formula CNF rimovendo qualsiasi clausola nella quale si presenta λ, e accorciando tutte le clausole in cui $\neg\lambda$ si presenta attraverso la risoluzione.

Le versioni moderne del DP, incluso POSIT e Tableau, incorporano unit propagators altamente ottimizzati, ed euristiche sofisticate per selezionare le variabili di ramificazione. L'euristica per selezionare le variabili di ramificazione utilizzata dall'implementazione di Bayardo e Schrag è inspirata dall'euristica del POSIT e del Tableau, sebbene è piuttosto semplice ridurre il carico dell'implementazione.

Dettagli della selezione della variabile di ramificazione sono come segue.

Se non ci sono clausole binarie, selezionare una variabile di ramificazione in modo random. Altrimenti, assegnare ad ogni variabile γ apparsa in alcune clausole binarie un punteggio $neg(\gamma) \cdot pos(\gamma) + neg(\gamma) + pos(\gamma)$ dove $neg(\gamma)$ e $pos(\gamma)$ sono i numeri delle occorrenze di γ e $\neg\gamma$ in tutte le clausole binarie.

Raggruppare tutte le variabili all'interno del 20% del miglior punteggio in un set candidato.

Se ci sono più di 10 candidati, rimuovere le variabili a caso finché rimangono solamente 10. Se c'è solamente un candidato, è ritornato come variabile di ramificazione.

Altrimenti, ogni candidato è punteggiato di nuovo come segue. Per un candidato γ, calcolare $pos(\gamma)$ e $neg(\gamma)$ come il numero di variabili valutato da UNIT-PROPAGATE dopo aver fatto l'assegnamento $\{\gamma\}$ e rispettivamente $\{\neg\gamma\}$. Probabile che le entrambe unit propagation portino ad una contraddizione, ritornare immediatamente γ come la prossima variabile di ramificazione e seguire l'assegnamento per questa variabile che porta alla contraddizione. Altrimenti, accordare un punteggio a γ utilizzando la stessa funzione come sopra. Ogni candidato deve essere punteggiato senza trovare una contraddizione, selezionare a caso una variabile di ramificazione da quelle candidato all'interno del 10% dei migliori punteggi.

Eccetto i casi di contraddizione notati sopra, il primo valore di verità assegnato ad una variabile di ramificazione è selezionato a caso. Bayardo e Schrag hanno applicato le randomizzazioni descritte solamente dove l'euristica tradizionale non è riuscita migliorare sostanzialmente attraverso le varie istanze.

7.2 Incorporare CBJ e apprendimento

Il pseudo-codice del DP di sopra realizza un backtracking semplice mediato da una stack con una funzione ricorrente. Il conflict directed backjumping CBJ [**107**] ritorna attraverso questo stack astratto in una maniera non sequenziale, dove possibile, saltellando per motivi d'efficienza i frame di stack. Il suo meccanismo richiede l'esaminazione degli assegnamenti fatti da UNIT-PROPAGATE, non solo assegnamenti a variabili di ramificazione, quindi è più complicato di quello che potrebbe rappresentare lo pseudo-codice DP.

Bayardo e Schrag hanno implementato il CBJ avendo UNIT-PROPAGATE che mantiene un pointer alla clausola nell'ingresso CNF che serve come reason per escludere di considerare un particolare assegnamento.

Per esempio, quando $\{\neg a, \neg b\}$ è parte del assegnamento corrente, la clausola di ingresso $(a \vee b \vee x)$ è il reason per escludere l'assegnamento $\{\neg x\}$. Ogni qualvolta si ha una contraddizione, sappiamo che qualche variabile ha entrambi i valori di verità non ammessi. Per questo fallimento il CBJ costruisce un *working reason* C risolvendo le due rispettive reason; poi indietreggia alla variabile δ più recentemente assegnata in C. Supponiamo che $\{\gamma\}$ è stato l'assegnamento più recente della variabile γ. Se $\{\neg\gamma\}$ è esclusa dalla reason D, allora creiamo un nuovo working reason E risolvendo C e D e si indietreggia alla variabile più recentemente assegnata in E. Altrimenti, installiamo C come la reason per escludere $\{\gamma\}$, cambiare l'assegnamento corrente per includere $\{\neg\gamma\}$, e procediamo con la DP.

Estendendo il nostro esempio, supponiamo che nel trovare l'errore abbiamo la reason complementare $(a \vee b \vee \neg x)$, e che b è stato assegnato dopo a. La risoluzione ci dà il working reason $(a \vee b)$, così CBJ indietreggia dove è stato fatto l'assegnamento $\{\neg b\}$. Se $\{b\}$ è escluso, supponiamo che il reason è $\{\neg b \vee y\}$. La risoluzione fornisce il nuovo working reason $(a \vee y)$ e CBJ lo mantiene. Se $\{b\}$ non è escluso (b era un variabile di ramificazione), $(a \vee b)$ diventa la reason escludendo $\{\neg b\}$, e $\{b\}$ sostituisce $\{\neg b\}$ nell'assegnamento corrente prima che la DP continui.

Gli schemi con apprendimento mantengono derived reason più lunghe di quanto fa il CBJ, che può scartarli appena loro non indicano più un valore come escluso. L'apprendimento senza restrizioni memorizza ogni derived reason esattamente come se fosse una clausola dall'istanza fondamentale, permettendo che sia utilizzato per il resto della ricerca. In quanto l'overhead dell'apprendimento senza restrizioni è alto, Bayardo e Schrag hanno applicato solamente gli schemi con apprendimento limitato come definito in [**5**]. L'apprendimento size-bounded dell'ordine i mantiene indefinitamente solamente quelli derived reason che contengono i o meno variabili. Ad esempio, la reason $(a \vee b)$ sarebbe mantenuta dall'apprendimento size-bounded del secondo ordine, ma reason più lunghi non potrebbero. L'apprendimento relevance-bounded dell'ordine i mantiene ogni reason che contiene al massimo i variabili i cui l'assegnamenti sono stati cambiati da quando la reason è stata ricavata. Per esempio, supponiamo che stiamo effettuando l'apprendimento relevance-bounded del secondo ordine, e deduciamo una reason $(a \vee b \vee y)$ dove le variabili a, b e y sono state assegnate nell'ordine in cui appaiono. Questo reason sarebbe mantenuto dall'apprendimento relevance-bounded del secondo ordine finché a rimane assegnata come $\neg a$. Appena a è re-assegnata o dis-assegnata da un ripristino, la reason sarebbe scartata.

7.3 I test

Bayardo e Schrag hanno utilizzato tre serie di test separati per comparare le performance del DP look-back-enhanced con altri algoritmi le cui performance sono state riferite per gli stessi esempi: istanze di pianificazione codificate SAT da Kautz e Selman[9]; istanze di pianificazione e diagnosi di circuiti selezionate dal DIMACS Challenge associate con la competizione SAT del 1993[10] ed istanze di pianificazione, scheduling e sintesi di circuiti dalla competizione SAT di Beijing del 1996[11].

Le istanze di pianificazione codificate SAT costruite da Kautz e Selman [**78**] sono elencate nella Tabella 7.1.

---

9 Disponibile a ftp://ftp.research.att.com/dist/ai/logistics.tar.Z e ftp://ftp.research.att.com/dist/ai/satplan.dati.tar.Z

10 Disponibile a ftp://dimacs.rutgers.edu/pub/challenge/satisfiability

11 Disponibile a http://www.cirl.edu/crawford/beijing

Le istanze *log* corrispondono ai problemi di pianificazione nella logistica; le istanze *bw* sono per il blocsworld che sono tuttavia difficili. Le istanze *gp* sono codifiche Graphplan, le istanze *dir* codifiche dirette (state-based per le istanze di logistica, lineare per blocks world), e le istanze *un* sono codifiche Graphplan non soddisfabili per dimostrare la non fattibilità dei piani corti.

instance	vars	clauses	sat	type
log_gp.b	2,069	29,508	Y	planning
log_gp.c	2,809	48,920	Y	planning
log_dir.a	828	6,718	Y	planning
log_dir.b	843	7,301	Y	planning
log_dir.c	1,141	10,719	Y	planning
log_un.b	1,729	21,943	N	planning
log_un.c	2,353	37,121	N	planning
bw_dir.c	3,016	50,457	Y	planning
bw_dir.d	6,325	131,973	Y	planning

Tabella 7.1: *Le istanze di pianificazione di Kautz e Selman*

instance	vars	clauses	sat	type
ssa2670-141	986	2,315	N	diagnosis
bf1355-075	2,180	6,778	N	diagnosis
hanoi4	718	4,932	Y	planning
hanoi5	1931	14,468	Y	planning

Tabella 7.2: *Le istanze DIMACS*

instance	vars	clauses	sat	type
e0-10-by-5-1	19,500	108,887	Y	scheduling
e0-10-by-5-4	19,500	104,527	Y	scheduling
en-10-by-5-1	20,700	111,567	Y	scheduling
en-10-by-5-8	20,700	113,729	Y	scheduling
ew-10-by-5-1	21,800	118,607	Y	scheduling
ew-10-by-5-8	22,500	123,329	Y	scheduling
3blocks	283	9,690	Y	planning
4blocksb	410	24,758	Y	planning
4blocks	758	47,820	Y	planning
2bitadd_10	590	1,422	N	synthesis
2bitadd_11	649	1,562	Y	synthesis
2bitadd_12	708	1,702	Y	synthesis
2bitcomp_5	125	310	Y	synthesis
2bitmax_6	252	766	Y	synthesis
3bitadd_31	8,432	31,310	Y	synthesis
3bitadd_32	8,704	32,316	Y	synthesis

Tabella 7.3: *Le istanze Beijing*

Nella serie DIMACS, Bayardo e Schrag hanno scelto le istanze per la diagnosi dei circuiti di Van Gelder e Tsuji *ssa* (single-stuck-at) e *bf* (bridge-fault), e le istanze di pianificazione torre di Hanoi di Selman che utilizzano anche esse codifica lineare. Brevemente, riportiamo solamente le più difficili, per tutti gli algoritmi investigati delle istanze single-stuck-at e bridge-fault (mostrato nella Tabella 7.2).

Bayardo e Schrag hanno analizzato tutte le istanze dalla serie Beijing (vedi Tabella 7.3). Pure le istanze di pianificazione *blocks* utilizzano la codifica lineare. Le istanze di scheduling *e* codificano i benchmark di Sadeh come descritto in [**17**]. Le istanze di sintesi di circuiti *bit* sono il contributo di Bart Selman.

7.4 Metodologia sperimentale

Gli algoritmi di Bayardo e Schrag sono programmati in C++ utilizzando meno di 2000 linee incluso header files, linee bianche, e commenti[12]. L'implementazione è flessibile, con diverse tecniche look-back e livelli e vari parametri di compile-time e run-time.

Bayardo e Schrag non hanno realizzato un'implementazione ottimizzata. Le performance potrebbero essere migliorate anche utilizzando un'euristica di selezione della variabile di ramificazione più sofisticata come quella di Freeman e delle tecniche di pre-processing delle istanze.

Bayardo e Schrag hanno sperimentato con versioni diverse dell'algoritmo DP. La versione senza look-back-enhancement è chiamata *naivesat*, quella con CBJ soltanto *cbjsat*, quella a cui è applicato l'apprendimento relevance-size-bounded del ordine *i relsat(i)*, mentre quella a cui è applicato size-bounded di ordine *i sizesat(i)*. Bayardo e Schrag hanno utilizzato solamente apprendimento di ordine *3* e *4*, quanto tempo per l'apprendimento di ordine più alto risultava un overhead, mentre l'apprendimento di ordine più basso aveva poco effetto.

Si deve avere cura quando si sperimenta con le istanze del mondo reale perché il numero delle istanze disponibili per la sperimentazione è spesso limitato. L'esperimento in qualche modo deve permettere che i risultati delle performance in uno spazio delle istanze limitato di generalizzare ad altre istanze simili. Bayardo e Schrag hanno trovato la variazione del run-time degli algoritmi che risolvono la stessa istanza ad essere estremamente alta, dato che sembrano essere differenze insignificanti nella politica utilizzata nell'ordinare i valori e le variabili, sia che l'istanza è o no soddisfacibile.

Kautz e Selman [**78**] hanno calcolato la media del run-time su più cicli. Bayardo e Schrag hanno scelto lo stesso approccio per far girare più volte gli algoritmi (*100*) su ogni istanza con un numero casuale diverso designato ad ogni esecuzione per assicurare modelli d'esecuzione diversi. Allo scopo di trattare con esecuzioni che potrebbero prendere un tempo estremamente maggiore, è stato imposto un tempo limite di 10 minuti se non specificato diversamente, dopo il quale l'algoritmo usciva con un fallimento. Bayardo e Schrag hanno riportato la percentuale delle istanze con cui un algoritmo falliva nel risolvere entro il tempo limite. È stato riportato anche il tempo CPU medio richiesto per un'esecuzione e qualche volta gli assegnamenti medi delle variabili per un'esecuzione, facendo la media sulle esecuzioni riuscite.

Gli esperimenti sono stati realizzati su una workstations SPARC-10. Kautz e Selman [**78**] hanno riportato i tempi di esecuzione su una SGI Challenge 110-MHz. Per *normalizzare* i tempi di esecuzione con quelli di Kautz e Selman per le stesse istanze, Bayardo e Schrag hanno risolto una serie selezionata di istanze e hanno comparato la media dei *flips al secondo* riportata da WSAT, concludendo che la macchina di Kautz e Selman è stata 1.6 volte più veloce della loro SPARC-10[13]. Nei risultati sperimentali che seguono, Bayardo e Schrag hanno riportato tutti i tempi di esecuzione in secondi CPU *normalizzati SPARC-10*. Invece di normalizzare i tempi di esecuzione riportati da Kautz e Selman per Tableau *ntab*, Bayardo e Schrag hanno ripetuto gli esperimenti sulla loro macchina, utilizzando la nuova

[12] Source code disponibili a http://www.cs.utexas.edu/users/bayardo

[13] Bayardo e Schrag non son riusciti a ripetere facilmente gli esperimenti di Kautz e Selman sulla loro macchina, a causa della necessità di sintonizzare a mano i tanti parametri del WSAT.

versione disponibile del Tableau, ntab_back[14]. Questa versione di Tableau incorpora uno schema di backjumping, e quindi è più simile alla loro *cbjsat.* In quanto ntab_back non incorpora randomizzazioni, i tempi di esecuzione riportati per quest'algoritmo sono per una singola esecuzione per ogni istanza.

