Encodings in SAT and Constraints

Ian Gent
University of St Andrews

Topics in this Series

- Why SAT & Constraints?
- SAT basics
- Constraints basics
- Encodings between SAT and Constraints
- Watched Literals in SAT and Constraints
- Learning in SAT and Constraints
- Lazy Clause Generation + SAT Modulo Theories

Encodings SAT & CP

- Maybe most obvious link SAT to CP
- Works outside solvers
- More interesting than you might think
- Propagation-optimal encodings
  - Examples CP to SAT
  - Example SAT to CP
  - Fundamental Conjecture of Reformulation
  - Why it’s false!

Encodings: Motivation

- Entire basis of NP-completeness is encoding
  - translate one problem to another
  - in reasonable (poly) time
  - and faithfully - solution preserving
- SAT is the first NP-complete problem
- So Why Not ...
  - just translate everything into SAT
  - and use a SAT solver?

Encodings: Motivation

- Not a straw man argument
- There are real advantages to using SAT (or CP) as basis, and then encoding to it
- We only need to write one solver
    - which can then be highly optimised
- It's typically easier to write translator than new solver
- Every time we optimise SAT (or CP)
    - we optimise every other NP-complete problem

Encodings: Motivation

- But it's not as simple as that ...
- We can't really afford to lose propagation
  - E.g. if we need to establish AC
  - then our encoded problem should do AC
  - using standard SAT techniques
- We can't really afford to lose time
  - E.g. we can establish AC in $O(ed^2)$
    - So it has to be this if we encode to SAT
      - ... and then use standard encoding
- Leads to idea of "propagation-optimal" encodings

Propagation Optimal Encodings

- Encoded version might not propagate as well
- Propagation in encoded version might be slow
  - if we lose O(n) time at each node, translation will never be competitive
- **Propagation Optimal Encoding**
  - translation time should be optimal for target consistency level
  - native propagation (e.g. unit prop.) on encoding should achieve target consistency level
  - and do it in optimal time for target consistency level

Encoding CSP to SAT

- Going to start with binary CSPs
  - but ideas do generalise
- Focus on two key encodings
  - Direct Encoding
    - folklore, Walsh 2000
    - Acts like Forward Checking
  - Support Encoding
    - Kasif 1990, Gent 2002
    - Acts like AC

Encoding CSPs into SAT

- e.g. CSP variable A domain size 3
  - SAT variables a1, a2, a3
  - a1=T ⇔ A=1
- “at-least-one” clause
  - a1 OR a2 OR a3
- “at-most-one” clauses
  - -a1 OR –a2
  - -a2 OR –a3
  - -a3 OR –a1

Conflict Clauses

A < B	A=1	A=2	A=3
B=1	✗	✗	✗
B=2	✓	✗	✗
B=3	✓	✓	✗

One conflict clause for each ✗

Conflict Clauses

A < B	A=1	A=2	A=3
B=1	-a1 OR –b1	✕	✕
B=2	✓	✕	✕
B=3	✓	✓	✕

If A = 1 then B ≠ 1.

Conflict Clauses

A < B	A=1	A=2	A=3
B=1	-a1 OR –b1	-a2 OR –b1	✗
B=2	✓	✗	✗
B=3	✓	✓	✗

If A = 2 then B $\neq$ 1.

Conflict Clauses

A < B	A=1	A=2	A=3
B=1	-a1 OR –b1	-a2 OR –b1	-a3 OR –b1
B=2	✓	-a2 OR –b2	-a3 OR –b2
B=3	✓	✓	-a3 OR –b3

Support Clauses

A < B	A=1	A=2	A=3
B=1	✗	✗	✗
B=2	✓	✗	✗
B=3	✓	✓	✗

One “support” clause for each row/column

Support Clauses

A < B	A=1	A=2	A=3	
B=1	✗	✗	✗	-b1
B=2	✓	✗	✗	
B=3	✓	✓	✗	

B=1 is impossible as no value of A supports it

Support Clauses

A < B	A=1	A=2	A=3	
B=1	✗	✗	✗	-b1
B=2	✓	✗	✗	a1 OR -b2
B=3	✓	✓	✗	

If A≠1, then there is no support for B=2

Support Clauses

A < B	A=1	A=2	A=3	
B=1	✗	✗	✗	-b1
B=2	✓	✗	✗	a1 OR -b2
B=3	✓	✓	✗	a1 OR a2 OR -b3

If A≠1 and A≠2, then there is no support for B=3

Support Clauses

A < B	A=1	A=2	A=3	
B=1	✕	✕	✕	-b1
B=2	✓	✕	✕	a1 OR -b2
B=3	✓	✓	✕	a1 OR a2 OR –b3
	b2 OR b3 OR -a1	b3 OR –a2	-a3	

Direct & Support Encodings

- "Direct Encoding" is most commonly used
  - almost folklore but see e.g. [Walsh, CP 2000]
  - at-least-one clauses
  - *at-most-one clauses optional*
  - conflict clauses
- "Support Encoding" [Gent, ECAI 2002]
  - at-least-one clauses
  - at-most-one clauses (not optional)
  - support clauses [Kasif, AIJ 1990]

Theoretical Comparison

- Compare CSP algorithms FC & MAC
  - FC = Forward Checking
  - MAC = Maintaining Arc Consistency
- With (simple) DPLL running on encoded versions
  - unit propagates between nodes
- Results on Direct Encoding
  - DPLL on Direct performs equivalent search to FC
    - [Genisson & Jegou ECAI 94]
  - MAC can outperform DPLL on Direct encoding
    - [Walsh CP 2000]

