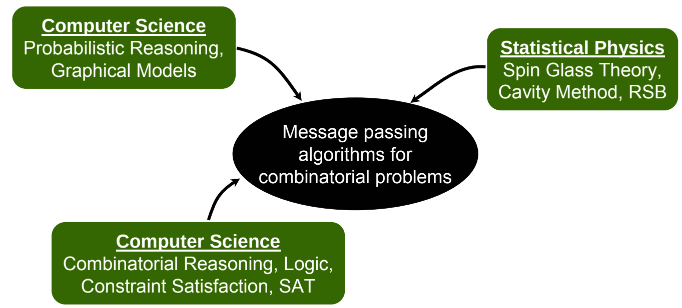
Satisfied by Message Passing:
Probabilistic Techniques for Combinatorial Problems

Lukas Kroc, Ashish Sabharwal, Bart Selman
Cornell University

AAAI-08 Tutorial
July 13, 2008

What is the Tutorial all about?

How can we use ideas from probabilistic reasoning and statistical physics to solve hard, discrete, combinatorial problems?



Why the Tutorial?

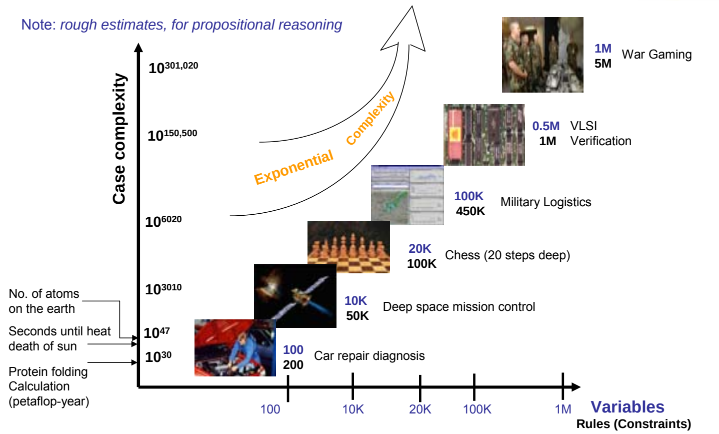
- A very active, multi-disciplinary research area
  - Involves amazing statistical physicists who have been solving a central problem in CS and AI: *constraint satisfaction*
  - They have brought in unusual techniques (unusual from the CS view) to solve certain hard problems with unprecedented efficiency
  - Unfortunately, can be hard to follow: *they speak a different language*
- Success story
  - Survey Propagation (SP) can solve 1,000,000+ variable problems in a few minutes on a desktop computer (demo later)
    - The best “pure CS” techniques scale to only 100s to ~1,000s of variables
  - Beautiful insights into the **structure of the space of solutions**
  - Ways of *using the structure* for faster solutions
- Our turf, after all ☺ It's time we bring in the CS expertise…

Combinatorial Problems

Credit: W.-J. van Hoeve

Exponential Complexity Growth: The Challenge of Complex Domains

Note: *rough estimates, for propositional reasoning*



[Credit: Kumar, DARPA; cited in Computer World magazine]

Tutorial Outline

1. **Introduction**
2. **Probabilistic inference using message passing**
3. **Survey Propagation**
4. **Solution clusters**
5. **Probabilistic inference for clusters**
6. **Advanced topics**

Tutorial Outline

1. **<u>Introduction</u>**
2. **Probabilistic inference using message passing**
3. **Survey Propagation**
4. **Solution clusters**
5. **Probabilistic inference for clusters**
6. **Advanced topics**

- Constraint satisfaction problems (CSPs)
  - SAT
  - Graph problems
- Random ensembles and satisfiability threshold
- Traditional approaches:
  - DPLL
  - Local search
- Probabilistic approaches:
  - Decimation
  - (Reinforcement)

Constraint Satisfaction Problem (CSP)

- Constraint Satisfaction Problem **P**:
  Input:
  a set V of **variables**
  a set of corresponding **domains** of variable values [discrete, finite]
  a set of **constraints** on V [constraint ≡ set of allowed tuples of values]
  Output:
  a solution, i.e., an assignment of values to variables in V such that all constraints are satisfied

- *Each individual constraint often involves a small number of variables*
  - Important for efficiency of message passing algorithms like Belief Propagation
  - Will need to compute sums over all possible values of the variables involved in a constraint: exponential in the number of variables appearing in the constraint

Boolean Satisfiability Problem (SAT)

- **SAT**: a special kind of CSP
  - Domains: {0,1} or {true, false}
  - Constraints: logical combinations of subsets of variables
- **CNF-SAT**: further specialization (a.k.a. SAT)
  - Constraints: disjunctions of variables or their negations (“clauses”)

  $\Rightarrow$ Conjunctive Normal Form (CNF) : a conjunction of clauses
- **k-SAT**: the specialization we will work with
  - Constraints: clauses with exactly k variables each

$$F = \underbrace{(\neg x \vee y \vee z)}_{\alpha} \wedge \underbrace{(x \vee \neg y \vee z)}_{\beta} \wedge \underbrace{(x \vee y \vee \neg z)}_{\gamma}$$