7.5 Risultati sperimentali

La Tabella 7.4 nostra le performance per relsat(4), WSAT, e ntab_back sulle istanze di pianificazione di Kautz e Selman.

instance	relsat(4)	% fail	WSAT	ntab_back
log_gp.b	12.9	0%	75.2	2,621
log_gp.c	39.4	0%	419.2	11,144
log_dir.a	4.1	0%	4.3	369.7
log_dir.b	16.6	0%	2.6	161.4
log_dir.c	90.3	22%	3.0	> 12 hours
log_un.b	66.8	0%	--	12,225
log_un.c	192.5	0%	--	> 12 hours
bw_dir.c	119	0%	1072	16.9
bw_dir.d	813.3	18%	1499	> 12 hours

Tabella 7.4: *Le performance di relsat(4) sulle istanze di pianificazione di Kautz e Selman.*

Il tempo limite è stato 10 minuti per ogni istanza eccetto bw_dir.d, quindi è stato di 30 minuti. I tempi per WSAT sono quelli riportati da Kautz e Selman [**78**], normalizzati a secondi CPU SPARC-10. Relsat(4) funziona meglio del WSAT sulla maggior parte delle istanze. Un'eccezione dove WSAT è chiaramente superiore è l'istanza log_dir.c che ha determinato il relsat(4) di raggiungere il tempo limite 22 volte. L'istanza bw_dir.d ha determinato e il relsat(4) di raggiungere il tempo limite 18 volte, ma gira ancora meglio del WSAT con vari minuti anche se Bayardo e Schrag hanno considerato 30 minuti per ogni tempo limite del relsat. Sebbene è difficile arrivare a conclusioni solide sulle performance del ntab_back quanto tempo il run-time riportato era solamente per una singola esecuzione, Bayardo e Schrag hanno deciso che relsat(4) è più efficace del ntab_back sulle istanze per le quali relsat(4) non ha mai raggiunto il limite, e ancora ntab_back richiede più di 10 minuti per risolvere. Questa include tutte le istanze log_gp e log_un.

algorithm	run-time	assgnmnts	% fail
naivesat	--	--	100%
cbjsat	115	999,555	0%
sizesat(3)	2.6	18,754	0%
sizesat(4)	.5	3,914	0%
relsat(3)	3.6	23,107	0%
relsat(4)	.6	4,391	0%

Tabella 7.5: *Le performance sull'istanza DIMACS bridge-fault bfl355-075*

Tabella 7.5 mostra le performance delle versioni DP di Bayardo e Schrag sull'istanza bfl355-075 del DIMACS, la più difficile delle istanze bridge-fault. Freeman [**26**] comunica che POSIT richiede 9.8 ore su una SPARC-10 per risolvere quest'istanza, mentre Bayardo e Schrag hanno trovato che ntab_back la risolve in 17.05 secondi. La Tabella 7.6 mostra le stesse informazioni per l'istanza ssa270-141 del DIMACS, la più difficile delle istanze single-stuck-at.

[14] Disponibile a http://www.cirl.uoregon.edu/crawford/ntab.tar

Freeman comunica che POSIT necessita *50* secondi per risolvere quest'istanza[15], e Bayardo e Schrag hanno trovato che ntab_back la risolve in *1.353* secondi. Le entrambe istanze sono non soddisfacibili.

algorithm	run-time	assgnmnts	% fail
naivesat	--	--	100%
cbjsat	415	9.4 Million	23%
sizesat(3)	242	5.0 Million	0%
sizesat(4)	278	4.7 Million	4%
relsat(3)	71	1.2 Million	0%
relsat(4)	46	.62 Million	0%

Tabella 7.6: *Le performance sull'istanza DIMACS single-stuck-at ssa270-141*

Naivesat non è stato capace di risolvere nessuna delle istanze entro *10* minuti in nessuna delle *100* esecuzioni. Aggiungendo CBJ ha risultato che l'istanza bridge-fault è stata risolta in tutte le *100* runs, ma l'istanza single-stuck-at ha provocato ancora *23* fallimenti. Tutti gli algoritmi di apprendimento hanno eseguito estremamente bene l'istanza bridge-fault. Per l'istanza single-stuck-at, l'apprendimento relevance-bounded e risultato ancora più veloce.

L'apprendimento size-bounded del quarto ordine, limitando la dimensione dello spazio di ricerca, funziona meno bene, dovuto al suo overhead alto, dell'apprendimento size-bounded del terzo ordine.

Le istanze DIMACS hanoi4 e hanoi5 sembrano contenere minimi locali molto profondi; anche se loro sono soddisfacibili, da quanto si sa, loro non sono stati risolti dagli algoritmi stocastici. Ntab_back risolve hanoi4 in *2,877* secondi ma è stato incapace risolvere hanoi5 entro *12* ore. I risultati delle versioni DP di Bayardo e Schrag su hanoi4 appaiono nella Tabella 7.7.

Sebbene sizesat(4) appare più veloce del relsat(3), il tempo medio di esecuzione è deviato dovuto al fatto che ha risolto con successo le istanze in soltanto *21%* delle esecuzioni. Relsat(3) ha avuto successo in quasi tutte le esecuzioni e relsat(4) in tutte eccetto una. Hanoi5 è stato risolto soltanto da relsat(4), e solamente in *4* su *100* tentativi. Il tempo medio di esecuzione in queste quattro esecuzioni riuscite è stato al di sotto dei tre minuti.

algorithm	run-time	assgnmnts	% fail
naivesat	--	--	100%
cbjsat	325	3.5 Million	94%
sizesat(3)	214	1.7 Million	92%
sizesat(4)	227	1.4 Million	79%
relsat(3)	254	1.6 Million	13%
relsat(4)	183	.89 Million	1%

Tabella 7.7: *Le performance sull'istanza di pianificazione hanoi4*

Le versioni DP di Bayardo e Schrag hanno lavorato relativamente bene nella maggior parte delle istanze Beijing. La tendenza generale è, con più look-back applicato, si ottengono migliori performance e un'abbassamento della probabilità di raggiungere il limite. Bayardo e Schrag hanno risolto tutte le istanze compresi in questa serie senza difficoltà utilizzando

[15] Freeman ha relazionato anche che POSIT esibisce alta variabilità di run-time su ssa2670-141, anche se la variazione non è quantificata.

relsat(4) con l'eccezione delle istanze del circuito *3bit* che non è stato mai risolto da alcuna delle loro versioni DP, invece queste istanze sono state banali per WSAT.

Le istanze di circuito *2bit* sono state banale (una frazione di secondo per trovare la soluzione) anche per cbjsat, con l'eccezione di 2bitadd_10, l'unica istanza non soddisfacibile. Quest'istanza non è stata risolvibile da nessuno degli algoritmi di Bayardo e Schrag entro *10* minuti. Dopo aver disabilitato il limite, relsat(4) ha stabilito che era non soddisfacibile dopo *18* ore.

Relsat(4) ha risolto *4* su *6* istanze di scheduling con *100%* percentuale di successo. Due delle istanze, e0-10-by-5-1 e en-10-by-5-1 hanno provocato percentuali di fallimento di *21%* e rispettivamente *18%*. Ripetendo gli esperimenti per queste due istanze con il limite di *30* minuti sono state ridotte le percentuali di fallimento a *3%* e rispettivamente *1%*. Crawford e Baker [**17**] hanno reso noto che ISAMP, un algoritmo randomizzato semplice, ha risolto questi tipi di istanze più efficacemente del WSAT o Tableau. L'implementazione ISAMP di Bayardo e Schrag ha risolto queste sei istanze con un ordine di grandezza più rapidamente del relsat(4), e con una percentuale di successo di *100%*, ma non è stato possibile di risolvere alcun altra istanza dalla serie di test presa in considerazione.

Dalle istanze di pianificazione di Beijing, relsat(3) e relsat(4) hanno trovato facile 3blocks, risolta con una percentuale di successo di *100%* mediamente in *6.0* e rispettivamente *6.4* secondi. Anche l'istanza 4blocksb è stata facile, sia relsat(3) che relsat(4) l'hanno risolta con *100%* successo, ma questa volta in *79* e rispettivamente *55* secondi. L'istanza 4blocks è stata più difficile. La percentuale di fallimento è stato di *34%* per relsat(3) e *17%* per relsat(4), con media di secondi CPU di *406* e rispettivamente *333* secondi.

8 Satz

Chu Min Li ed Anbulagan propongono una procedura DPL molto semplice chiamata Satz che assume solamente alcune tecniche look-ahead: un'euristica con ordinamento di variabili, un controllo di consistenza in avanti FCC (Unit propagation) e una risoluzione limitata prima della ricerca, dove l'euristica è se stessa basata su unit propagation. Satz è comparata sui problemi 3SAT random con tre procedure di DPL fra le migliori nella letteratura per questi tipi di problemi. Inoltre, su un grande numero di problemi in 4 ben conosciuti benchmark SAT, Satz raggiunge o addirittura supera le performance delle altre tre procedure DPL per i problemi SAT strutturati. I risultati comparativi suggeriscono che un utilizzo appropriato della tecnica look-ahead può permettere di fare a meno nella procedura DPL delle sofisticate tecniche look-back.

Consideriamo un set di variabili booleane $\{x_1, x_2, \ldots, x_n\}$. Per le definizioni del letterale, clausole, formule proposizionali e soddisfacibilità vedi [1.5.1], [1.5.2], [1.5.3]

Distinguiamo due tipi di problemi SAT:

- problemi che hanno strutture come regolarità, simmetrie;
- problemi random senza alcuna struttura.

Mentre i problemi del mondo reale sono spesso strutturati, i problemi random rappresentano il core del SAT e sono indipendenti da qualsiasi dominio particolare.

I più efficaci algoritmi sistematici sono basati sulla procedura ben conosciuta di Davis-Putnam nella forma di Loveland - procedura DPL [**19**]. Procedure DPL come CSAT [**22**], Tableau [**16**], e POSIT [**26**] di solito utilizzano un'euristica con ordinamento di variabili, e un FCC noto come tecnica look-ahead in termini di CSP. Questi algoritmi attualmente hanno in parte difficoltà nel risolvere problemi SAT strutturati. Recentemente molti autori hanno proposto di includere (e enfatizzare) tecnica look-back come backjumping (noto anche come backtracking intelligente o backtracking non cronologico) e apprendimento (noto anche come nogood) in una procedura DPL per affrontare i problemi SAT strutturati. GRASP [**117**] e relsat(4) [**6**] sono tali procedure DPL che utilizzano le tecniche look-ahead e look-back, che sono efficaci per i problemi SAT strutturati ma non sono efficaci per i problemi SAT random.

Chu Min Li and Anbulagan hanno proposto una procedura DPL molto semplice chiamata Satz che utilizza solamente la tecnica look-ahead e un semplice pre-processing della formula CNF per aggiungere delle soluzioni di lunghezza ≤ 3 nel database delle clausole. I risultati comparativi sperimentali di Satz con vari procedure DPL (CSAT, Tableau, POSIT, GRASP, relsat(4)) suggeriscono che l'utilizzo appropriato dell'unit propagation e il pre-processing possono essere efficaci sia per i problemi SAT random che per quelli molto strutturati.

Tutti gli esperimenti sono stati fatti su una workstation SUN Sparc-20 a 125 MHz.

8.1 Satz

La procedura DPL è schematizzata nella Figura 8.1.

Essenzialmente, la procedura DPL costruisce un albero di ricerca binaria attraverso lo spazio degli assegnamenti di verità possibile fin quando si trova un assegnamento di verità soddisfacente, oppure conclude che non esiste alcun assegnamento di questo tipo, ogni chiamata ricorsiva costituendo un nodo dell'albero.

```
procedure DPL(F)
        begin
        if F is empty, return satisfiable
        while F contains a pure literal, satisfy the literal and simplify F.
        F:=UnitPropagation(F); If F contains an empty clause, return unsatisfiable.

        /* branching rule */
        select a variable x in F according to a heuristic H
        if the calling of DPL(F∪{x}) returns satisfiable then return satisfiable
                else return the result of calling DPL(F∪{¬x}).
        end

procedure UnitPropagation(F)
        begin
        while there is no empty clause and a unit clause l exists in F
                assign a truth value to the variable contained in l to satisfy l and simplify F
                return F
        end
```

Figura 8.1: *La procedura DPL*

Attualmente, la più popolare euristica SAT è l'euristica di Mom che riguarda la ramificazione dopo la variabile avendo le massime occorrenze nelle clausole di dimensione minima [**16**], [**22**], [**26**], [**65**], [**68**], [**98**]. Intuitivamente, queste variabili permettono di utilizzare bene la potenza dell'unit propagation ed aumentare l'opportunità di raggiungere una clausola vuota.

Comunque l'euristica di Mom può anche non massimizzare l'efficacia dell'unit propagation, in quanto prende in considerazione solamente clausole di dimensione minima per pesare una variabile, anche se alcune estensioni provano a prendere in considerazione anche clausole più lunghe con pesi esponenzialmente piccoli (per esempio, cinque clausole ternarie sono contate come una clausola binaria).

Recentemente un'altra euristica basata sull'unit propagation (euristica UP), ha dimostrato utilità e permette di utilizzare ancora di più la potenza dell'unit propagation [**16**], [**26**], [**87**], [88].

Data un variabile x, l'euristica UP esamina x aggiungendo rispettivamente le clausole unità x e $\neg x$ a F, ed indipendentemente fa due unit propagation. L'euristica UP permette poi di prendere tutte le clausole che contengono una variabile, e le loro relazioni in accordo in un modo molto efficace per pesare la variabile. Come un effetto secondario, permette di scoprire i così chiamati $failed\ literals$ in F, che quando soddisfatti rendono falso F in una singola unit propagation. Comunque, siccome l'esaminazione di una variabile da parte di due unit propagation rappresenta consumo di tempo, è naturale provare a limitare le variabili da esaminare.

Il successo dell'euristica di Mom suggerisce che più è grande il numero delle occorrenze binarie di una variabile, maggiore è la sua probabilità di essere una buona variabile di ramificazione, implicando che si dovrebbe limitare l'euristica UP a quelle variabili che hanno un numero sufficiente di occorrenze binarie.

In [**88**], Chu Min Li e Anbulagan hanno studiato i comportamenti di diverse restrizioni dell'euristica UP sui problemi 3SAT random difficili. Loro hanno trovato che l'euristica UP è sostanzialmente migliore di quella di Mom persino nella sua forma pura, dove tutte le variabili libere sono esaminate in tutti i nodi. Inoltre, più variabili sono esaminate, l'albero di ricerca è più piccolo, confermando i vantaggi dell'euristica UP, ma troppe unit propagation rallentano l'esecuzione.

Basandosi su valutazioni sperimentali di vari alternative, Chu Min Li e Anbulagan hanno avanzato una restrizione dinamica dell'euristica UP, assicurando che almeno T variabili selezionate dall'euristica di Mom sono esaminate dalle unit propagation. L'euristica UP risultante è realizzata dal predicato unario $PROP_z$.

Definizione

> Sia *PROP* un predicato binario tale che *PROP(x,i)* è *True* se *x* si presenta sia positivo che negativo nelle clausole binarie e avendo almeno *i* occorrenze binarie in *F*, *T* è un numero intero, allora $PROP_z(x)$ è definito ad essere il primo dei tre predicati *PROP(x,4), PROP(x,3), True* di cui la semantica denotazionale contiene più di *T* variabili.

$PROP_z$ è ottimale per i problemi 3SAT random difficili. Come vedremo, è anche molto potente per quelli strutturati.

Satz è una procedura DPL con l'euristica UP $PROP_z$ con T che è empiricamente fissato a 10. Sia $diff(F_1,F_2)$ una funzione che da il numero di clausole di dimensione minima in F_1 ma non in F_2, la regola di ramificazione di Satz è mostrata nella Figura 8.2, dove l'equazione $H(x)$ è suggerita da Freeman [**26**] in POSIT.

```
for each free variable x such that PROP_z(x) is true do
        let F' and F'' be two copies of F
        begin
        F':=UnitPropagation(F'∪{x});
        F'':=UnitPropagation(F''∪{¬x});
        if both F' e F'' contains an empty clause then
                return F is unsatisfiable
        if F' contains an empty clause then
                x:=0;
                F=F';
        if neither F' e F'' contains an empty clause then
                let w(x) denote the weight of x
                w(x):= diff(F',F);
                w(¬x):= diff(F'',F);
        end

for each variable x do
        H(X):= w(x)* w(¬x)*1024+ w(¬x)+ w(x);
        Branching on the free variable x such that H(x) is the greatest.
```

Figura 8.2: *La regola di ramificazione di Satz*

Satz permette una realizzazione molto semplice e naturale, che corrisponde esattamente alla descrizione di sopra dell'algoritmo, eccetto il fatto che il backtracking non è ricorsivo e la sua implementazione iterativa è inspirata originalmente all'ULog [**49**], un interprete del linguaggio Prolog. Non c'è nessun'altra tecnica in Satz diversa dei due miglioramenti descritti nella sezione [**8.3**]. Per riprodurre le performance di Satz, nel programma si utilizzano solamente degli arrays invece di strutture complesse di dati. Satz utilizza liste collegate per alimentare le clausole, poi, prima della ricerca copia tutti i dati negli arrays.