Arc Consistency in SAT

- Natural correspondence in the support encoding
  - $a1=T \Leftrightarrow A=1$
  - $a1=F \Leftrightarrow 1 \notin \text{domain}(A)$
  - $a1=\{T,F\} \Leftrightarrow 1 \in \text{domain}(A)$
- Key result on Support Encoding
  - When unit propagation terminates without failure, the SAT variables correspond to Arc Consistent domains in the CSP
- Simple Corollary
  - DPLL on Support Encoding = MAC on CSP

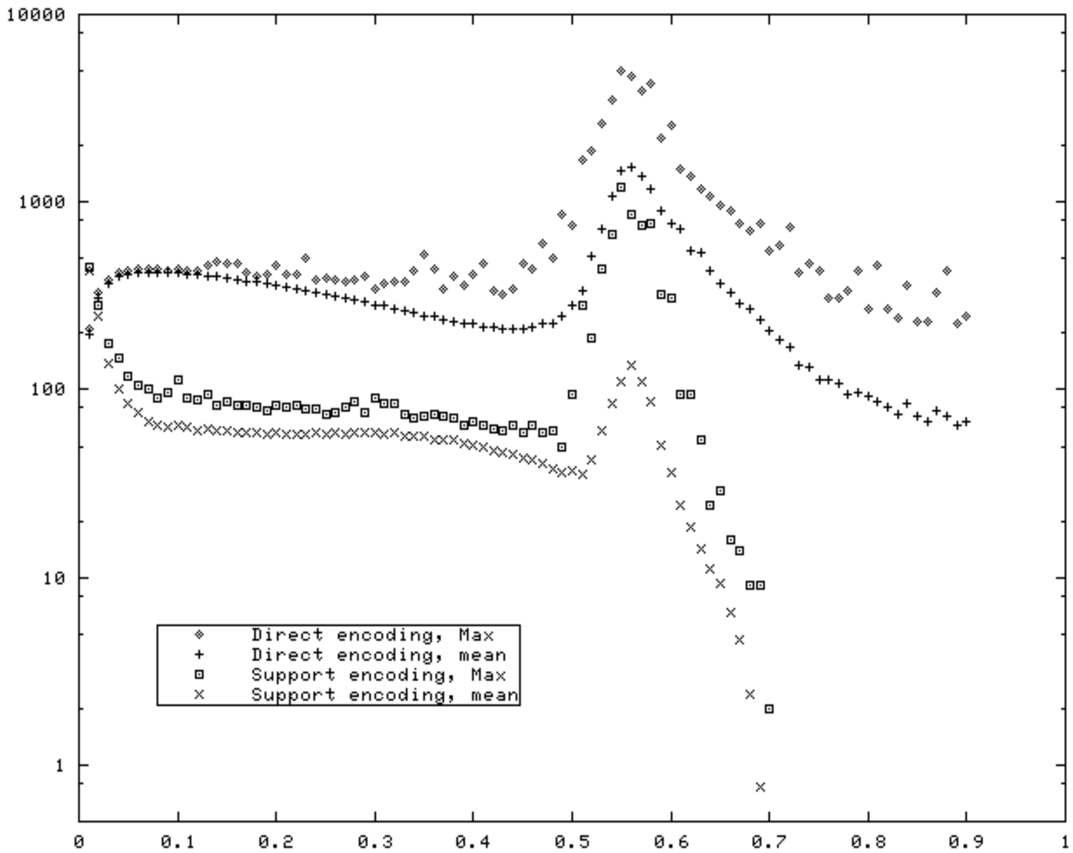
Support Encoding is AC-Optimal

- For a CSP with e constraints, domain size d
  - unit propagation takes time $O(ed^2)$
    - *including* translation time
  - this is optimal worst case time for AC
    - in fact maybe the second optimal algorithm for AC [Kasif 90]
- So translation to SAT & use of DPLL
  - is equivalent to MAC
  - is optimal time algorithm for MAC
  - benefits from any other techniques used in SAT
    - e.g. clause learning key in Chaff

Experimental Comparison

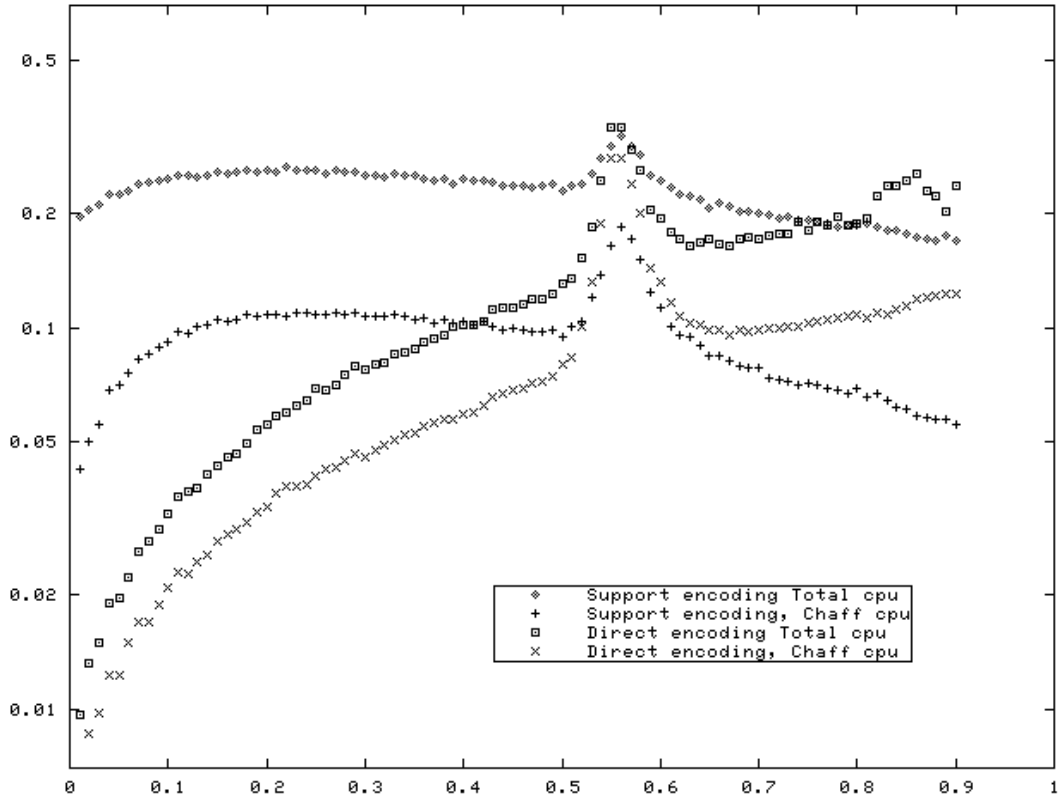
- Implemented translation in Common Lisp
- Used Chaff on translated instances
- Tested on hard random binary CSP’s
- At peak difficulty, about 5-6 times slower than MAC2001 [Bessière/Regin IJCAI 2001]