SAT Solvers: Practical Reasoning Tools

Tremendous improvement in the last 15 years:
Can solve much larger and much more complex problems

From academically interesting to practically relevant

Regular SAT Competitions (industrial, crafted, and random benchmarks)
and SAT Races (focus on industrial benchmarks)
[Germany ’89, Dimacs ’93, China ’96, SAT-02, SAT-03, …, SAT-07, SAT-08]

E.g. at SAT-2006:

- 35+ solvers submitted, most of them open source
- 500+ industrial benchmarks
- 50,000+ benchmark instances available on the www

This constant improvement in SAT solvers is the key to making technologies such as SAT-based planning very successful

Automated Reasoning Tools

Many successful fully automated discrete methods are based on SAT

- Problems modeled as rules / constraints over Boolean variables
- “SAT solver” used as the inference engine

Applications: *single-agent search*

- AI planning

  SATPLAN-06, **fastest step-optimal planner**;
  ICAPS-06 competition

- Verification – hardware and software

  Major groups at Intel, IBM, Microsoft, and universities such as CMU, Cornell, and Princeton.
  SAT has become the dominant technology.

- Many other domains: Test pattern generation, Scheduling, Optimal Control, Protocol Design, Routers, Multi-agent systems, E-Commerce (E-auctions and electronic trading agents), etc.

Tutorial Outline

1. **Introduction**
2. **Probabilistic inference using message passing**
3. **Survey Propagation**
4. **Solution clusters**
5. **Probabilistic inference for clusters**
6. **Advanced topics**

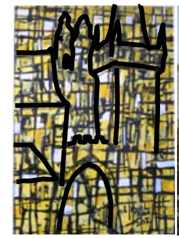
- Constraint satisfaction problems (CSPs)
  - SAT
  - Graph problems
- Random ensembles and satisfiability threshold
- Traditional approaches:
  - DPLL
  - Local search
- Probabilistic approaches:
  - Decimation
  - Reinforcement

Random Ensembles of CSPs

- Were a strong driving force for early research on SAT/CSP solvers (1990's)
  - Researchers were still struggling with 50-100 variable problems
  - *Without demonstrated potential of constraint solvers, industry had no incentive to create and provide "real-world instances"*
- Still provide very hard benchmarks for solvers
  - Easy to parameterize for experimentation: generate small/large instances, easy/hard instances
  - See 'random' category of SAT competitions
    - The "usual" systematic solvers can only handle <1000 variables
    - Local search solvers scale somewhat better
- Have led to an amazing amount of theoretical research, at the boundary of CS and Mathematics!

Random Ensembles of CSPs

- Studied often with N, the number of variables, as a scaling parameter
  - Asymptotic behavior: what happens to *almost all* instances as N $\rightarrow \infty$?
- While not considered *structured*, random ensembles exhibit remarkably precise "almost always" properties. E.g. -
  - Random 2-SAT instances are almost always satisfiable when #clauses < #variables, and almost always unsatisfiable otherwise
  - Chromatic number in random graphs of density d is almost always f(d) or f(d)+1, for some known, easy to compute, function f
  - As soon as almost any random graph becomes connected (as d increases), it has a Hamiltonian Cycle




- Note: although these seem easy as *decision problems,* this fact does not automatically yield an easy way to *find* a coloring or ham-cycle or satisfying assignment

Dramatic Chromatic Number

Structured or not?

With high probability, the chromatic number of a random graph with average degree $d=10^{60}$ is either

3771455490672260758090142394938336005516126417647650681575

or

3771455490672260758090142394938336005516126417647650681576

[credit: D.Achlioptas]

Random Graphs

- The **G(n,p)** Model (Erdos-Renyi Model):
  - Create a graph G on n vertices by including each of the $\binom{n}{2}$ potential edges in G independently with probability p
  - Average number of edges: p$\binom{n}{2}$
  - Average degree: p (n-1)
- The **G(n,m)** Model: [without repetition]
  - Create a graph G on n vertices by including exactly m randomly chosen edges out of the $\binom{n}{2}$ potential edges
  - **Graph density: α = m/n**

Fact: Various random graph models are essentially equivalent w.r.t. properties that hold almost surely

CSPs on Random Graphs

Note: can define all these problems on non-random graphs as well

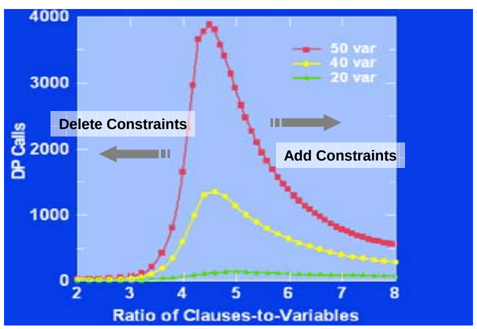
- **k-COL**
  Given a random graph G(n,p), can we color its nodes with k colors so that no two adjacent nodes get the same color?
  - Chromatic number: minimum such k
- **Vertex Cover of size k**
  Given a random graph G(n,p), can we find k vertices such that every edge is touches these k vertices?
- **Independent set of size k**
  Given a random graph G(n,p), can we find k vertices such that there is no edge between these k vertices?