8.2 Discussione sull'euristica UP

Hooker e Vinay [**65**] hanno studiato le ipotesi di soddisfazione e di semplificazione, che sono spesso utilizzate per motivare o spiegare la regola di ramificazione di una procedura DPL. A parità di condizioni, l'ipotesi di soddisfazione parte dal presupposto che una regola di

ramificazione funziona meglio quando crea sottoproblemi che sono più probabili ad essere soddisfacibili, mentre l'ipotesi di semplificazione parte dal presupposto che una regola di ramificazione funziona meglio quando crea sottoproblemi con meno clausole e che sono più corte. Una delle loro conclusioni è che l'ipotesi di semplificazione è migliore dell'ipotesi di soddisfazione.

Satz utilizza un'altra ipotesi chiamata ipotesi dei vincoli *constraint hypothesis* che parte dal presupposto che una regola di ramificazione funziona meglio quando crea sottoproblemi con più vincoli e più forti, così una contraddizione può essere trovata prima. Da quando le clausole di dimensione minima sono i vincoli più forti in una formula di CNF, $w(x)$ e $w(\neg x)$ nella Figura 8.2 rappresentano rispettivamente i nuovi vincoli dei due sottoproblemi generati se x è selezionato come la prossima variabile di ramificazione. Secondo constraint hypothesis, la procedura DPL dovrebbe ramificare dopo x tale che $w(x)$ e $w(\neg x)$ sono i maggiori. L'equazione di definizione di $H(x)$ e di collegamento $w(x)$ e $w(\neg x)$ in Figura 8.2 per favorire x tale che $w(x)$ e $w(\neg x)$ solo a grosso modo uguali e bilanciare i due sottoalberi è sperimentalmente meglio del seguente:

$$w(x) + w(\neg x) + \alpha * min(w(x), w(\neg x)) \qquad (*)$$

dove α è una costante.

Sembra che il CSAT utilizzi l'ipotesi di semplificazione e l'equazione (*) per definire $H(x)$, *mentre* POSIT prova combinare le ipotesi di semplificazione e dei vincoli, e Tableau utilizza le ipotesi dei vincoli e la stessa equazione come POSIT per definire $H(x)$.

CSAT [**22**] esamina una variabile vicina alle foglie di un albero di ricerca da due unit propagation (chiamato processing locale) per scoprire rapidamente letterali errati. Anche Pretolani ha utilizzato un approccio simile (chiamato metodo di potatura) basato su hypergraphs in H2R [**105**]. Ma il processing locale ed il metodo di potatura come sono presentati rispettivamente in [**22**] e [**105**] non contribuiscono all'euristica di ramificazione.

Troviamo il primo utilizzo valido dell'euristica UP in POSIT [**26**] e Tableau [**16**]. POSIT e Tableau hanno un'idea simile al CSAT, per determinare le variabili da esaminare dall'unit propagation nei nodi dell'albero di ricerca: x sarà esaminato dall'unit propagation se x è fra le più pesanti k variabili.

La differenza principale tra Satz confrontato con Tableau e POSIT, è che Satz non specifica un limite superiore k del numero di variabili che saranno esaminate dall'unit propagation in un nodo. Invece, Satz specifica un confine più basso T. Infatti, Satz esamina molte più variabili in un nodo.

Data la profondità di un nodo, Tabella 8.1 illustra il numero medio delle variabili libere ($\#free\_vars$) ed il numero medio delle variabili esaminate dal Satz nel nodo ($\#examined\_vars$), con la profondità della radice che è 0. Per una comparazione con CSAT, Tableau e POSIT sono dati anche i valori teoretici nel nodo di k_C (per CSAT), k_T (per Tableau) e k_P (per POSIT), rispettivamente secondo le definizioni di k in [**22**], [**16**], [**26**].

Dalla Tabella 8.1, è chiaro che Satz esamina in ogni nodo molte più variabili del CSAT, Tableau o POSIT. Vicino alla radice, Satz esamina tutte le variabili libere. Altrove Satz esamina un numero sufficiente di variabili.

8.3 Miglioramenti resovent-driven del Satz

Sempre sotto l'ipotesi dei vincoli, Li e Ambulagan hanno fatto dei miglioramenti resovent-driven in Satz. Il primo miglioramento è la pre-processing della formula di ingresso aggiungendo delle soluzioni di lunghezza ≤ 3, ispirato da [**3**].

depth	$\#free\_vars$	$\#examined\_vars$	k_C	k_T	k_P
1	298.24	298.24	0	263	265
2	296.52	296.52	0	227	230
3	294.92	293.89	0	193	198
4	292.44	292.21	0	141	149
5	288.60	282.04	0	61	72
6	285.36	252.14	0	0	10 or 3
7	281.68	192.82	0	0	10 or 3
8	277.54	125.13	0	0	10 or 3
9	273.17	71.51	0	0	10 or 3
10	268.76	40.65	0	0	10 or 3
11	264.55	26.81	0	0	10 or 3
12	260.53	21.55	0	0	10 or 3
13	256.79	19.80	0	0	10 or 3
14	253.28	19.24	0	0	10 or 3
15	249.96	19.16	0	0	10 or 3
16	246.77	19.28	0	0	10 or 3
17	243.68	19.57	0	0	10 or 3
18	240.68	19.97	0	0	10 or 3
19	237.73	20.46	0	0	10 or 3
20	234.82	20.97	0	0	10 or 3

Tabella 8.1: *Il numero medio di variabili esaminate da Satz per problemi 3SAT random con 300 variabili e 1275 clausole (sono stati risolti 500 problemi)*

Per esempio, se F contiene due clausole

$$x_1 \vee x_2 \vee x_3;\ \neg x_1 \vee x_2 \vee \neg x_4$$

allora aggiungiamo la clausola $x_2 \vee x_3 \vee \neg x_4$ ad F. Le nuove clausole a turno possono essere utilizzate per produrre altre soluzioni di lunghezza ≤ 3. Il processo è compiuto fino alla saturazione. In pratica, imponiamo due vincoli:

- una soluzione risultata da due clausole binarie dovrebbe essere unaria per essere aggiunta nel database della clausola,
- una soluzione risultata da una clausola binaria e una clausola ternaria dovrebbe essere binaria per essere aggiunta nel database della clausola.

La pre-processing senza i due vincoli è stabilito come una opzione.

Chiaramente, le soluzioni aggiunte diventano vincoli espliciti in F come le altre clausole. Inoltre, una variabile vincolata da un letterale e della sua forma negata ha una occorrenza di più per essere vantaggiata dalla regola di ramificazione, nell'esempio di sopra questo è il caso per x_2 che è vincolato da entrambi x_1 e $\neg\ x_1$.

Il pre-processing standard rende Satz approssimativamente 10% più veloce per i problemi 3SAT random difficili, e permette di risolvere istantaneamente la classe di problemi aim del benchmark DIMACS. Inoltre Satz risolve due volte più veloce i problemi della classe $dubois$. La pre-processing senza i due vincoli permette di risolvere istantaneamente tutti i problemi $dubois$.

Il secondo miglioramento consiste nel ponderare con più precisione le variabili nei nodi dove $PROP_z$ è $True$, per tutte le variabili, perché questi nodi sono spesso vicini alla radice dell'albero di ricerca e la loro variabile di ramificazione ha un effetto più importante per ridurre la dimensione dell'albero. Definiamo $w(x)$ come il numero di soluzioni, le nuove clausole binarie prodotte dovrebbero risultare in F' da un singolo passo di risoluzione. $w(\neg x)$ è definito in modo simile.

Riguardo alla Figura 8.2, dopo avere eseguito $F' = UnitPropagation(F' \cup \{x\})$, $w(x)$ è definito:

$$[f(\bar{l}) + f(\bar{l}')]_{cond} \qquad ,cond = l \vee l' \text{ è in F' ma non in F}$$

dove $f(\bar{l})$ è il numero di occorrenze pesate di $\bar{l}$ in F, un'occorrenza $\bar{l}$ in una clausola binaria essendo contata come 5 in clausole di lunghezza 4,... Il fattore esponenziale 5 è fissato empiricamente ed è migliore, nella sperimentazione di Li e Ambulagan, di 2 - il fattore utilizzato nella regola Jeroslow-Wang [**68**].

8.4 Risultati comparativi sperimentali sui problemi 3SAT difficili

Li e Ambulagan hanno comparato Satz con CSAT, Tableau, e POSIT, le altre tre procedure DPL fra le migliori nella letteratura per i problemi 3SAT random difficili. I problemi 3SAT sono generati utilizzando il metodo di Mitchel e al. [**98**] da 4 serie di n variabili e m clausole con il rapporto $m/n = 4.25$, n in passi da 50 tra 250 variabili e 400 variabili. I problemi 3SAT random generati con il rapporto $m/n = 4.25$ sono i più difficili da risolvere.

Li e Ambulagan hanno utilizzato un eseguibile del CSAT del luglio 1996. La versione di Tableau utilizzata è chiamata $3tab$ ed è la stessa utilizzata per la sperimentazione presentata in [**16**].

POSIT è stato compilato utilizzando il *make* fornito, su una workstation Sun SPARC-20 dal source chiamato *posit-1.0.tar.gz*[16]. Le Tabelle 8.2 e 8.3 mostrano che le performance delle 4 procedure DPL sui problemi 3SAT random difficili con $250, 300, 350$ e 400 variabili, dove *time* - run-time medio reale si ottiene dal comando unix */usr/bin/time* e *t_size* - la dimensione dell'albero di ricerca è riportata o calcolata dal numero di rami riportati dalla procedura DPL. Notiamo che nessuna delle procedure CSAT, POSIT, 3tab e Satz utilizza il backjumping.

	250 vars 159 problems		300 vars 170 problems		350 vars 144 problems		400 vars 51 problems	
System	*time*	*t_size*	*time*	*t_size*	*time*	*t_size*	*time*	*t_size*
C-SAT	5.3	4610.1	39	23980	240	123198	1813	747478
Tableau	5.3	4013.8	41	22531	272	123106	1836	616693
POSIT	4.9	6393.6	38	40262	270	246715	2362	1814814
Satz	3.8	3846.0	19	18083	106	90071	614	461991

Tabella 8.2: *Run-time medio (in secondi) e la dimensione media dell'albero di ricerca del C-SAT, Tableau, POSIT e Satz per un rapporto m/n=4.25 per problemi soddisfacibili*

	250 vars 141 problems		300 vars 130 problems		350 vars 106 problems		400 vars 49 problems	
System	*time*	*t_size*	*time*	*t_size*	*time*	*t_size*	*time*	*t_size*
C-SAT	17.0	16142.0	128	83235	882	481936	5905	2538072
Tableau	16.1	11736.4	128	69862	947	430323	7362	2469466
POSIT	10.8	14455.5	83	89959	665	609623	4872	3726644
Satz	9.6	9864.0	54	51998	335	288812	1825	1389700

Tabella 8.3: *Run-time medio (in secondi) e la dimensione media dell'albero di ricerca del C-SAT, Tableau, POSIT e Satz per un rapporto m/n=4.25 per problemi non soddisfacibili*

Le Tabelle 8.2 e 8.3 mostrano che sui problemi 3SAT random difficili, Satz è più veloce delle sopra citate versioni di CSAT, Tableau e POSIT, la dimensione dell'albero di ricerca per

[16] disponibile e via ftp a ftp.cis.upenn.edu in pub/freeman

il Satz è la più piccola, e il run-time e la dimensione dell'albero di ricerca del Satz crescono più lentamente. Tabella 8.4 nostra che il vantaggio di Satz comparato con le versioni citate di CSAT, Tableau e POSIT al rapporto $m/n=4.25$.

Ogni voce è calcolata dalle Tabelle 8.2 e 8.3 (media di tutti i problemi in un punto) utilizzando la seguente equazione:

$$gain=(value(system)/value(Satz)-1)*100\%$$

	250 vars 300 problems		300 vars 300 problems		350 vars 250 problems		400 vars 100 problems	
System	*time*	*t_size*	*time*	*t_size*	*time*	*t_size*	*time*	*t_size*
C-SAT	66%	50%	126%	51%	152%	58%	216%	77%
Tableau	60%	15%	132%	31%	175%	45%	276%	66%
POSIT	18%	53%	68%	89%	133%	130%	198%	200%

Tabella 8.4: *Il guadagno di Satz rispetto a C-SAT, Tableau, POSIT in termini di run-time medio (in secondi) e la dimensione media dell'albero di ricerca per un rapporto m/n=4.25 calcolato dalle Tabelle 8.2 e 8.3*

dove $value$ è la media reale del run-time o la media reale della dimensione del albero di ricerca e $system$ è CSAT, Tableau o POSIT. Dalla Tabella 8.4, risulta chiaro che il vantaggio del Satz cresce con la dimensione della formula in ingresso.

8.5 Risultati comparativi sperimentali dei problemi SAT strutturati

Compariamo Satz con le altre tre procedure DPL (POSIT, GRASP e relsat(4)) fra le migliore nella letteratura per i problemi SAT strutturati, dove GRASP e POSIT sono le due migliori in [**17**] comparate con CSAT, Tableau, H2R , SATO, TEGUS sui benchmark DIMACS e UCSC e relsat(4) è la procedura migliore esaminata in [**6**]. Li e Ambulagan hanno utilizzato 4 ben conosciuti benchmark per i problemi SAT strutturati:

- DIMACS[17];
- UCSC[18];
- i problemi Beijing[19] challenging;
- i problemi di pianificazione proposti da Kautz e Selman[20].

La versione del POSIT è la stessa dell'ultima sezione. Li e Ambulagan hanno utilizzato un eseguibile del sistema GRASP (la versione del maggio 1996) disponibile da Joao M. Silva. Il relsat(4) system V1.00 sono stati ricevuti da Roberto J. Bayardo Jr.

Seguendo Hooker & Vinay [**65**] e Silva & Sakallah [**117**], Li e Ambulagan hanno utilizzato un tempo di $cutoff$ (2 ore) come un sostituto per il run-time reale e suddiviso i benchmark DIMACS e UCSC in classi, per esempio la classe $aim100$ include tutti i problemi con il nome $aim100*$. In ogni classe, $\#M$ denota il numero totale di membri della classe, mentre $\#S$ il numero di problemi risolti efficacemente dalla procedura DPL corrispondente in meno di due ore. $Time$ rappresenta il tempo totale CPU in secondi impiegati per trattare tutti i membri

[17] disponibile a ftp://dimacs.rutgers.edu/pub/challenge/satisfiability

[18] disponibile a ftp://dimacs.rutgers.edu/pub/challenge/sat/contributed/UCSC

[19] disponibile a http://www.cirl.uoregon.edu/crawford/beijing

[20] disponibile a ftp://ftp.ricerca.att.com/dist/ai/logistics.tar.Z

di una classe - un problema che non può essere risolto in meno di 2 ore contribuisce con 7200 secondi al tempo totale. Li e Ambulagan non hanno incluso le classe F, G, PAR32 e Hanoi5 che nessuno degli algoritmi Satz, GRASP, POSIT e relsat(4) riesce a risolvere in meno di 2 ore. I problemi difficili di Beijing e i problemi di pianificazione sono elencati individualmente e un problema che non può essere risolto in meno di 2 ore è marcato da >7200. I risultati ottenuti sono mostrati nelle Tabelle 8.5, 8.6, 8.7 e 8.8.