DPLL for Support vs Direct:



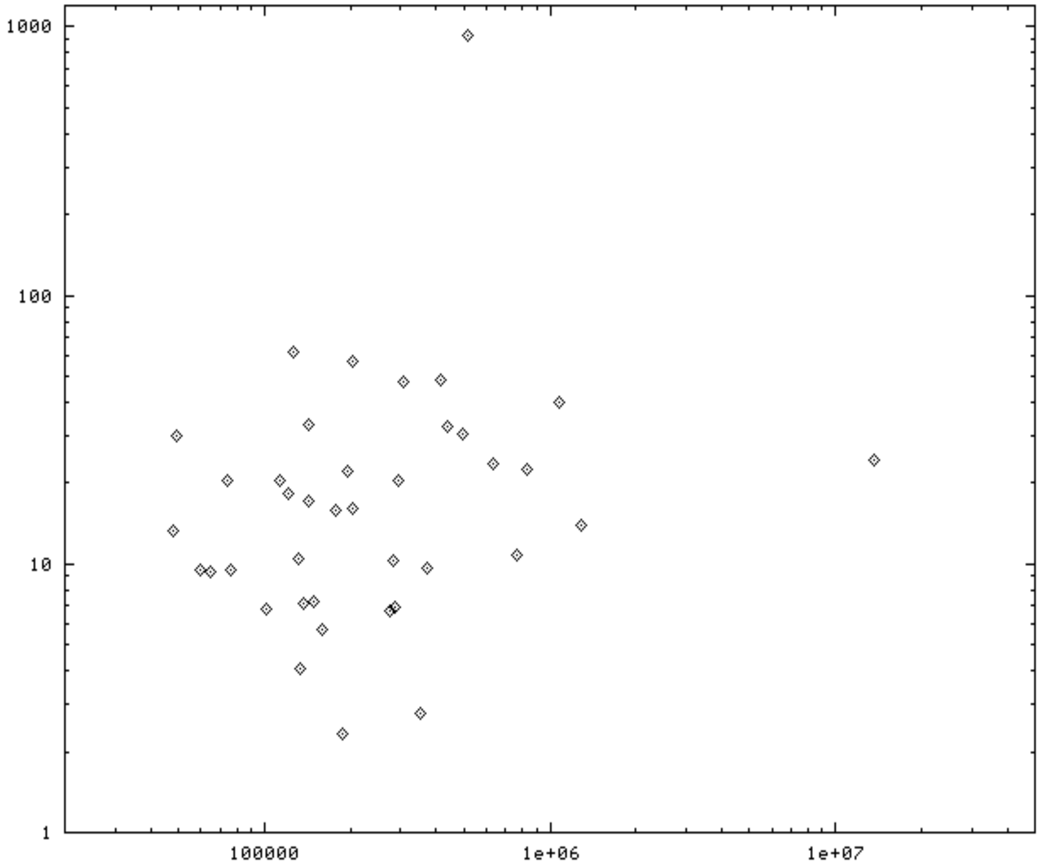
- Chaff used as DPLL solver
- N=50
- x axis is constraint tightness, p2
- y axis is nodes searched
- Support always searches less
- Support max is less than direct mean
- Zero search for p2>0.7

DPLL for Support vs Direct:



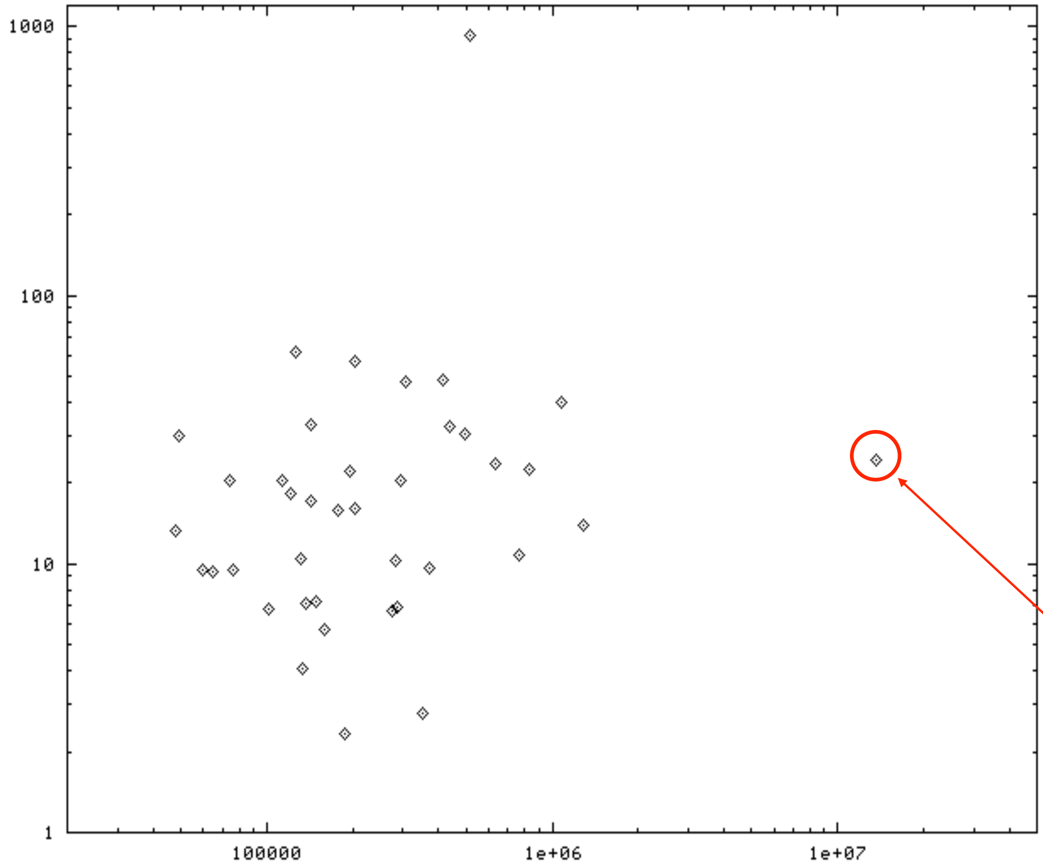
- same data as previous slide
- y axis is mean cpu time
- top line includes translation time
- bottom line just chaff time
- Support encoding usually slower
- Support just faster at peak of hardness
- At N=100, support encoding about 3x faster at peak

WalkSAT for Support vs Direct:



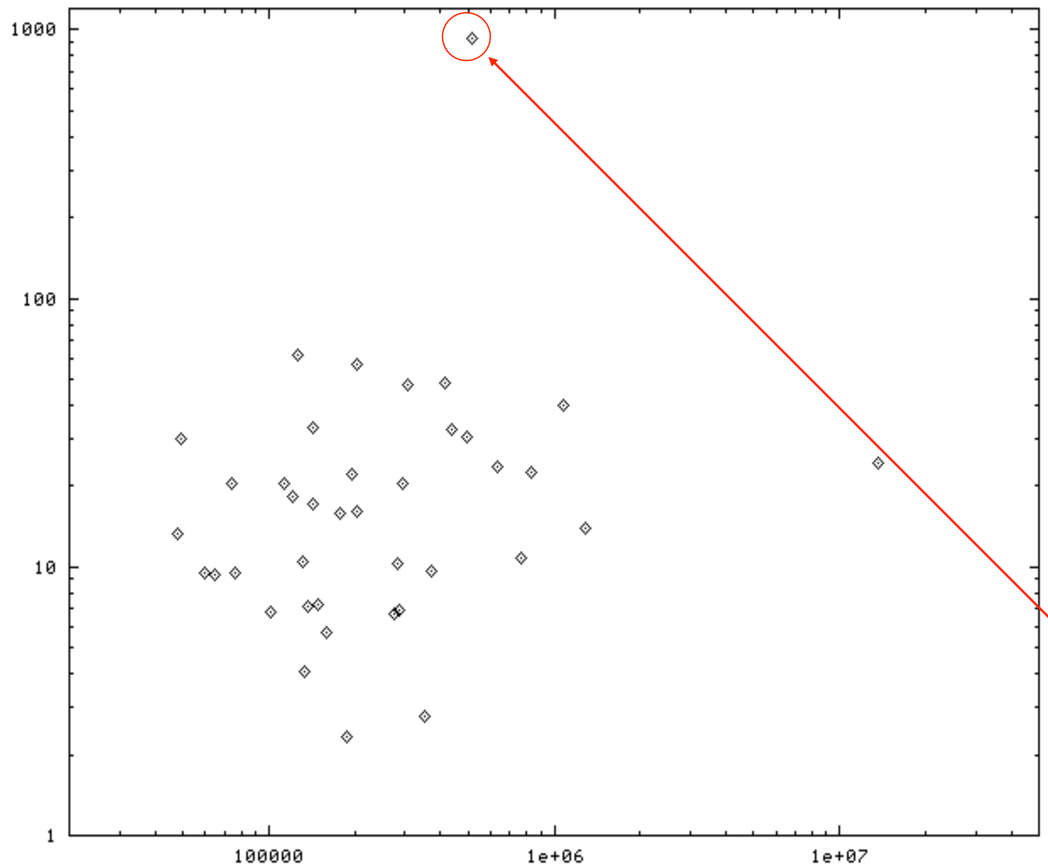
- Hoos's Novelty+ variant
- each point one instance
- x axis is #flips for support encoding
- y axis is flips-speedup of support vs direct encoding
- Umm, got that yet?