Una prima osservazione è che Satz, GRASP, relsat(4) sono tutte significativamente migliori del POSIT su questi benchmark. In seguito Li e Anbulagan hanno analizzato soltanto le performance di Satz, GRASP e relsat(4).

Li e Ambulagan hanno trovato che Satz è comparabile con relsat(4) e GRASP su più problemi DIMACS e UCSC (un totale di 22 classi) eccetto la classe $pret$ dove non c'è nessuna soluzione di lunghezza ≤ 3 e la maggior parte delle variabili sono simmetriche così che l'euristica UP fallisce nel distinguerli. Satz è significativamente migliore del GRASP e relsat(4) in 3 classi ($hole$, $ii16$, $par16$) e per 12 classi $aim50$, $aim100$, $aim200$, bf, $dubois$, $ii8$, jnh, $par8$, $bf0432$, $ssa0432$, $ssa6288$ e $ssa7552$, ha delle performance equivalenti. Satz è anche molto efficace su più problemi ssa e bf, eccetto un piccolo numero fra loro (8 ssa su 102 e 3 bf su 223) dove per Satz aumenta il tempo totale per risolvere le corrispondenti classi.

Per esempio, mentre la maggior parte dei problemi della classe $bf1355$ possono essere risolti entro 3 secondi, il problema $bf1355243$ impiega 1395 secondi per essere risolto. Il problema più difficile per Satz nelle classi ssa e bf è $bf2670\text{-}244$ su cui impiega 3 ore e 47 minuti per essere risolto.

Problem Class	#M	Satz		GRASP		POSIT		relsat(4)	
		#S	Time	#S	Time	#S	Time	#S	Time
aim-50	24	24	13.8	24	0.5	24	0.3	24	8.1
aim-100	24	24	3.6	24	1.2	24	352	24	8.4
aim-200	24	24	4.3	24	8.4	13	82792	24	8.2
bf	4	4	25.6	4	5.8	2	14415	4	5.9
dubois	13	13	1.7[a]	13	6.0	8	50599	13	4.6
hanoi4	1	1	677	1	3910	1	42.8	1	38.0
hole	5	5	483	4	9838	5	429	5	3671
ii8	14	14	5.6	14	17.1	14	1.0	14	7.5
ii16	10	10	106	9	7515	8	14454	10	331
ii32	17	16	7624	17	6.0	16	7551	17	2158
jnh	50	50	7.0	50	10.8	50	0.3	50	19.4
par8	10	10	0.8	10	0.2	10	0.04	10	3.6
par16	10	10	251	10	11349	10	27.0	10	463
pret	8	4	29612	8	13.0	4	29156	8	5.6
ssa	8	8	1748	8	3.9	8	35.1	8	39.1

Tabella 8.5: *Run-time totale (in secondi) dei problemi DIMACS*

Nel banchmark Beijing, Satz risolve lo stesso numero di problemi come relsat(4) e uno in più del GRASP. Sui problemi di pianificazione di Kautz e Selman, Satz è anche comparabile con GRASP e relsat(4) eccetto 4 problemi su 31. Notiamo che Satz contiene un pre-processing della formula CNF d'ingresso per cancellare le clausole duplicate, le tautologie, e i letterali duplicati nelle clausole, che rappresentano un consumo di tempo per i problemi grandi come i problemi di pianificazione di Kautz e Selman o alcuni problemi Beijing difficili che contengono più di 100.000 clausole e non sono trascurabili specialmente quando la stessa ricerca impiega poco tempo.

8.6 Look-ahead contro look-back

In termini di CSP, Satz adopera essenzialmente qualche semplice tecnica look-ahead per raggiungere al più presto possibile ad un dead end: un'euristica di ordinamento delle variabili (euristica UP), un controllo di consistenza in avanti (Unit propagation) e un pre-processing semplice, l'euristica essendo se stessa basata sul controllo della consistenza in avanti. Comunque, Satz non include tecniche look-back come backjumping ed apprendimento, un'altra classe di tecniche per i problemi CSP.

Problem Class	#M	Satz		GRASP		POSIT		relsat(4)	
		#S	Time	#S	Time	#S	Time	#S	Time
bf0432	21	21	35.7	21	35.3	21	20.6	21	28.0
bf1355	149	149	3576	149	109	68	648849	149	133
bf2670	53	51	26445	53	50.7	53	1149	53	359
ssa0432	7	7	1.7	7	0.6	7	0.1	7	3.6
ssa2670	12	7	40062	12	35.4	12	1139	12	257
ssa6288	3	3	7.2	3	0.2	3	7.7	3	4.4
ssa7552	80	80	60.5	80	15.4	80	1722	76	43.6

Tabella 8.6: *l run-time totale(in secondi) dei problemi UCSC*

Problem	Type	#vars	#clauses	Satz	GRASP	POSIT	relsat(4)
2bitadd_10	synthesis	590	1422	> 7200	> 7200	> 7200	> 7200
2bitadd_11	synthesis	649	1562	201	6.6	0.3	0.5
2bitadd_12	synthesis	708	1702	0.4	6.0	0.05	0.5
2bitcomp_5	synthesis	125	310	0.03	0.03	0.01	0.5
2bitmax_6	synthesis	252	766	0.07	0.1	0.01	0.4
3bitadd_31	synthesis	8432	31310	> 7200	> 7200	> 7200	> 7200
3bitadd_32	synthesis	8704	32316	4512	> 7200	> 7200	> 7200
3blocks	planning	370	13732	2.0	28.6	1.8	2.9
4blocksb	planning	540	34199	8.2	473	49.3	6.0
4blocks	planning	900	59285	1542	> 7200	> 7200	296
e0ddr2-10-by-5-1	scheduling	19500	108887	215	143	> 7200	299
e0ddr2-10-by-5-4	scheduling	19500	104527	232	88.5	3508	30.8
enddr2-10-by-5-1	scheduling	20700	111567	> 7200	95.3	> 7200	33.2
enddr2-10-by-5-8	scheduling	21000	113729	229	89.6	> 7200	33.7
ewddr2-10-by-5-1	scheduling	21800	118607	339	101	283	33.4
ewddr2-10-by-5-8	scheduling	22500	123329	279	102	> 7200	41.7

Tabella 8.7: *I run-time (in secondi) dei problemi Beijing*

L'euristica del GRASP si può spiegare dall'ipotesi di soddisfazione, che intende soddisfare direttamente il numero più grande di clausole, mentre per il relsat(4) dall'ipotesi di semplificazione, che intende valutare il maggior numero di variabili quando ramifica. La loro euristica con ordinamento è più semplice di quella del Satz. Comunque loro sfruttano un backjumping sofisticato ed apprendimento per affrontare i problemi SAT strutturati. I risultati sperimentali presentati nell'ultima sezione suggeriscono che le buone performance del GRASP e relsat(4) su molti problemi SAT strutturati possono essere raggiunte semplicemente da una risoluzione limitata in cima all'albero di ricerca ed un'ottima euristica UP. L'ultimo approccio ha i vantaggi di essere più semplice e molto più efficace per i problemi 3SAT random.

8.6.1 Euristica contro backjumping

Un'euristica migliore per l'ordinamento delle variabili permette di evitare molte backtracking inutili. Siano x_{i1},x_{i2},...x_{ib},...x_{id} le traiettorie dall'origine al dead end in un albero di ricerca DPL, il backtracking cronologico standard retrocede verso x_{id-1}, mentre il backjumping potrebbe saltare a x_{ib} nel caso quale le variabili x_{ib+1},...x_{id-1} non

contribuiscono al conflitto scoperto in x_{id}, evitando in questo modo il tempo per esplorare una regione inutile dello spazio di ricerca. Comunque se guardiamo più attentamente al caso, troviamo che se il backjumping permette di evitare l'esplorazione inutile, è perché le variabili di ramificazione $x_{ib+1},...x_{id-1}$ non sono buone. Se l'euristica di ordinamento delle variabili sceglie x_{id} come variabile di ramificazione immediatamente dopo x_{ib} il backjumping potrebbe essere semplicemente backtracking cronologico.

Problem	sat	#vars	#clauses	Satz	GRASP	POSIT	relsat(4)
bw_large.a	Y	459	4675	0.4	0.3	0.04	0.6
bw_large.b	Y	1087	13772	1.4	3.1	0.5	1.5
bw_large.c	Y	3016	50457	7.0	225	3.4	95.1
bw_large.d	Y	6325	131973	2980	3915	?	418
f7hh.14	Y	4814	114132	1797	114	> 7200	80.8
f7hh.14.simple	Y	3269	61295	1580	59.8	> 7200	32.1
f7hh.15	Y	5315	140258	28.9	313	> 7200	253
f7hh.15.simple	Y	3759	75030	22.3	153	> 7200	96.7
f8h_10	N	2459	25290	1.6	13.6	0.6	2.6
f8h_10.simple	N	1415	14346	1.1	5.7	0.3	1.2
f8h_11	Y	2883	37388	3.0	23.9	371	3.8
f8h_11.simple	Y	1782	20895	2.2	11.4	26.7	2.9
facts7h.10	N	2218	22539	1.4	12.7	2.0	2.1
facts7h	Y	2595	32952	2.6	19.8	0.9	3.5
facts7hh.12	N	3814	68300	> 7200	784	> 7200	200
facts7hh.12.simple	N	2353	37121	> 7200	165	> 7200	66.6
facts7hh.13-simp	Y	4315	48072	12.1	46.3	> 7200	38.2
facts7hh.13-simp.simple	Y	2514	48072	9.7	36.5	1357	25.0
facts7hh.13	Y	4315	90646	15.1	79.3	> 7200	90.1
facts7hh.13.simple	Y	2809	48920	11.3	45.6	> 7200	19.5
facts7hha.12	N	2990	39618	> 7200	111	> 7200	105
facts7hha.12.simple	N	1729	21943	> 7200	58.3	> 7200	36.4
facts7hha.13	Y	3371	53824	10.0	27.6	2134	7.7
facts7hha.13.simple	Y	2069	29508	7.7	15.2	637	5.9
facts8.13	Y	3727	66735	6.5	49.9	141	14.4
facts8h.12	Y	3303	51223	5.7	27.4	1890	5.0
logistics.a	Y	828	6718	212	31.7	11.8	3.0
logistics.b	Y	843	7301	0.7	34.1	0.3	1.5
logistics.c	Y	1141	10719	6.6	116	> 7200	111
rocket_ext.a	Y	331	4446	0.2	2.6	0.2	0.8
rocket_ext.b	Y	351	2398	0.3	3.8	0.02	0.8

Tabella 8.8: *I run-time (in secondi) dei problemi di pianificazione di Kautz e Selman*

Chiaramente l'euristica UP permette a Satz di fare a meno del backjumping in maggior parte dei casi.

8.6.2 Apprendimento

Satz non include alcuna delle tecniche di apprendimento classiche applicate durante la ricerca. Ma Satz include un *apprendimento statico* standard che consiste in una risoluzione limitata prima della ricerca per aggiungere alcune soluzioni di lunghezza ≤ 3 nel database della clausola. Una ricerca completa di tutte le soluzioni di lunghezza ≤ 3 prima della ricerca permettono di risolvere istantaneamente tutti i problemi della classe *dubois*, mentre l'*apprendimento statico* standard rende Satz due volte più veloce. D'altra parte, molti problemi come *ssa** e *ii** potrebbero esplodere la memoria del computer per una ricerca completa di tutte le soluzioni di lunghezza ≤ 3.

Sembra promettente un'integrazione accurata con un modesto overhead dell'euristica UP ed un apprendimento dinamico come quello proposto in relsat(4).

8.7 SAT random a confronto con SAT strutturato

Sembra che le tecniche look-back, come descritte in GRASP e relsat(4), non siano appropriate per i problemi *3*SAT random. Per esempio, la Tabella 8.9 mostra il comportamento del relsat(4) e del GRASP sui problemi *3*SAT random difficili confrontati con Satz.

D'altra parte, le tecniche look-ahead come quelle implementate in Satz possono essere utilizzate per affrontare i problemi *3*SAT random e molti problemi SAT strutturati. Se una tecnica è potente per i problemi SAT random, probabilmente è potente per problemi SAT strutturati.

	200 vars 300 problems		250 vars 300 problems	
System	161 *sat*	139 *unsat*	159 *sat*	141 *unsat*
Satz	0.8	1.8	3.8	9.6
relsat(4)	10.4	35.3	129.9	571.7
GRASP	280.4	1693.8	—	—

Tabella 8.9: *Run-time medio (in secondi) di Satz, relsat(4) e GRASP per il rapporto m/n=4.25*

La strategia essenziale per scrivere una procedura DPL è di provare a raggiungere al più presto possibile ad un dead end e le euristiche sono probabilmente, i metodi più efficaci per realizzare la strategia. Comunque, se l'euristica fallisce nel lavorare bene, per esempio, per i problemi dove la maggior parte delle variabili sono simmetriche, si dovrebbero provare altri metodi, come l'apprendimento o simmetry detection. In questo senso, Satz può essere ancora migliorata nel futuro.

9 Novelty

9.1 Introduzione

Le performance di una procedura di ricerca locale stocastica dipendono dall'impostazione del parametro di *noise,* che determina la probabilità di uscire dai minimi locali facendo mosse non ottimali. Nel simulated annealing [**82**], [**23**], questa è la *temperatura*; nella ricerca tabu [**51**], [**52**], il *periodo di possesso* - la durata di tempo per il quale una variabile modificata è tabu; nel GSAT [**113**] - il parametro *random walk*, e nel WSAT anche chiamata walksat, [**114**] - il parametro è stato semplicemente chiamato *noise*. L'impostazione ottimale del parametro noise dipende dalle caratteristiche delle istanze del problema, e dai dettagli strutturali della procedura di ricerca che potrebbe essere influenzata da altri parametri. Si richiede uno sforzo considerevole per trovare il parametro noise ottimale che mette per una data distribuzione di problemi usando trial-and-error. Inoltre, qualche volta si affronta un problema unico, difficile da risolvere, e perciò non si può regolare il noise risolvendo altri problemi simili.

Così sarebbe desiderabile di trovare un modo per impostare il parametro noise che non varia con particolari algoritmi di ricerca o istanze di problemi particolari.

McAllester e al. hanno studiato sei variazioni dell'architettura del WSAT di base su una classe di istanze di problemi random difficili. Secondo questo studio hanno scoperto due *invariants*.

Primo, per una data classe di problemi, il livello di noise misurato dal *valore della funzione obiettivo* (il numero di clausole non soddisfatte) all'impostazione del parametro ottimale è stato approssimativamente costante attraverso le strategie. McAllester e al. hanno chiamato questo *invariante del livello di noise*. Hanno scoperto addirittura un principio più generale, che mostra che l'impostazione del parametro ottimale minimizza approssimativamente il rapporto della media della funzione obiettivo alla sua varianza. Hanno mostrato come questo *invariante dell'ottimalità* può essere utilizzato per impostare il parametro noise per un'unica istanza del problema, senza avere prima risolto quell'istanza od una simile. Come vedremo, il valore ottimale del parametro noise, per una strategia data, può essere estimato rapidamente e accuratamente analizzando le proprietà statistiche di molte esecuzioni corte della strategia.

Per verificare che questi invarianti non sono dovuti semplicemente alle proprietà speciali delle istanze random, poi McAllester e al. hanno confermato le loro scoperte su istanze estremamente strutturate dai domini di pianificazione e Graph Coloring.

I risultati presentati provvedono immediati orientamenti pratici per l'impostazione dei parametri per il WSAT e le sue versioni. McAllester e al. hanno ulteriormente ipotizzato che gli stessi invarianti si mantengono attraverso altre classi di procedure di ricerca locale, perché le versioni di WSAT che avevano considerato erano infatti basate su alcune di queste procedure. Lo stato attuale della teoria della ricerca locale non permette di dedurre analiticamente l'esistenza di questi invarianti.

Un'altra conseguenza pratica del lavoro di McAllester e al., è che può essere utilizzato per aiutare alla progettazione di nuove euristiche per la ricerca locale. Siccome le euristiche locali sono così sensibili all'impostazione del loro parametro noise, un'euristica si può solamente escludere se è esaminata alla sua calibrazione ottimale. Quando si verificano dozzine o centinaio di euristiche, comunque è proibitivo computazionalmente esaminare esaurientemente ogni parametro dell'impostazione. Comunque, nella ricerca delle euristiche migliori, l'impostazione dei parametri determinati dagli invarianti hanno prodotto costantemente le migliori performance per ogni strategia. Questo ha permesso a McAllester e

al. di identificare rapidamente due nuove euristiche che migliorano rispetto a tutte le altre variazioni del WSAT sulle istanze di prova.

Certamente, ci sono stati precedenti studi comparativi sulle performance dei diversi algoritmi di ricerca locale per il SAT. Per esempio, Gent e Walsh [**36**] hanno comparato le performance di alcune versioni del GSAT, concludendo che l'HSAT potrebbe risolvere più rapidamente i problemi random. Qui lo scopo è diverso: McAllester e al. sono meno interessati nel trovare il migliore algoritmo per le istanze random che nel trovare principi generali che rivelano se algoritmi differenti stanno percorrendo nella stessa maniera o no lo stesso spazio di ricerca. Parkes e Walser [**105**] hanno studiato una versione modificata del WSAT, mentre usando la funzione di minimizzazione del GSAT con il livello di noise impostato a $p = 50\%$. Loro conclusero che il WSAT originale era superiore alla versione modificata. Come vedremo, comunque, il parametro p ha valori ottimali diversi per strategie diverse, ed in particolare non è ottimale a 50% per il WSAT modificato. Un recente lavoro di Battiti e Protasi [**3**] è abbastanza simile allo studio presente, loro sviluppano uno schema di reazione per calibrare i parametri di noise degli algoritmi di ricerca locale SAT. Il loro calcolo è basato sulla distanza media di Hamming.

9.2 Procedure di ricerca locale per la soddisfacibilità booleana

Consideriamo gli algoritmi per risolvere i problemi di soddisfacibilità booleana in CNF. Nel 3SAT, ogni clausola contiene esattamente tre letterali distinti. Le clausole nella formula 3SAT random sono generate scegliendo a caso tre letterali distinti, e poi negando o no ognuno con uguale probabilità. Mitchell et al. [**98**] hanno mostrato che i problemi 3SAT random sono computazionalmente difficili, quando il rapporto tra clausole a variabili in tale formula è approssimativamente uguale a 4.3.

Una procedura di ricerca locale si muove in uno spazio di ricerca dove ogni punto è un assegnamento di verità delle variabili date. La procedura WSAT comincia considerando un assegnamento di verità random. Cerca una soluzione selezionando a caso ripetutamente una clausola violata, e poi utilizzando alcune euristiche per selezionare una variabile da modificare (flip - la modifica da $True$ a $False$ o vice-versa) in quella clausola.

La funzione obiettivo della ricerca locale per SAT tenta minimizzare il numero totale di clausole non soddisfatte. La caratteristica della strategia di ricerca che la induce a fare mosse che sono non ottimali - nel senso che le mosse aumentano o non riescono far decrescere la funzione obiettivo, anche quando sono disponibili tali mosse migliorativi nel vicinato locale dello stato corrente - è chiamato $noise$. Come notato prima, il noise permette alla procedura di ricerca locale di uscire dall'ottimo locale.

Ogni euristica descritta sotto prende un parametro che può variare la quantità del noise nella ricerca. Come vedremo, i valori presunti da questo parametro non sono direttamente comparabili attraverso le strategie: per esempio, un valore di 0.4 per una strategia può produrre una ricerca con mosse frequentemente non migliorative, della ricerca compiuta da una strategia diversa con lo stesso valore del parametro.

McAllester e al. hanno considerato sei euristiche per selezionare una variabile all'interno di una clausola. Le prime quattro sono variazioni delle procedure conosciute, mentre le ultime due sono nuove. Loro sono:

- G - con probabilità p scelgo qualsiasi variabile, altrimenti scelgo un variabile che minimizza il numero totale di clausole non soddisfatte. Il valore p è il parametro noise che varia da 0 a 1.
- B - con probabilità p scelgo qualsiasi variabile, altrimenti scelgo una variabile che minimizza il numero di clausole che sono $True$ nello stato corrente, ma che diventerebbe

False se il cambio sarebbe fatto. Nella descrizione originale del WSAT, questa è stata chiamata *minimizing breaks*. Di nuovo p è il parametro noise.

- *SKC* - come la precedente, ma non fa mai una mossa random se esiste una con break-value di 0. Notiamo che quando il break-value è 0, allora la mossa è garantita a migliorare anche la funzione obiettivo. Questo è il WSAT originale proposto da Selman, Kautz e Cohen (1994).

- *TABU* - la strategia è di scegliere una variabile che minimizza il numero delle clausole non soddisfatte. Comunque, ad ogni passo rifiuta modificare qualsiasi variabile che è stata modificata negli ultimi t passi; se tutte le variabili delle clausole non soddisfatte sono tabu, sceglie invece una clausola diversa non soddisfatta. Se tutte le variabili in tutte le clausole non soddisfatte sono tabu, allora la lista tabu è ignorata. La lista tabu di lunghezza t è il parametro noise.

- *NOVELTY* - questa strategia ordina le variabili in base al numero totale di clausole non soddisfatte, come fa *G*, ma rompendo il collegamento in favore della variabile modificata più recentemente. Consideriamo la migliore e la seconda migliore variabile sotto questo criterio. Se la migliore variabile non è la variabile più recentemente modificata nella clausola, la selezioniamo. Altrimenti, con probabilità p selezioniamo la seconda migliore variabile, e con probabilità $1 - p$ la migliore variabile.

- *R_NOVELTY* - questo è lo stesso come NOVELTY, eccetto il caso quando la migliore variabile è la più recentemente modificata. In questo caso, sia n la differenza della funzione obiettivo tra la migliore e la seconda migliore variabile. Notiamo che $n \geq 1$. Ci sono poi quattro casi:

  1. Quando $p < 0.5$ e $n > 1$, scelgo alla migliore
  2. Quando $p < 0.5$ e $n = 1$, allora con probabilità $2p$ scelgo la seconda migliore, altrimenti scelgo la migliore.
  3. Quando $p \geq 0.5$ e $n = 1$, scelgo la seconda migliore
  4. Quando $p \geq 0.5$ e $n > 1$, allora con probabilità $2\,(\,p - 0.5\,)$ scelgo la seconda migliore variabile, altrimenti la migliore.

L'intuizione dietro il NOVELTY e che si vuole modificare ripetutamente la stessa variabile indietro e in avanti. L'intuizione dietro il R_NOVELTY e che la funzione obiettivo dovrebbe influenzare la scelta tra la migliore e la seconda migliore variabile - una grande differenza della funzione obiettivo favorisce la migliore. Notiamo che la R_NOVELTY è quasi deterministico. Per rompere i loop deterministici nella ricerca, ogni 100 flips la strategia seleziona una variabile random dalla clausola. Anche se pochi flips comportino non determinismo, vedremo che le performance del R_NOVELTY sono ancora abbastanza sensibili all'impostazione del parametro p.

9.3 Invariante del livello di noise

Il noise nella ricerca locale può essere controllato da un parametro che specifica la probabilità di una mossa locale non ottimale, come nelle strategie G, B e SKC, NOVELTY, e R_NOVELTY. Invece le procedure tabu prendono un parametro che specifica la lunghezza della lista tabu. Le ricerche con liste tabu corte sono più suscettibili ai minimi locali (sono meno rumorose) delle ricerche con liste tabu lunghe.

Nella Figura 9.1 sono mostrati i risultati di una serie di esecuzioni delle diverse strategie come una funzione dell'impostazione del parametro noise, su una collezione di 400 istanze 3SAT random difficili. Sull'orizzontale è rappresentata la probabilità di una mossa random.

La lunghezza del tabu varia da 0 a 20, è stata normalizzato nel grafico alla serie $0 - 100$. L'asse verticale specifica la percentuale delle istanze risolte. Ogni punto di dati rappresenta 16.000 esecuzioni con una formula diversa per ognuna, dove il numero massimo di flips per esecuzione è fissato a 10.000.

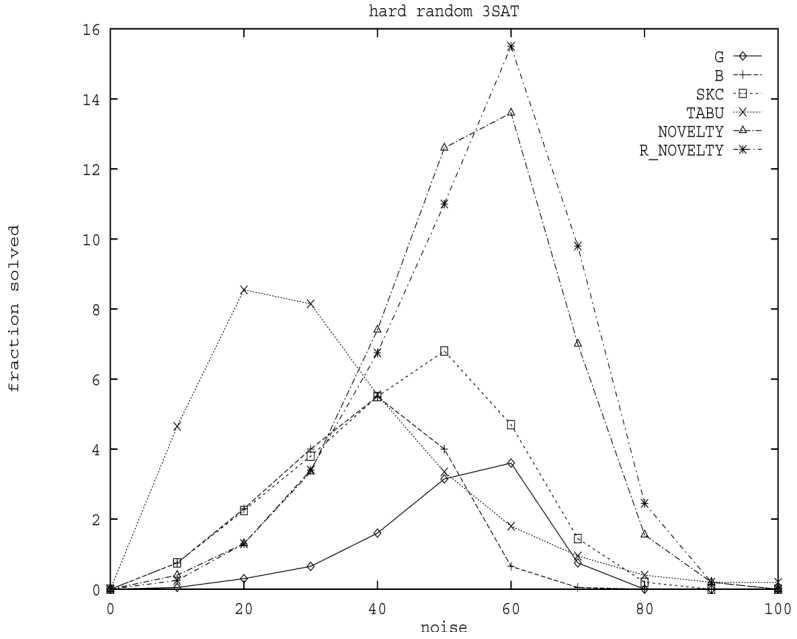

Figura 9.1: *Le sensitività al noise*

Sono stati rappresentati il valore del parametro noise rispetto alla frazione delle istanze che sono state risolte. Per esempio, R_NOVELTY ha risolto quasi 16% delle istanze del problema quando p è stato impostato su 60%. Considerando la frazione risolta con un numero fisso di flips ha permesso di raccogliere statistiche accurate sull'efficacia di ogni strategia. Se invece si tentava di risolvere ogni istanza, si affrontava il problema di trattare con la variazione alta nel run-time delle procedure stocastiche - per esempio, alcune esecuzioni potrebbero richiedere millioni di flips - ed il problema di trattare con esecuzioni che non convergono mai.

Come è chiaro dalla figura, le performance di ogni strategia variano altamente, dipendendo dall'impostazione del parametro noise. Per esempio, il funzionamento di R_NOVELTY ad un livello di noise di 40% invece di 60% degrada le sue performance con più di 50%. Inoltre, le performance ottimali appaiono ad impostazioni di parametri diversi per ogni strategia. Questo suggerisce che nella comparazione delle strategie si deve ottimizzare attentamente per ogni impostazione del parametro, e che addirittura modifiche minori ad una strategia richiedono che i parametri devono essere riaggiustati in modo appropriato.

Data la precedente osservazione, sorge la domanda: esiste una caratterizzazione migliore del livello noise che è meno sensibile ai dettagli delle strategie individuali?

McAllester e al. hanno esaminato un numero di misure diverse del comportamento delle strategie di ricerca locale. Definiamo il *livello noise normalizzato* di una procedura di ricerca su un'istanza del problema dato come il valore medio della funzione obiettivo durante un'esecuzione di quell'istanza. Allora, si può osservare che il livello noise normalizzato *ottimale* è approssimativamente costante attraverso le strategie. Questo è illustrato nella Figura 9.2.

In altre parole, quando il parametro noise è ottimamente calibrato per ogni strategia, allora il numero medio delle clausole non soddisfatte durante un'esecuzione è approssimativamente lo stesso attraverso le strategie.

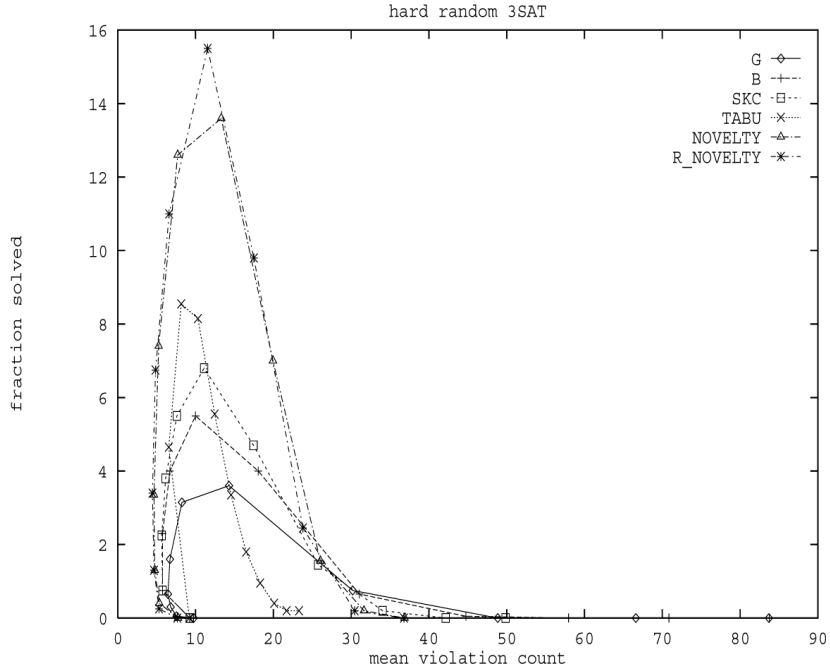

Figura 9.2: *Invariante del livello di noise normalizzato nelle formule random*

Chiamiamo questo fenomeno *invariante del livello di noise*, che rappresenta un attrezzo utile per la progettazione e per calibrare i metodi di ricerca locale. Una volta McAllester e al. hanno determinato il conteggio medio delle violazioni dando le performance ottimali per una

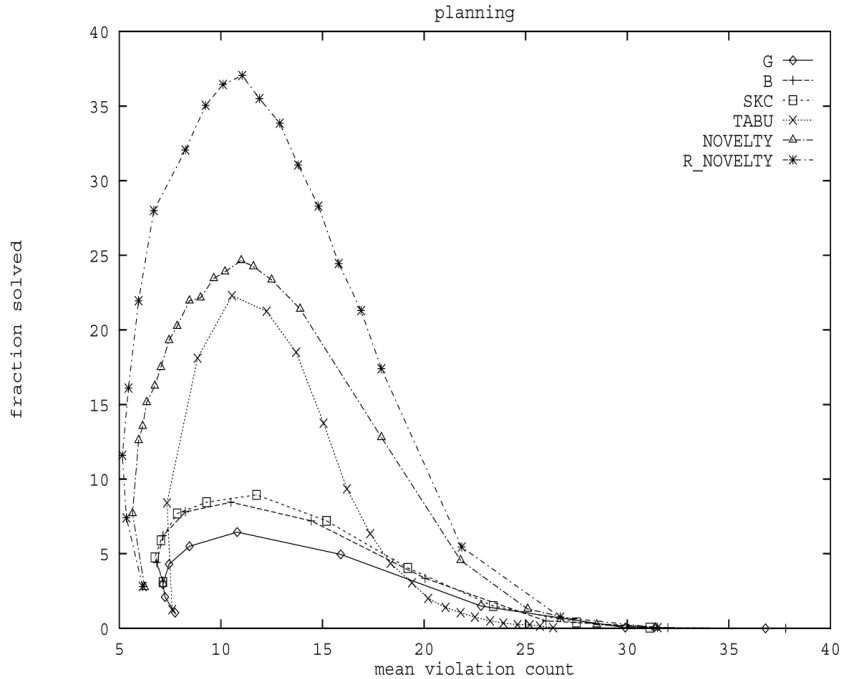

Figura 9.3: *Invariante del livello di noise normalizzato nelle formule di pianificazione*

singola strategia su una distribuzione data di problemi, hanno potuto semplicemente calibrare

le strategie altre per funzionare allo stesso conteggio medio delle violazioni, nella conoscenza che questo ci darà vicino alle performance ottimali.