WalkSAT for Support vs Direct:



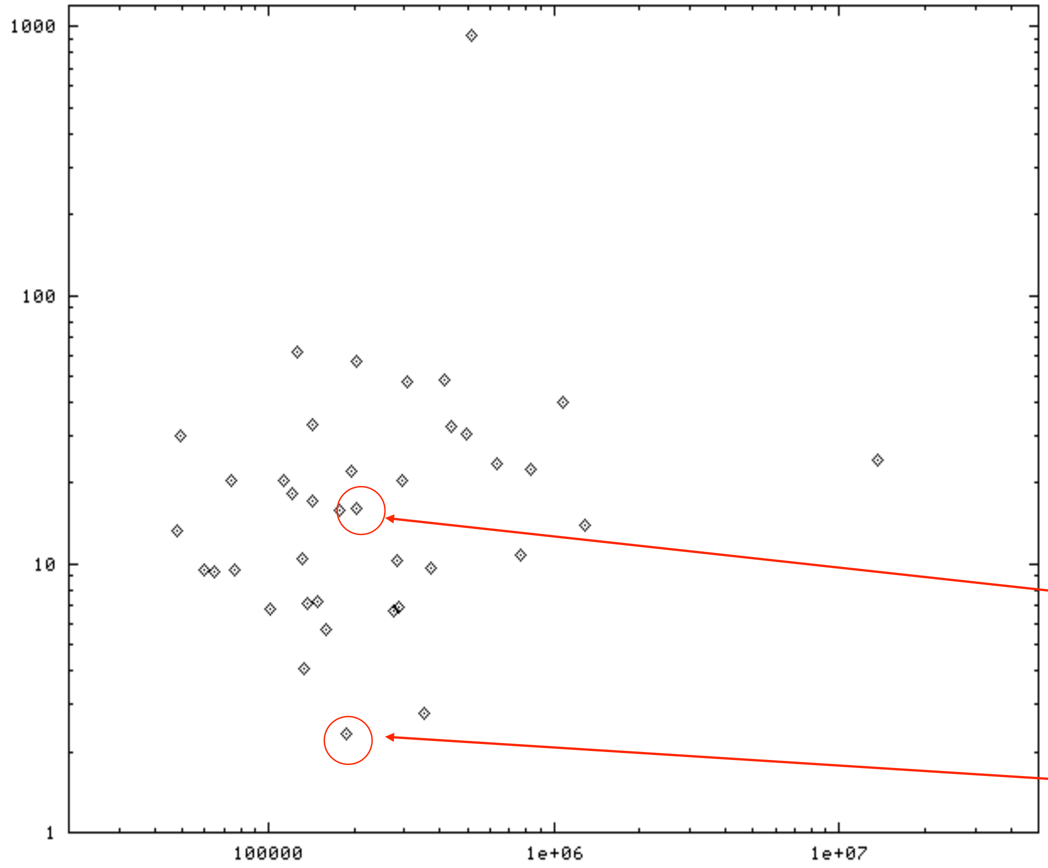
- Hoos's Novelty+ variant
- each point one instance
- x axis is #flips for support encoding
- y axis is flips-speedup of support vs direct encoding
- This instance took about 10,000,000 flips for support encoding, but 20 x more in the direct encoding

WalkSAT for Support vs Direct:



- Hoos's Novelty+ variant
- each point one instance
- x axis is #flips for support encoding
- y axis is flips-speedup of support vs direct encoding
- This instance took about 500,000 flips for support encoding, but 922 x more in the direct encoding

WalkSAT for Support vs Direct:



- Hoos's Novelty+ variant
- each point one instance
- x axis is #flips for support encoding
- y axis is flips-speedup of support vs direct encoding
- The median was 16 x more flips using the direct encoding
- The best the direct encoding could do was 2.34 x more flips

Optimal Encodings: Pluses and Minuses

Pluses

+ Just need to implement a translation
+ Take advantage of state of the art SAT solvers …
+ … and future developments
+ Can be competitive with direct CSP solvers

Minuses

- Space complexity is worse
- Hits worst case time complexity in average case
- Direct implementation should always be faster

Support Encoding

- Generalised to non binary constraints
  - with similar propagation-optimality
  - meaning we can search arbitrary constraint problems using GAC
  - Bessiere, Hebrard, Walsh 2002
- Investigated further on local search
  - with mixed results
  - Prestwich 2004
  - Interesting further ideas
  - Introducing as many solutions as possible
    - while preserving correctness of course

So that’s that!

- Encodings are great
  - Ok there are some minuses
- But we’ve got an ideal solution
  - we can propagate any constraint
  - in optimal time
  - using only simplish encodings + SAT solvers
  - so what’s the problem?

“Space complexity is worse”

- Forgot one little word...

“Space complexity is **exponentially** worse”

- Forgot one BIG word...

Exponentially worse?

- Well, not in the case of AC
- But in the case of GAC
- Remember I said ...
  - we almost never list all tuples in constraints?
- Well we have to in support encodings
  - all allowed tuples
- Or in direct encodings
  - all disallowed tuples
- Which can be exponentially bigger than an implicit representation
  - e.g. all different has *n!* allowed tuples and *far more* disallowed

Ok Forget It

- So there’s a cute encoding for AC in SAT
- But we can’t do well in general
- So encodings are useless, right?

Find smarter encodings

- Give up on the idea of one true encoding
  - Just like there's no single key constraint
- Have an army of encodings
  - One for every constraint we want
  - Maybe propagation optimal for that
- Steal ideas from propagation algorithms?
  - E.g. GAC-Lex

Inspiration

- We present an encoding of GACLex
  - I'll tell you what that is in a minute
- The encoding was inspired by an algorithm for maintaining GACLex
- Initial algorithm proposed by Miguel/Frisch/Walsh
- Later variants and study presented in
  - Global Constraints for Lexicographic Orderings
  - Frisch, Hnich, Kiziltan, Miguel, Walsh, 2002 [CP], 2006 [AIJ]

Lexicographic Constraint

- Arrays A/B of variables
- A $\leq$ B if
  - A[1] < B[1]
  - A[1] = B[1] & A[2] < B[2]
  - ...
  - A[I]=B[I] for all I
- Application in symmetry
  - A/B indistinguishable
  - A $\leq$ B breaks symmetry

Lexicographic Constraint

- Arrays A/B of variables
- A ≤ B if
  - A[1] < B[1]
  - A[1] = B[1] & A[2] < B[2]
  - ...
  - A[I]=B[I] for all I
- Application in symmetry
  - A/B indistinguishable
  - A ≤ B breaks symmetry

A	B
0	0
0	1
1	0
1	1
0	0
...	...

GAC: Generalised Arc Consistency

- A ≤ B is GAC if
  - any value A[I] is allowed by some setting of the values of other A/B vars
  - similarly for B[I]
- If A ≤ B is not GAC
  - we can establish GAC

A	B
0	0
1	1
*	*
0	0
1	0
...	...

GAC: Generalised Arc Consistency

- $A \leq B$ is GAC if
  - any value A[I] is allowed by some setting of the values of other A/B vars
  - similarly for B[I]
- If $A \leq B$ is not GAC
  - we can establish GAC
- E.g. A[3] = 1 is not possible, as then A > B
- Similarly B[3] = 0

A	B
0	0
1	1
*	*
0	0
1	0
...	...

GAC: Generalised Arc Consistency

- $A \leq B$ is GAC if
  - any value A[I] is allowed by some setting of the values of other A/B vars
  - similarly for B[I]
- If $A \leq B$ is not GAC
  - we can establish GAC
- Establish GAC by setting A[3] = 0, B[3]=1

A	B
0	0
1	1
0	1
0	0
1	0
...	...

GAC: Generalised Arc Consistency

- GAC Lex can be established in O(n) time for binary domains
  - Frisch et al, CP 2002
  - specialised algorithm
- We encode GAC Lex using new constraints

A	B
0	0
1	1
0	1
0	0
1	0
...	...

Encoding GAC Lex

- Assume that A/B indexed from 1
- Introduce new array a[] indexed from 0
  - two values of each a[I]
- Meaning of a[]
  - a[I] = 1 $\Leftrightarrow$ A[1]=B[1], ... A[I]=B[I]
  - a[I] = 0 $\Leftrightarrow$ A $\leq$ B guaranteed by A[1..I], B[1..I]
- Add O(n) constraints linking A/B/a[]

5 Constraints for GAC Lex

1) a[0]=1

- Presentational convenience
- Allows uniform presentation of remaining constraints

5 Constraints for GAC Lex

1) $a[0]=1$
2) $a[I]=0 \rightarrow a[I+1]=0$

- $0 \leq I \leq n-1$
- Monotonicity
- If GAC Lex guaranteed by 1..I, it is guaranteed by 1..I+1

5 Constraints for GAC Lex

1) $a[0]=1$
2) $a[I]=0 \Rightarrow a[I+1]=0$
3) $a[I]=1 \Rightarrow A[I]=B[I]$

- $0 \leq I \leq n-1$
- Equality
- Monotonicity implies each $a[J]=1$ for $J \leq I$
- 3) gives $A[J] = B[J]$ for sequence up to I
- Gives intended meaning to $a[I]=1$

5 Constraints for GAC Lex

1) $a[0]=1$
2) $a[I]=0 \rightarrow a[I+1]=0$
3) $a[I]=1 \rightarrow A[I]=B[I]$
4) $a[I]=1$ & $a[I+1]=0 \rightarrow A[I+1] < B[I+1]$

- $0 \leq I \leq n-1$
- Inequality
- $a[I+1]=0$ means we want to guarantee $A<B$ from $1..I$
- But $a[I]=1$ means we have $A[1..I]=B[1..I]$
- So we must set $A[I+1] < B[I+1]$

5 Constraints for GAC Lex

1) $a[0]=1$
2) $a[I]=0 \rightarrow a[I+1]=0$
3) $a[I]=1 \rightarrow A[I]=B[I]$
4) $a[I]=1$ & $a[I+1]=0 \rightarrow A[I+1] < B[I+1]$
5) $a[I]=1 \rightarrow A[I+1] \leq B[I+1]$

- $0 \leq I \leq n-1$
- Redundant constraint
- Implied by (2) & (3)
- But not deduced by AC
- 5) included so that AC can do implication
  - In fact only needed for domain size > 2

5 Constraints for GAC Lex

1) a[0]=1
2) a[I]=0 → a[I+1]=0
3) a[I]=1 → A[I]=B[I]
4) a[I]=1 & a[I+1] =0 → A[I+1] < B[I+1]
5) a[I]=1 → A[I+1] ≤ B[I+1]

a	A	B
*		
*	0	0
*	1	1
*	*	*
*	0	0
*	1	0
…	…	…

5 Constraints for GAC Lex

1) a[0]=1
2) a[I]=0 ➔ a[I+1]=0
3) a[I]=1 ➔ A[I]=B[I]
4) a[I]=1 & a[I+1] =0 ➔ A[I+1] < B[I+1]
5) a[I]=1 ➔ A[I+1] ≤ B[I +1]

a	A	B
1		
*	0	0
*	1	1
*	*	*
*	0	0
*	1	0
...	...	...