Dopo aver ipotizzato l'esistenza di questo invariante, McAllester e al. hanno voluto verificare anche per le classi del mondo reale, problemi strutturati di soddisfacibilità. Le Figure 9.3 e 9.4 presentano la conferma di quest'evidenza.

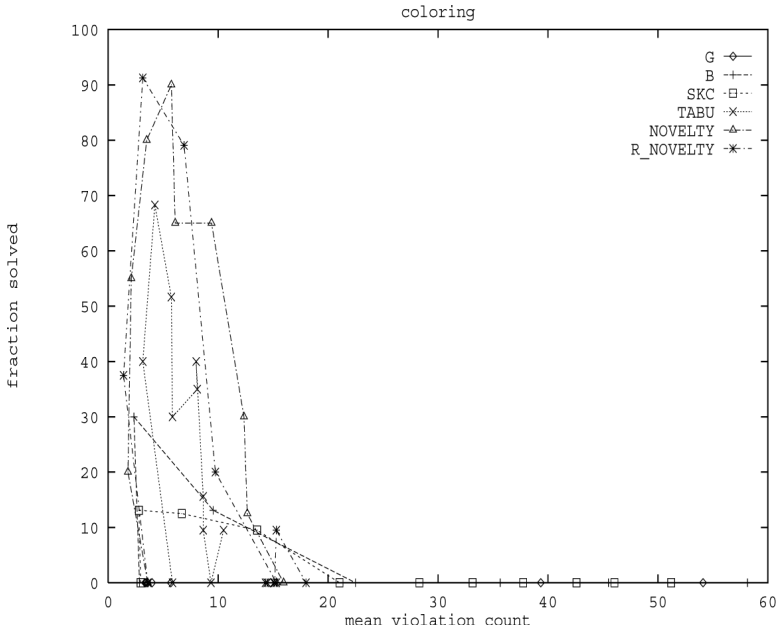

Figura 9.4: *Invariante del livello di noise normalizzato nelle formule di graph coloring*

La Figura 9.3 è basata sulla risoluzione di un problema di soddisfacibilità che codifica un problema di pianificazione blocksworld, l'istanza $bw\_large.a$ da Kautz e Selman [**78**]. Il problema originale è di trovare un piano 6-step che risolve un problema di pianificazione che comporta 9 blocchi, dove ogni passo nuove un blocco. Dopo il problema è codificato e semplificato dall'unit propagation, contiene 459 variabili e 4675 clausole. Ogni procedura stocastica è stata eseguita 16.000 volte. Notiamo che questo è diverso del caso con le formule random, dove è stata generata una formula diversa per ogni prova.

Figura 9.4 mostra l'invariante del livello di noise su una codifica SAT di un problema di Graph Coloring. L'istanza è basata su 18 colori di 125 nodi di un grafo (Johnson e al. [**70**]). Questa formula contiene 2.250 variabili e 70.163 clausole. Siccome questa formula è così grande, McAllester e al. non hanno potuto compiere molte esecuzioni come nei precedenti esperimenti. Ogni punto è basato su appena 1.000 campioni, e questo spiega la natura piuttosto irregolare delle curve.

L'invariante del livello di noise non implica che tutte le strategie sono equivalenti in termini del loro livello di performance ottimali.

McAllester e al. hanno sperimentato con un gran numero di euristiche per selezionare la variabile da cambiare nel WSAT per risolvere i problemi di soddisfacibilità booleana. L'invariante del livello di noise ha permesso di valutare rapidamente più di 50 versioni del WSAT, e ognuna è stato esaminata con il suo livello di noise ottimale. Questo ha portato allo sviluppo delle strategie NOVELTY e R_NOVELTY che costantemente migliorano quasi due volte rispetto alle altre versioni.

9.4 Invariante di ottimalità

L'invariante del livello di noise dà delle possibilità di trattare con la sensività del noise delle procedure di ricerca locale. Comunque per usarlo si ha bisogno di essere capaci di raccogliere delle statistiche sulla rata di successo di almeno una strategia attraverso un esemplare di una distribuzione di problemi. In pratica McAllester e al. hanno affrontato spesso la necessità di risolvere una particolare istanza di un nuovo problema. Inoltre, quest'istanza può essere estremamente difficile, e risolverla può richiedere una grande quantità di calcolo anche all'impostazione del noise ottimale. Perciò, si desidera di predire rapidamente l'impostazione del parametro di noise per una singola istanza del problema, senza doverla risolvere.

Lo studio empirico della sensività del noise, ha prodotto un principio preliminare per impostare i parametri di noise in base alle *proprietà statistiche* della ricerca. McAllester e al. hanno eseguito molte esecuzioni corte della procedura di ricerca, e hanno registrato il valore finale della funzione obiettivo per ogni esecuzione e la variazione dei valori su quell'esecuzione. Poi, hanno preso la media di questi valori.

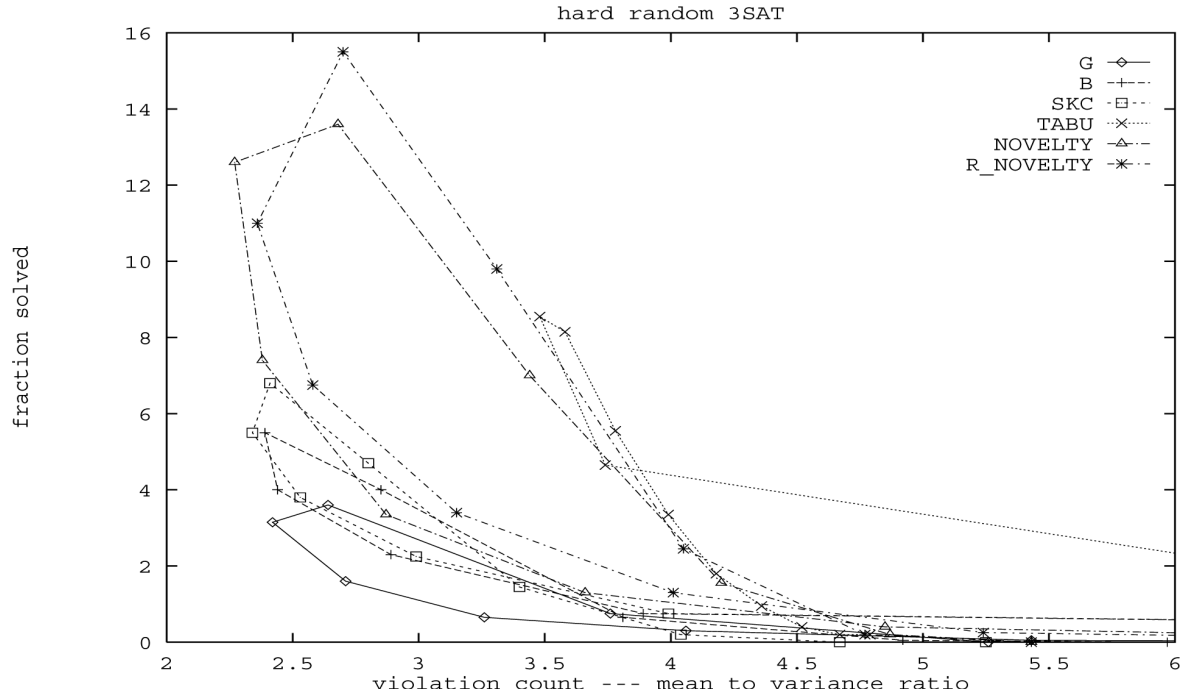

Figura 9.5: *Tuning noise sulle istanze random*

A livelli bassi del noise, il valore medio della funzione obiettivo è piccolo - sono stati raggiunti degli stati con bassi numeri di clausole non soddisfatte. Comunque, la variazione è molto piccola; così piccola che l'algoritmo raggiunge raramente uno stato con 0 clausole non soddisfatte. Quando accade questo, l'algoritmo si blocca in un minimo locale profondo. D'altra parte a livelli di noise alti, la variazione è grande, ma il numero medio di clausole non soddisfatte è ancora più grande. Ancora una volta, è improbabile che l'algoritmo raggiunga uno stato con 0 clausole non soddisfatte.

Perciò, McAllester e al. hanno dovuto trovare l'equilibrio corretto fra il valore medio e la variazione. I loro sperimenti mostrano che il rapporto fra il valore medio e la variazione assicura un utile equilibrio. Infatti, le performance ottimali sono ottenute quando il valore del noise è lievemente sopra il valore ottenuto quando il rapporto e minimizzato.

Hanno chiamato quest'osservazione *invariante di ottimalità*. Inoltre, questi invarianti si mantengono per tutte le variazioni del WSAT che loro hanno considerato.

Per illustrare il principio, McAllester e al. hanno presentato nella Figura 9.5 i problemi risolti come una funzione del rapporto tra il valore medio e la variazione su una collezione di istanze di problemi random difficili. Per ogni strategia i dati puntano a formare un loop.

Traversando il loop in una direzione destrorsa che comincia dall'angolo basso in destra corrisponde a incrementare il livello di noise da 0 al suo valore massimo. Nello stesso punto durante questo attraversamento si raggiunge un valore minimo del rapporto tra il valore medio e la variazione. Per esempio, la strategia R_NOVELTY (la curva più alta) ha un minimo del rapporto tra il valore medio e la variazione intorno a 2.5. A quel punto risolve approssimativamente 11% delle istanze. Alzando il noise, e aumentando di nuovo il rapporto tra il valore medio e la variazione, giungiamo alle performance di picco di 15% ad un rapporto di 2.8. Si osserva lo stesso modello per tutte le strategie. Nei loro esperimenti McAllester e al. hanno trovato le performance ottimali quando il rapporto è approssimativamente 10% più alto del suo valore minimo.

Le Figure 9.6 e 9.7 confermano di nuovo questa osservazione sulle istanze di pianificazione e Graph Coloring. Di nuovo si vede che tutte le curve raggiungono leggermente il loro picco alla destra del rapporto tra il valore medio e la variazione.

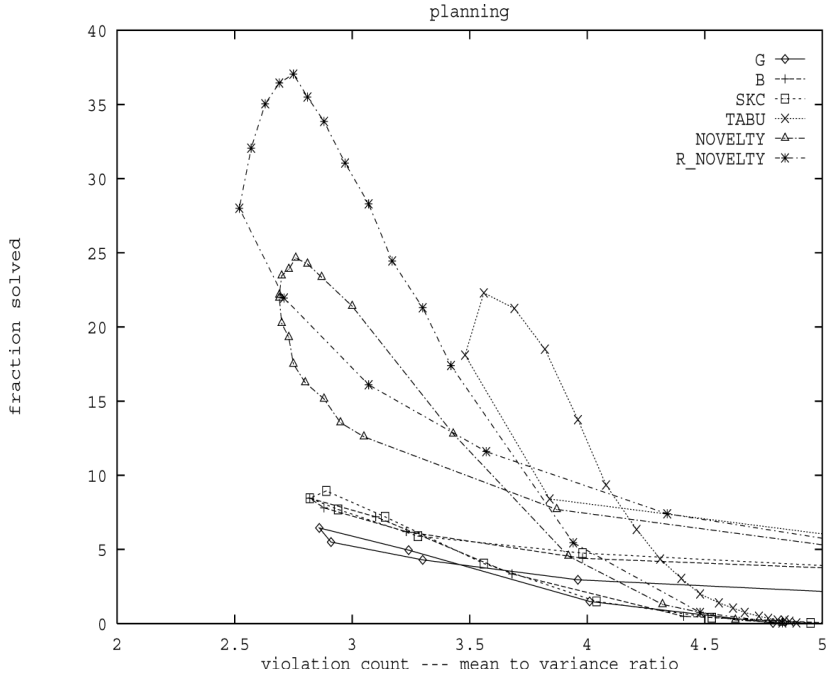

Figura 9.6: *Tuning noise sulle istanze di pianificazione*

McAllester e al. hanno sottolineato di nuovo che misurando il rapporto tra il valore medio e la variazione non richiede la risoluzione dell'istanza del problema. Per ogni valore di noise, hanno potuto misurare il rapporto facendo semplicemente molte esecuzioni corte e hanno calcolato il valore medio e la variazione del conteggio delle violazioni durante l'esecuzione. Poi, ripetendo questa procedura per impostazioni diverse del parametro di noise, sono riusciti a determinare le impostazioni necessarie per ottenere il rapporto tra il valore medio e la variazione che è 10% sopra il suo minimo.

Un altro modo di risolvere un'istanza di un unico problema è avviare l'esecuzione ad un livello di noise arbitrario. Poi si può misurare il rapporto tra il valore medio e la variazione durante l'esecuzione, e aggiustare dinamicamente il livello di noise per ottenere le performance ottimali.

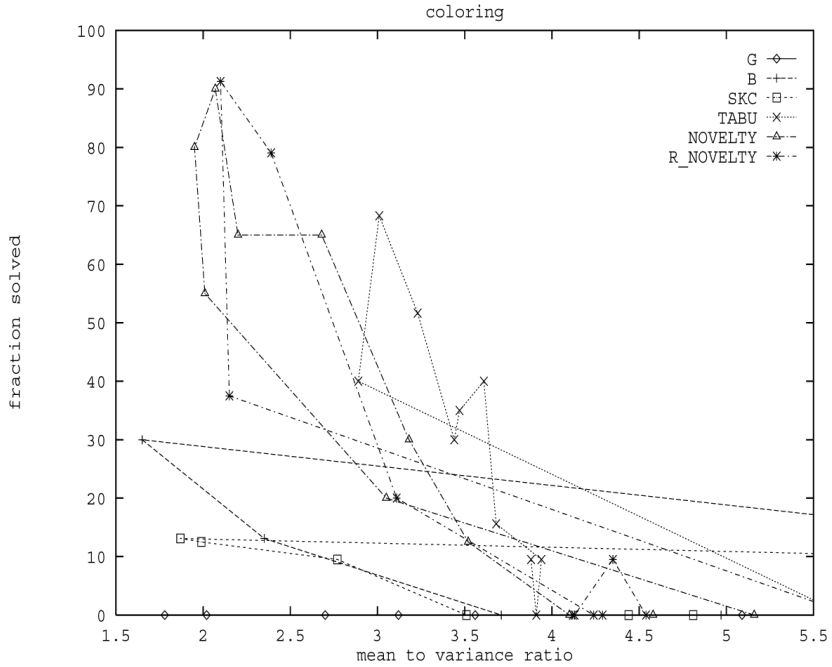

Figura 9.7: *Tuning noise sulle istanze di coloring*

10 Conclusioni

Selman e al. hanno comparato vari meccanismi per uscire dai minimi locali in problemi di soddisfacibilità: simulated annealing, random noise, e random walk combinata. La strategia walk introduce perturbazioni nello stato corrente che sono relativi direttamente ai vincoli non soddisfatti del problema. Gli esperimenti di Selman e al. mostrano che questa strategia, migliora significativamente rispetto al simulated annealing e random noise su molte classi di problemi di soddisfacibilità. Le ultime due strategie possono produrre perturbazioni che sembrano meno focalizzate, nel senso che possono coinvolgere variabili che non appaiono in alcuna clausola non soddisfatta.