5 Constraints for GAC Lex

1) a[0]=1
2) a[I]=0 ➔ a[I+1]=0
3) a[I]=1 ➔ A[I]=B[I]
4) a[I]=1 & a[I+1] =0 ➔ A[I+1] < B[I+1]
5) a[I]=1 ➔ A[I+1] ≤ B[I +1]

a	A	B
1		
1	0	0
*	1	1
*	*	*
*	0	0
*	1	0
…	…	…

5 Constraints for GAC Lex

1) a[0]=1
2) a[I]=0 ➔ a[I+1]=0
3) a[I]=1 ➔ A[I]=B[I]
4) a[I]=1 & a[I+1] =0 ➔ A[I+1] < B[I+1]
5) a[I]=1 ➔ A[I+1] ≤ B[I +1]

a	A	B
1		
1	0	0
1	1	1
*	*	*
*	0	0
*	1	0
…	…	…

5 Constraints for GAC Lex

1) a[0]=1
2) a[I]=0 ➔ a[I+1]=0
3) a[I]=1 ➔ A[I]=B[I]
4) a[I]=1 & a[I+1] =0 ➔ A[I+1] < B[I+1]
5) a[I]=1 ➔ A[I+1] ≤ B[I+1]

a	A	B
1		
1	0	0
1	1	1
*	*	*
*	0	0
0	1	0
…	…	…

5 Constraints for GAC Lex

1) a[0]=1
2) a[I]=0 ➔ a[I+1]=0
3) a[I]=1 ➔ A[I]=B[I]
4) a[I]=1 & a[I+1] =0 ➔ A[I+1] < B[I+1]
5) a[I]=1 ➔ A[I+1] ≤ B[I +1]

a	A	B
1		
1	0	0
1	1	1
*	*	*
0	0	0
0	1	0
...	...	...

5 Constraints for GAC Lex

1) a[0]=1
2) a[I]=0 ➔ a[I+1]=0
3) a[I]=1 ➔ A[I]=B[I]
4) a[I]=1 & a[I+1]=0 ➔ A[I+1] < B[I+1]
5) a[I]=1 ➔ A[I+1] ≤ B[I+1]

a	A	B
1		
1	0	0
1	1	1
0	*	*
0	0	0
0	1	0
...	...	...

5 Constraints for GAC Lex

1) a[0]=1
2) a[I]=0 ➔ a[I+1]=0
3) a[I]=1 ➔ A[I]=B[I]
4) a[I]=1 & a[I+1] =0 ➔ A[I+1] < B[I+1]
5) a[I]=1 ➔ A[I+1] ≤ B[I+1]

a	A	B
1		
1	0	0
1	1	1
0	0	1
0	0	0
0	1	0
…	…	…

For 0/1 domains, x<y ⇔x=0, y=1

Theoretical Analysis

- Arc Consistency (AC) establishes GAC Lex
  - Specifically:
    - A/B GAC Lex in any AC state of A/B/a[] variables
- Time Complexity in Boolean Domains
  - AC takes O(n) time
  - Encoding+AC = optimal algorithm for GAC
  - This is a “propagation-optimal” encoding
-

Stable Marriage

- Stable Marriage problem
  - Assume every female has a preference list of all males
  - And vice versa
  - And there are *n* males and *n* females
- Find a **stable** matching of females to males
  - There is no pair Ann & Andy, not married
  - Where Ann prefers Andy to her husband
  - And Andy prefers Ann to his wife
  - i.e. Ann & Andy would elope with each other
  -

Stable Marriage

- Gale-Shapley algorithm is low polynomial time
  - Inspired SAT encoding
  - Which achieves AC in same poly time
  - And solutions can be read off from AC domains
  - Gent, Irving, Manlove, Prosser, Smith 2001

SAT to Constraints

- Don't need to encode SAT to Constraints?
- We do if we want propagation-optimal
- At first sight looks hard/impossible
  - assuming we use AC propagation
- Taking boolean domains to n-ary
- And AC per constraint is O(d^2)
  - Maybe we'll lose O($d/2$) = $O(d)$ or something
- But there is a propagation optimal encoding

Extended Literal Encoding

- Based on the "literal encoding"
  - Bennaceur 1996
- But extension makes it propagation-optimal
  - Gent, Prosser, Walsh, 2003
- Also called "Place Encoding"
  - Jarvisalo & Niemela, 2004

Literal Encoding

- For each *k*-clause C in the SAT problem
  - Variable x_C in CSP encoding
  - Domain of x_C is {1..k}
- The meaning of x_C = i
  - is that the i^{th} literal of clause C is satisfied
  - from which we can read solution of SAT problem
- For every pair of clauses C_1, C_2
  - If there are any ...
    - Add constraint ruling out complementary literals

Literal Encoding Example

C_1: a OR b OR c

C_2: -a OR -b OR c

C1/C2	$x_2=1$	$x_2=2$	$x_2=3$
$x_1=1$	x	✓	✓
$x_1=2$	✓	x	✓
$x_1=3$	✓	✓	✓

Literal Encoding Problem

- can do unit propagation, but ...
  - u.p. should take O(*mk*)
- But each constraint O(k^2)

c1/c2	x_2=1	x_2=2	x_2=3
x_1=1	x	✓	✓
x_1=2	✓	x	✓
x_1=3	✓	✓	✓

Literal Encoding Problem

- can do unit propagation, but ...
  - u.p. should take $O(mk)$
- But each constraint $O(k^2)$
- And there can be $O(m^2)$
  - because var might occur
    - m/2 times positively
    - m/2 times negatively
- So this is $O(m^2k^2)$

c1/c2	$x_2=1$	$x_2=2$	$x_2=3$
$x_1=1$	x	✓	✓
$x_1=2$	✓	x	✓
$x_1=3$	✓	✓	✓

Extended Literal Encoding

- As before:
  - For each *k*-clause C in the SAT problem
    - Variable x_C in CSP encoding
    - Domain of x_C is {1..k}
- Extension
  - Reintroduce original boolean variables
  - Domain {0,1}
- Constraints between booleans and clause vars
  - none between clause vars and other clause vars

Extended Literal Encoding Example

C_1: a OR b OR c

C_2: -a OR -b OR c

a/C1	x_1=1	x_1=2	x_1=3
a=0	x	✓	✓
a=1	✓	✓	✓

Extended Literal Encoding Example

C_1: a OR b OR c

C_2: -a OR -b OR c

c/C1	$x_1=1$	$x_1=2$	$x_1=3$
c=0	✓	✓	x
c=1	✓	✓	✓

Extended Literal Encoding Example

C_1: a OR b OR c

C_2: -a OR -b OR c

a/C2	$x_1=1$	$x_1=2$	$x_1=3$
c=0	✓	✓	✓
c=1	x	✓	✓

Extended Literal Encoding Example

C_1: a OR b OR c

C_2: -a OR -b OR c

a/C2	x_2=1	x_2=2	x_2=3
a=0	✓	✓	✓
a=1	x	✓	✓

Extended Literal Encoding Example

C_1: a OR b OR c

C_2: -a OR -b OR c

b/C2	x_2=1	x_2=2	x_2=3
b=0	✓	✓	✓
b=1	✓	x	✓

Extended Literal Encoding Example

C_1: a OR b OR c

C_2: -a OR -b OR c

c/C2	$x_1=1$	$x_1=2$	$x_1=3$
c=0	✓	✓	x
c=1	✓	✓	✓

Extended Literal Complexity

- This example looks worse
  - 6 constraints/36 cells
  - compared to 1 constraint/18 cells
- But asymptotics are better
- We have O(mk) constraints
  - One for each literal in each clause
  - Each propagates in time O($2k$) = $O(k)$
  - Total O(mk^2) propagation time

Extended Literal Complexity

- This example looks worse
  - 6 constraints/36 cells
  - compared to 1 constraint/18 cells
- But asymptotics are better
- We have O(*mk*) constraints
  - One for each literal in each clause
  - Each propagates in time $O(2k) = O(k)$
  - Total $O(mk^2)$ propagation time

Extended Literal Complexity

- This is propagation optimal if we fix k
  - Plus there is easy propagation optimal encoding k-SAT to 3-SAT
  - e.g. a OR b OR c OR d becomes
    - a OR b OR z
    - -z OR c OR d
- So we have propagation optimal encoding of k-SAT to CSP

Is the Extended Literal Encoding worthwhile?

- Not really
- Why not?

Why not?

- Hard to see the advantages of translating SAT to CSP in general
- It's unlikely that the translated version will propagate as fast in practice as in SAT
  - which is true from CSP to SAT too but ...
- Also harder to see advantages we get
  - CP solvers good at propagating multiple different types of constraints together
  - And writing specialised propagators, for (e.g.) clauses
  - If we're going to only propagate one type of constraint, why not build a (SAT) solver to do it?
- Overall, encodings SAT to CP have attracted little interest

Fundamental Conjecture of Reformulation

- In early 2000s, work such as above on AC, GACLex, SATtoCP, StableMarriage, ...
- Led me to suggest the
- “Fundamental Conjecture of Reformulation”

Fundamental Conjecture of Reformulation

- This says that ...
- For any [reasonable] constraint propagator taking time *p(n)* (for some polynomial p)
  - not saying what reasonable is
- There is an encoding of the constraint so that a standard AC algorithm can do the same work as the propagator in time *p(n)*, including translation time
- Since we have optimal encodings both ways, AC can be interchanged SAT

Fundamental Conjecture of Reformulation

- If true, ...
- There would be a strong argument that encodings should become key focus of SAT/CP research
- Including techniques to beat some of the disadvantages
  - e.g. hitting worst case space complexity

Encoding All-Different

- It was always obvious that All-Different would be an acid test of the conjecture
  - Key constraint
  - Very good GAC algorithm (Regin)
    - flow based
    - beats “obvious” encoding easily

Encoding All-Different

- Fundamental conjecture *fails* the acid test
- I.e. it's false
- Key result
  - Bessiere, Katsirelos, Narodytska, Walsh, 09
- It is *impossible* to encode all-different to SAT
  - in a polynomial sized number of clauses
  - and obtain GAC

Impossibility Result

- Result based on *circuit complexity*
- Encoding constraint *c* to into SAT
  - gives a SAT checker [ie. there being a solution tuple to *c*]
  - gives monotone circuit of poly size
- So if there's no monotone circuit of poly size
  - there's no encoding of *c* into SAT

Impossibility Result

- Perfect matching has *no* monotone circuit of poly size
  - Rasborov 85, Tardos 88
- All-Different subsumes perfect-matching
- But we already had ...
  - *So if there's no monotone circuit of poly size*
    - *there's no encoding of c into SAT*
- Proof by contradiction:
  - We are done.
- There is no propagation-optimal encoding of AllDifferent into SAT (or generic AC)

Encodings Summary

- More to encodings than you might think
- Attractive, fun and interesting area
- And valuable....
  - increase power of SAT solving especially
- But still some problems
  - Can’t expect to beat native implementation
  - Can have space complexity problems
  - Can hit worst case *all the time*