Il miglioramento relativo trovato utilizzando random walk su altri metodi aumenta con l'aumento della dimensione del problema.

Selman e al. hanno anche mostrato che GSAT con walk è notevolmente efficiente nel risolvere problemi di sintesi dei circuiti. Questo risultato è specialmente interessante perché i problemi di sintesi non hanno nessun elemento casuale, e sono molto difficili per i metodi sistematici.

Finalmente, Selman e al. hanno dimostrato che GSAT con walk migliora anche sui migliori algoritmi per MAX-SAT. Data l'efficacia della strategia random walk combinata su problemi di soddisfacibilità booleana, una direzione interessante per la ricerca futura potrebbe essere l'esplorazione di strategie simili sui problemi con soddisfacimento dei vincoli.

L'incremento del look-back chiaramente rende il DP un algoritmo più efficace. Per quasi ognuna delle istanze esaminate da Bayardo e Schrag, l'apprendimento e CBJ sono stati critici per ottenere delle buone performance. Bayardo e Schrag hanno sospettato che il miglioramento delle performance risultato dall'inclusione del look-back sia infatti dovuto alla sinergia tra le tecniche look-ahead e look-back applicate. L'euristica di selezione della variabile tenta di cercare l'area con i vincoli maggiori dello spazio di ricerca per realizzare al piu presto possibile fallimenti inevitabili. Gli schemi d'apprendimento, attraverso la memorizzazione delle clausole ricavate, possono creare sottospazi di ricerca vincolati da sfruttare dall'euristica di selezione della variabile.

L'apprendimento size-bounded è efficace quando le istanze hanno dei nogood relativamente corti che possono essere ricavati senza inferenza. L'apprendimento relevance-bounded è efficace quando anche molti sottoproblemi che corrispondono all'assegnamento DP corrente hanno questa proprietà. Le scoperte di Bayardo e Schrag indicano che le istanze del mondo reale contengono spesso sottoproblemi con nogoods corti facilmente ricavati. Le istanze della phase transition del *3*SAT random tendono ad avere dei nogood molto corti [**111**], ma questi sembrano richiedere inferenza profonda, ed il DP look-back-enhanced assicura un piccolo vantaggio su di loro [**4**].

Poche istanze di test sono state non fattibili per il DP look-back-enhanced, ma facili oppure banali per WSAT. Tuttavia, combinando tecniche buone per look-ahead look-back e possibile ottenere performance migliori attraverso una larga serie di problemi.

Li e Ambulagan hanno proposto una procedura di DPL molto semplice che assume solamente delle tecniche look-ahead: un'euristica con ordinamento delle variabili, un controllo di consistenza in avanti FCC e una risoluzione limitata prima della ricerca, dove l'euristica è se stessa basata sull'unit propagation. I risultati sperimentali comparativi del Satz sui problemi 3SAT random con tre fra le migliori procedure DPL nella letteratura, mostra che questo è molto efficace sui problemi *3*SAT random. Mentre i migliori algoritmi nella letteratura, per risolvere i problemi SAT strutturati, utilizzano di solito le entrambe tecniche, look-ahead e look-back, e accentuano l'ultimo, Satz arriva a migliorare le loro performance su molti problemi SAT strutturati. I risultati suggeriscono che un utilizzo appropriato della

tecnica look-ahead, può permettere di fare a meno, per molti problemi SAT strutturati, delle sofisticate tecniche look-back in una procedura DPL.

Se una procedura DPL è efficace per i problemi SAT random, dovrebbe essere efficace anche per molti problemi strutturati. Per problemi in cui l'euristica UP non riesce a distinguere le variabili, per esempio problemi con molte simmetrie, Satz può essere ancora migliorata con tecniche d'apprendimento.

McAllester e al. hanno presentato due misure statistiche del progresso degli algoritmi di ricerca locale che permettono di trovare rapidamente l'impostazione del noise ottimale. Prima, McAllester e al. hanno mostrato che il valore medio ottimale della funzione obiettivo è approssimativamente costante attraverso le diverse strategie di ricerca locale.

Secondo, McAllester e al. hanno mostrato che si possono ottimizzare le performance di una procedura di ricerca locale misurando il rapporto tra il valore medio e la variazione della funzione obiettivo durante l'esecuzione. La seconda misura permette di trovare buone impostazioni del livello del noise per classi di problema non risolte precedentemente.

Finalmente, McAllester e al. hanno applicato questi principi per valutare le nuove euristiche di ricerca locale, e di conseguenza scoprire due nuove euristiche migliorativi rispetto ad altre versioni del WSAT su tutti i dati di test.

In conclusione possiamo affermare che c'è stato un progresso considerevole fatto nello studio della soddisfacibilità proposizionale negli ultimi 10 anni. Nel presente lavoro ci siamo occupati ad analizzare soltanto alcuni di questi algoritmi, che a nostro parere hanno rappresentato un passo importante nello sviluppo della soddisfacibilità proposizionale. Il progresso è stato sia teoretico che pratico. Sul lato pratico, ora abbiamo procedure di ricerca completi e locali capaci risolvere problemi con centinaio e, in alcuni casi, migliaia di variabili. Molti problemi d'interesse pratico sono stati anche risolti codificandoli in SAT. Sul lato teoretico, abbiamo, un ritratto più sofisticato di quello che rende i problemi SAT difficili da risolvere.Cosa ci può aspettare nel futuro?

SAT rimane un campo dove forse gli esperimenti sono probabilmente in anticipo rispetto alla teoria. Possiamo aspettarci perciò, che la teoria faccia qualche accelerazione, specialmente nell'analisi teoretica dei metodi di ricerca locale. Possiamo aspettarci che la codifica dei problemi in SAT potrà focalizzare la comunità di ricerca sulla rappresentazione e formulazione dei problemi. Questi sono temi che risalgono ai primi giorni dell'AI e quali rimangono centrali alla risoluzione dei problemi di ricerca difficili.

11 Biblioteche di benchmarck ed altre risorse

Molti di questi problemi sono stati raccolti in biblioteche di benchmarck. Una delle più vecchie è la biblioteca di benchmarck che è stata sviluppata per la DIMACS International Algorithm Implementation Challenge del 1993. Molte centinaia di problemi da questa competizione possono essere trovati a:
ftp://dimacs.rutgers.edu/pub/challenge/soddisfacibilità/benchmarks/cnf/.

Questa competizione ha proposto anche un formato testo semplice per specificare problemi di soddisfacibilità che da allora è diventato uno standard.

Un'altra biblioteca di benchmarck è stata creata per International Competition and Symposium on Satisfiability Testing tenuto nel marzo 1996 a Beijing. I problemi da questa biblioteca sono disponibili presso: http://www.cirl.uoregon.edu/jc/beijing.

Più recentemente, Hoos e Stutzle hanno creato la biblioteca di SATLIB a http://www.informatik.tu-darmstadt.de/AI/SATLIB, che contiene molti problemi benchmarck, con link ai vari risolutori e altre risorse.

12 Bibliografia e altri documenti riguardanti la sodisfacibilita proposizionale

1. Michael N. Barber. Finite-size scaling. In Phase Transitions and Critical Phenomena, Volume 8, pages 145-266. Academic Press, 1983

2. P. Barth. A Davis-Putnam based enumeration algorithm for linear pseudoboolean optimization. Research report mpi-i-95-2-003, Max Plack Institut fur Informatik, Saarbrucken, 1995

3. R. Battiti and M. Protasi. Reactive Search, a history based heuristic for MAX-SAT. Technical report, Dipartimento di Matematica, Univ. of Trento, Italy

4. Roberto Bayardo and Robert Schrag. Using CSP look-back techniques to solve exceptionally hard SAT instances. In Proceedings of Second International Conference on Principles and Practice of Constraint Programming (CP96), 1996

5. R. J. Bayardo and D. P. Miranker. A Complexity Analysis of Space-Bounded Learning Algorithms for the Constraint Satisfaction Problem. In Proceedings 13th National Conference on Artificial Intelligence, 558-562, 1996

6. Roberto Bayardo and Robert Schrag. Using CSP look-back techniques to solve real-world SAT instances. In Proceedings of the 14th National Conference on AI. American Association for Artificial Intelligence, 1997

7. R. Beigel and D. Eppstein. 3-coloring in time $O(1.3446^n)$: a No-MIS algorithm. Technical report, ECCC, 1995. TR95-33

8. R.E. Bryant. Symbolic Boolean manipulation with ordered binary-decision diagrams. ACM Computing Surveys, 24(3):293-318, 1992

9. M. Buro and H. Buning. Report on a SAT competition. Technical Report, Dept. of Mathematics and Informatics, University of Paderborn, Germany, 1992

10. M. Cadoli, A. Giovanardi, and M. Schaerf. Experimental analysis of the computational cost of evaluation quantified Boolean formulae. In Proceedings of the AI*IA-97, pages 207-218. Springer-Verlag, LNAI-1321, 1997

11. P. Cheeseman, B. Kanefsky, and W.M. Taylor. Where the really hard problems are. In Proceedings of the 12th IJCAI, pages 331-337. International Joint Conference on Artificial Intelligence, 1991

12. V. Chvatal and B. Reed. Mick gets some (the odds are on his side). In Proceedings of the 33rd Annual Symposium on Foundations of Computer Science, pages 620-627. IEEE, 1992

13. S.A. Cook. The complexity of theorem proving procedures. In Proceedings of the 3rd Annual ACM Symposium on the Theory of Computation, pages 151-158, 1971

14. S.A. Cook and D.G. Mitchell. Finding hard instances of the satisfiability problem: A survey. In Satisfiability Problem: Theory and Applications. Dimacs Series in Discrete Mathematics and Theoretical Computer Science, Volume 35, 1997

15. J.M. Crawford and L.D. Auton. Experimental Results on the Cross-Over Point in Satisfiability Problemes. In Proceedings of AAAI 1993 Spring Symposium on AI and NP-Hard Problems, 1993

16. J.M. Crawford and L.D. Auton. Experimental results on the crossover point in random 3-SAT. Artificial Intelligence, 81:31-57, 1996

17. J.M. Crawford and A.D. Baker. Experimental Results on the Application of Satisfiability Algorithms to Scheduling Problems. In Proceedings of 12th National Conference on AI, pages 1092--1097. American Association for Artificial Intelligence, 1994

18. Marcello D'Agostino. Are tableaux an improvement on truth-tables? Journal of Logic, Language and Information

19. M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. Comms. ACM, 5:394-397, 1962

20. M. Davis and H. Putnam. A computing procedure for quantification theory. Journal of the Association for Computing Machinery, pages 201-215, 1960

21. R. Dechter and I. Rish. Directional resolution: The Davis-Putnam procedure, revisited. In Proceedings of KR-94, pages 134-145, 1994. A longer version is available from http://www.ics.uci.edu/irinar

22. O. Dubois, P. Andre, Y. Boufkhad, and J. Carlier. SAT versus UNSAT. In Proceedings of the Second DIMACS Challenge, 1993

23. K. A. Dowsland. Simulated annealing. In Modern Heuristic Techniques for Combinatorial Problems,C.R. Reeves (E.), John Wiley & Sons, 20-69, 1988

24. G.W. Ernst. A definition-driven theorem prover. In Proceedings of the 3rd IJCAI, pages 51-55. International Joint Conference on Artificial Intelligence, 1973

25. J. Frank. Learning short term weights for GSAT. In Proceedings of the 14th IJCAI. International Joint Conference on Artificial Intelligence, 1995

26. J. W. Freeman. Improvements to Propositional Satisfiability Search Algorithms. Ph.D. Dissertation, U. Pennsylvania Dept. of Computer and Information Science, 1995

27. E. Friedgut. Sharp thresholds for graph properties and the k-SAT problem, 1998

28. Frieze and S. Suen. Analysis of two simple heuristics on a random instance of k-SAT. Journal of Algorithms, 20:312-355, 1996

29. D. Frost, I. Rish, and L. Vila. Summarizing CSP hardness with continuous probability distributions. In Proceedings of the 14th National Conferenceon AI, pages 327--333. American Association for Artificial Intelligence, 1997

30. Masayuki Fujita, John Slaney and Frank Bennett. Automatic generation of some results in finite algebra. In Proceedings of the 13th IJCAI, pages 52-57. International Joint Conference on Artificial Intelligence, 1993

31. M.R. Garey and D.S. Johnson. Computers and intractability: a guide to the theory of NP-completeness. W.H. Freeman, 1979

32. I.P. Gent. On the stupid algorithm for satisfiability. Technical report, APES-03-1998, 1998, available from http://www.cs.strath.ac.uk/apes/reports/apes-03-1998.ps.gz

33. I.P.Gent, E. MacIntyre, P. Prosser, and T. Walsh. The constrainedness of search. In Proceedings of AAAI-96, pages 246-252, 1996

34. I.P. Gent, E. MacIntyre, P. Prosser, and T. Walsh. The scaling of search cost. In Proceedings of the 14th National Conference on AI, pages 315-320. American Association for Artificial Intelligence, 1997

35. Gent and T. Walsh. The Enigma of SAT Hill-climbing Procedures. Technical Report 605, Dept. of Artificial Intelligence, University of Edinburgh, 1992

36. Gent and T. Walsh. Towards an Understanding of Hill-climbing Procedures for SAT. In Proceedings of the 11th National Conference on AI.American Association for Artificial Intelligence, 1993

37. I.P. Gent and T. Walsh. An empirical analysis of search in GSAT. Journal of Artificial Intelligence Research, 1:23-57, 1993

38. I.P. Gent and T. Walsh. Easy problems are sometimes hard. Artificial Intelligence, pages 335-345, 1994

39. I.P.Gent and T. Walsh. The hardest random SAT problems. In Proceedings of KI-94, Saarbrucken, 1994

40. I.P. Gent and T. Walsh. The SAT phase transition. In A G Cohn, editor, Proceedings of 11th ECAI, pages 105-109. John Wiley & Sons, 1994

41. I.P. Gent and T. Walsh. Unsatisfied variables in local search. Research Paper 721, Dept. of Artificial Intelligence, University of Edinburgh, 1994. Presented at AISB-95

42. I.P. Gent and T. Walsh. The satisfiability constraint gap. Artificial Intelligence, 81(1-2), 1996

43. I.P. Gent and T. Walsh. Beyond np: the QSAT phase transition. Technical report APES-05-1998, 1998, available from http://www.cs.strath.ac.uk/apes/reports/apes-05-1998.ps.gz

44. I.P. Gent and T. Walsh. The Search for Satisfaction, Department of Computer Science, University of Strathclyde, Glasgow, 1999

45. M. Ginsberg and D. McAllester. GSAT and Dynamic Backtracking, Principles of Knowledge Representation and Reasoning: Proceedings of the Fourth International Conference, 226-237, 1994

46. F. Giunchiglia and R. Sebastiani. Building decision procedures for modal logics from propositional decision procedures - the case study of modal k. In Proceedings of the 13th International Conference on Automated Deduction (CADE-13). Springer Verlag, 1996

47. F. Giunchiglia and R. Sebastiani. A SAT-based decision procedure for alc. In Proceedings of the 5th International Conference on Principles of Knowledge. Representation and Reasoning - KR'96, 1996

48. F. Giunchiglia and R. Sebastiani. A new method for testing decision procedures in modal logics. In Proceedings of the 14th International Conference on Automated Deduction (CADE-14). Springer Verlag, 1997

49. P.Y. Gloess. ULog - a Unified Object Logic, Revue d'intelligence artificielle, Vol. 5, No. 3, pp. 33-66, 1991

50. F. Glover. Tabu Search - Part I. ORSA Journal on Computing, 1(3):190-206, 1989

51. F. Glover. Future path for integer programming and links to artificial intelligence. Computers & Ops. Res. , 5:533-549, 1993

52. F. Glover and M. Laguna. Tabu search. In Modern Heuristic Techniques for Combinatorial Problems, C.R. Reeves (E.), John Wiley & Sons, 70-150, 1996.

53. Goerdt. A threshold for unsatisfiability. In I. Havel and V. Koubek editors, Mathematical Foundations of Computer Science, Lecture Notes in Computer Science, pages 264-274. Springer Verlag, 1992

54. Gomes, B. Selman, and H. Kautz. Boosting combinatorial search through randomization. In Proceedings of 15th National Conference on Artificial Intelligence, pages 431-437. AAAI Press - The MIT Press, 1998

55. J. Gu. Efficient local search for very large-scale satisfiability problems. SIGART Bulletin, 3(1):8-12, 1992

56. J. Gu. Global Optimisation for satisfiability (SAT) problem. IEEE Transactions on Data and Knowledge Engineering, 6(3):361-381,1994

57. J. Hansen and B. Jaumard. Algorithms for the maximum satisfiability problem. Computing, 44, 279-303, 1990

58. P.Hayes. The naive physics manifesto. In D. Michie, editor, Expert Systems in the Microelectronic Age. Edinburgh University Press, 1979

59. A. Hertz and D. De Werra. Using tabu search techniques for Graph Coloring. Computing, pages 345-351, 1987

60. T. Hogg. Quantum computing and phase transitions in combinatorial search. Journal of Artificial Intelligence Research, 4:91-128, 1996

61. J. N. Hooker. Resolution vs. cutting plane solution of inference problems: some computational experience. Operations Research Letters, 7(1):1-7, 1988

62. J. N. Hooker and C. Fedjki. Branch-and-cut solution of inference problems in propositional logic. Annals of Mathematics and Artificial Intelligence, 1:123--139, 1990

63. J.N. Hooker. Branch-and-cut solution of inference problems in propositional logic. Annals of Mathematics and Artificial Intelligence, 1:123-139, 1990

64. J.N. Hooker. Solving the incremental satisfiability problem. Journal of Logic Programming, 15:177-186, 1993

65. J.N. Hooker and V. Vinay. Branching Rules for Satisfiability. Journal of Automated Reasoning, 15:359-383, 1995

66. H. Hoos. Stochastic Local Search - Methods, Models, Applications. PhD thesis, TU Darmstadt, 1998. Available from http://www.cs.ubc.ca/spider/hoos/publ-ai.html

67. H. Hoos and T. Stutzle. Evaluating Las Vegas algorithms - pitfalls and remedies. In Proceedings of 14th Annual Conference on Uncertainty in Artificial Intelligence (UAI-98), 1998. Available from http://www.sis.pitt.edu/~dsl/uai98.html

68. R. E. Jeroslow and J. Wang. Solving satisfiability propositional problems. Annals of Mathematics and Artificial Intelligence, 1:167-187, 1990

69. D.S. Johnson. Optimization algorithms for combinatorial problems. J. of Comp. and Sys. Sci., 9:256-279, 1974

70. D.S. Johnson, C.R. Aragon, L.A. McGeoch and C. Schevon.Optimization by simulated annealing: an experimental evaluation; part II, Graph Coloring and number partioning. Operations Research, 39(3):378-406, 1991

71. Y. Jiang, H. Kautz, and B. Selman. Solving problems with hard and soft constraints using a stochastic algorithm for MAX-SAT. In First International Joint Workshop on Artificial Intelligence and Operations Research, 1995

72. K.A. De Jong and W.M. Spears. Using Genetic Algorithms to Solve NP-Complete Problems. In Third International Conference on Genetic Algorithms. Morgan Kaufmann, 1989

73. A.P. Kamath, N.K. Karmarkar, K.G. Ramakrishnan and M.G.C. Resende.A continuous approach to inductive inference. Mathematical Programming, 57, 215-238, 1991

74. A.P. Kamath, R. Motwani, K. Palem, and P. Spirakis. Tail bounds for occupancy and the satisfiability threshold conjecture. Randomized Structure and Algorithms, 7:59-80, 1995

75. A.P. Kamath, N.K. Karmarkar, K.G. Ramakrishnan, and M.G.C. Resende. An interior point approach to Boolean vector function synthesis. In Proceedings of the 36th MSCAS, 1993

76. K. Kask and R. Dechter. GSAT and local consistency. In Proceedings of the 14th IJCAI, pages 616-622. International Joint Conference on Artificial Intelligence, 1995

77. H. Kautz and B. Selman. Planning as Satisfiability. In Proceedings of the 10th ECAI, pages 359-363. European Conference on Artificial Intelligence, 1992

78. H. Kautz and B. Selman. Pushing the envelope: planning, propositional logic, and stochastic search. In Proceedings of the 13th National Conference on AI, pages 1194-1201. American Association for Artificial Intelligence, 1996

79. H. Kautz and B. Selman. BLACKBOX: A new approach to the application of theorem proving to problem solving. In Working notes of the Workshop on Planning as Combinatorial Search, 1998. Held in conjunction with AIPS98, Pittsburgh, PA, 1998

80. H. Kautz and B. Selman. The role of domain-specific knowledge in the planning as satisfiability framework. In Proceedings of AIPS-98, Pittsburgh, PA, 1998

81. S. Kim, H. Zhang. ModGen: Theorem proving by model generation, Proceedings of the 12th National Conference on Artificial Intelligence (AAAI94), pp. 162-167, 1994

82. S. Kirkpatrick, C.D. Gelatt Jr., and M.P.Vecchi. Optimisation by Simulated Annealing. Science, 220:671-680, 1983

83. S. Kirkpatrick and B. Selman. Criticial comportamento in the satisfiability of random boolean expressions. Science, 264:1297-1301, 1994

84. L.M. Kirousis, E. Kranakis, and D. Krizanc. Approximating the unsatisfiability threshold of random formulas. In Proceedings of the 4th Annual European Symposium on Algorithms (ESA'96), pages 27-38, 1996

85. E. Koutsoupias and C.H. Papadimitriou. On the greedy algorithm for satisfiability. Information Processing Letters, 43:53-55, 1992

86. T. Larrabee. Test pattern generation using Boolean satisfiability. IEEE Transactions on Computer-Aided Design, pages 4-15, 1992

87. C.M. Li, Exploiting yet more the power of unit clause propagation to solve 3SAT problem, Proceedings of the ECAI'96 Workshop on Advances in propositional deduction, Budapest, Hungary, pp. 1116, August 1996

88. C.M. Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In Proceedings of the 15th IJCAI, pages 366-371. International Joint Conference on Artificial Intelligence, 1997

89. C.M. Li and Anbulagan, Look-ahead versus look-back for satisfiability problems, in preceedings of third international conference on Principles and Practice of Constraint Programming - CP97, Springer-Verlag, LNCS 1330, Page 342-356, Autriche, 1997.

90. R.J. Lipton. Using DNA to solve NP-complete problems. Science, 268:542-545, 1995

91. M. Mazure, L. Sais and E. Gregoire. Detecting Logical Inconsistencies. In Proc. of the Fourth International Symposium on Artificial Intelligence and Mathematics, 116-121, 1996

92. M. Mazure, L. Sais, and E. Gregoire. Tabu search for SAT. In Proceedings of 14th National Conference on Artificial Intelligence, pages 281-285. American Association for Artificial Intelligence, AAAI Press/The MIT Press, 1997

93. McAllester, B. Selman, and H. Kautz. Evidence for invariants in local search. In Proceedings of the 14th National Conference on AI, pages 321-327. American Association for Artificial Intelligence, 1997

94. W. W. McCune. Otter user's guide 0.91. Maths and CS Division, Argonne National Laboratory, Argonne, Illinois, 1988

95. P. Meseguer and T. Walsh. Interleaved and discrepancy based search. In Proceedings of the 13th ECAI. European Conference on Artificial Intelligence, Wiley, 1998

96. S. Minton, M.D. Johnston, A.B. Philips, and P. Laird. Solving large-scale constraint satisfaction and scheduling problems using a heuristic repair method. In Proceedings of AAAI'90, pages 17-24, 1990

97. S. Minton, M.D. Johnston, A.B. Philips, and P. Laird. Minimizing Conflicts: A Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems. Artificial Intelligence, 52:161-205, 1992

98. D. Mitchell, B. Selman, and H. Levesque. Hard and Easy Distributions of SAT Problems. In Proceedings of the 10th National Conference on AI, pages 459-465. American Association for Artificial Intelligence, 1992

99. P. Morris. The Breakout method for escaping from local minima. In Proceedings of the 11th National Conference on AI. American Association for Artificial Intelligence, 1993

100. Jens Otten. On the Advantage of a Non-Clausal Davis-Putnam Procedure. Technical Report AIDA-97-01, FG Intellektik, FB Informatik, TH Darmstadt, January 1997

101. C.H. Papadimitriou and K. Steiglitz. Combinatorial optimization. Englewood Cliffs, NJ: Prentice-Hall Inc, 1982

102. C.H. Papadimitriou. On selecting a satisfying truth assigment. In Proceedings of the Conference on the Foundations of Computer Science, pages 163-169, 1991

103. C.H. Papadimitriou. Computational Complexity. Addison-Wesley, 1994

104. P. Pandurang Nayak and B.C. Williams. Fast context switching in real-time propositional reasoning. In Proceedings of the 14th National Conference on AI. American Association for Artificial Intelligence, 1997

105. A.J. Parkes and J.P. Walser. Tuning local search for satisfiability testing. In Proceedings of AAAI-96, pages 356-362, 1996

106. D. Pretolani. Satisfiability and hypergraphs, Ph.D. Thesis, Dipartimento di Informatica, Universita di Pisa, 1993

107. P. Prosser. Hybrid Algorithms for the Constraint Satisfaction Problem. Computational Intelligence 9(3):268-299, 1993

108. Paul Walton Purdom Jr. and Cynthia A. Brown. The pure literal rule and polynomial average time. SIAM Journal of Computing, 14(4):943--953, November 1985

109. R. Reiter and A. Mackworth. A logical framework for depiction and image interpretation. Artificial Intelligence, 42(2):125-155, 1989

110. R. Rodosek. A new approach on solving 3-Satisfiability. In Proceedings of the 3rd International Conference on Artificial Intelligence and Symbolic Mathematical Computation, pages 197-212. Springer, LNCS 1138, 1996

111. R. Schrag and J. M. Crawford. Implicates and Prime Implicates in Random 3SAT. Artificial Intelligence 81(1-2), 199-222, 1996

112. B. Selman and H. Kautz. Domain-independent extensions to GSAT: Solving large structured satisfiability problems. In Proceedings of the 13th IJCAI. International Joint Conference on Artificial Intelligence, 1993

113. B. Selman and H. Kautz. An empirical study of greedy local search for satisfiability testing. In Proceedings of the 11th National Conference on AI, pages 46-51. American Association for Artificial Intelligence, 1993

114. B. Selman, H. Kautz, and B. Cohen. Noise Strategies for Improving Local Search. In Proceedings of the 12th National Conference on AI, pages 337- 343. American Association for Artificial Intelligence, 1994

115. B. Selman and S. Kirkpatrick. Critical behaviour in the computational cost of satisfiability testing. Artificial Intelligence, 81:273-295, 1996

116. B. Selman, H. Levesque, and D. Mitchell. A New Method for Solving Hard Satisfiability Problems. In Proceedings of the 10th National Conference on AI, pages 440-446. American Association for Artificial Intelligence, 1992

117. J.P. Silva., K.A. Sakallah. Conflict Analysis in Search Algorithms for propositional satisfiability, Proceedings of the International Conference on Tools with Artificial Intelligence, November 1996

118. J. Slaney, M. Fujita, and M. Stickel. Automated reasoning and exhaustive search: quasigroup existence problems. Computers and Mathematics with Apllications, 29:115-132, 1995

119. Raymond M Smullyan. First Order Logic. Springer-Verlag, Berlin, 1968

120. W.M. Spears. A nn algorithm for boolean satisfiability problems. In Proceedings of the 1996 International Conference on Neural Networks, pages 1121-1126, 1996

121. W.M. Spears. Simulated annealing for hard satisfiability problems. In D.S. Johnson and M.A. Trick editors, In Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, pages 533-558. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Volume 26, American Mathematical Society, 1996

122. P. R. Stephan, R. K.Brayton, A. L. SangiovanniVincentelli. Combinational Test Generation Using satisfiability, Memorandum No. UCB/ERL M92/112, EECS Department, University of California at Berkeley, October 1992

123. T.E. Uribe and M.E. Stickel. Ordered binary decision diagrams and the Davis-Putnam procedure. In Proceedings of the First International Conference on Constraints in Computational Logics, Munich, Germany, pages 34-49, 1994

124. van Gelder and Y.K. Tsuji. Satisfiability testing with more reasoning and less guessing. In D.S. Johnson and M.A. Trick editors, In Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Volume 26, American Mathematical Society, 1996

125. J.P. Walser. Solving linear pseudo-boolean constraints with local search. In Proceedings of the 14th National Conference on AI, pages 269--274. American Association for Artificial Intelligence, 1997

126. B.C. Willians and P. Pandurang Nayak. A model-based approach to reactive self-configuring systems. In Proceedings of the 13th National Conference on AI. American Association for Artificial Intelligence, 1996

127. N. Yugami. Theoretical analysis of Davis-Putnam procedure and propositional satisfiability. In Proceedings of IJCAI-95, pages 282-288, 1995

128. H. Zhang. SATO: An efficient propositional prover. In Proceedings of 14th Conference on Automated Deduction, pages 272-275, 1997

129. H. Zhang. Specifying Latin squares in propositional logic. In R. Veroff, editor, Automated Reasoning and Its Applications, Essays in honor of Larry Wos, chapter 6. MIT Press, 1997

130. H. Zhang, M.P. Bonacina, and J. Hsiang. Psato: A distributed propositional prover and its application to quasigroup problems. Journal of Symbolic Computation, 11:1-18, 1996

131. H. Zhang and Stickel M. Implementing the Davis-Putnam Algorithm by Tries. Technical report, University of Iowa, 1994

132. H. Zhang and M.E. Stickel. An efficient algorithm for unit propagation. In Working Notes of the Fourth International Symposium on Artificial Intelligence and Mathematics, Fort Lauderdale, Florida, 1996

133. J. Zhang and H. Zhang. SEM: a System for Enumerating Models. In Proceedings of IJCAI-95, volume 1, pages 298-303. International Joint Conference on Artificial Intelligence, 1995

134. J. Zhang and H. Zhang. Combining local search and backtracking techniques for constraint Satisfaction. In Proceedings of 13th National Conference on Artificial Intelligence, pages 369-374. American Association for Artificial Intelligence, AAAI Press/The MIT Press, 1996

13 Simboli / Abbreviazioni

CSP	problema con soddisfacimento di vincoli (Constraint Satisfaction Problem)
FCC	controllo di consistenza in avanti (forward consistency checking)
GCP	problema di Graph Coloring (Graph Coloring Problem)
NP	classe di complessità NP (non deterministica, tempo polinomiale)
N	numeri naturali incluso zero
p, p_i	probabilità
P	classe di complessità P (deterministica, tempo polinomiale)
R	numeri reali
R^+	numeri reali positivi
SAT	problema della soddisfacibilità proporzionale (Propositional Satisfiability Problem)
SLS	ricerca locale stocastica (stochastic local search)
TSP	problema del commesso viaggiatore (the Travelling Salesperson Problem)
$UNSAT$	problema della non soddisfacibilità proporzionale (Propositional Unsatisfiability Problem - il complemento del SAT)
VAL	Propositional Validity Problem