Random k-SAT

- k-CNF: every clause has exactly k literals (a "k-clause")
- The **F(n,p)** model:
  - Construct a k-CNF formula F by including each of the $\binom{n}{k}2^k$ potential k-clauses in F independently with probability p
- The **F(n,m)** model: [without repetition]
  - Construct a k-CNF formula F by including exactly m randomly chosen clauses out of the $\binom{n}{k}2^k$ potential k-clauses in F independently
  - **Density: α = m/n**

Typical-Case Complexity: k-SAT

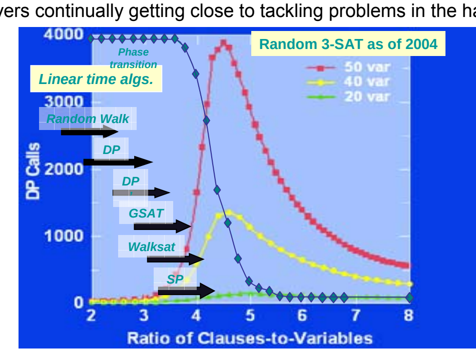
A key hardness parameter for k-SAT: the ratio of clauses to variables



Problems that are not critically constrained tend to be much easier in practice than the relatively few critically constrained ones

Typical-Case Complexity

SAT solvers continually getting close to tackling problems in the hardest region!



SP (survey propagation) now handles 1,000,000 variables very near the phase transition region

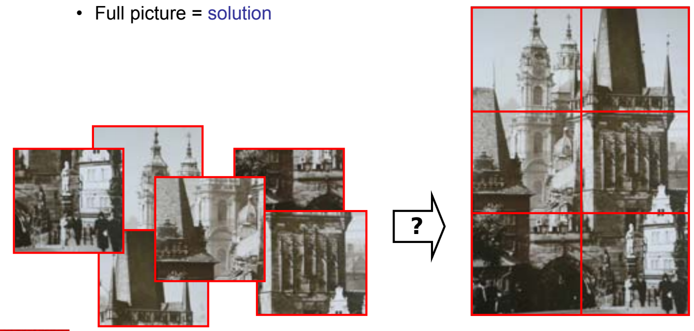
Tutorial Outline

1. **Introduction**
2. **Probabilistic inference using message passing**
3. **Survey Propagation**
4. **Solution clusters**
5. **Probabilistic inference for clusters**
6. **Advanced topics**

- Constraint satisfaction problems (CSPs)
  - SAT
  - Random graphs
- Random ensembles and satisfiability threshold
- Traditional approaches:
  - DPLL
  - Local search
- Probabilistic approaches:
  - Decimation
  - Reinforcement

CSP Example: a Jigsaw Puzzle

- Consider a puzzle to solve:
  - Squares = unknowns
  - Pieces = domain
  - Matching edges = constraints
  - Full picture = solution



Solving SAT: Systematic Search

One possibility: enumerate all truth assignments one-by-one, test whether any satisfies F

- Note: testing is easy!
- But too many truth assignments (e.g. for N=1000 variables, have $2^{1000} \approx 10^{300}$ truth assignments)

$$\left.\begin{array}{l} 00000000 \\ 00000001 \\ 00000010 \\ 00000011 \\ \ldots\ldots \\ 11111111 \end{array}\right\} 2^N$$

Solving SAT: Systematic Search

Smarter approach: the "DPLL" procedure [1960's]
(Davis, Putnam, Logemann, Loveland)

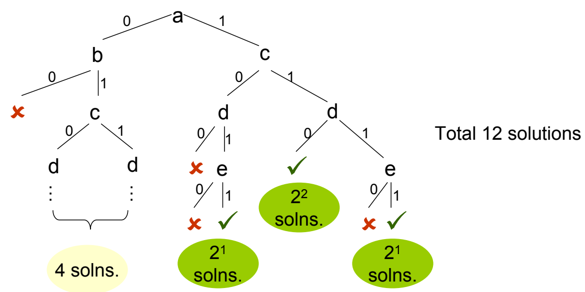
1. Assign values to variables one at a time ("partial" assignments)
2. Simplify F
3. If contradiction (i.e. some clause becomes False), "backtrack", flip last unflipped variable's value, and continue search

- Extended with many new techniques -- 100's of research papers, yearly conference on SAT
  e.g., extremely efficient data-structures (representation), randomization, restarts, learning "reasons" of failure
- Provides proof of unsatisfiability if F is unsat. ["complete method"]
- Forms the basis of dozens of very effective SAT solvers!
  e.g. minisat, zchaff, relsat, rsat, … (open source, available on the www)

Solving SAT: Systematic Search

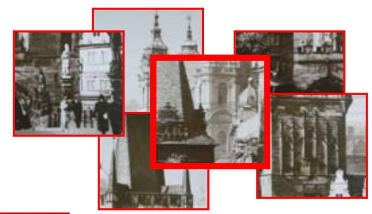
- For an N variable formula, if the residual formula is satisfiable after fixing d variables, count 2^{N-d} as the model count for this branch and backtrack.

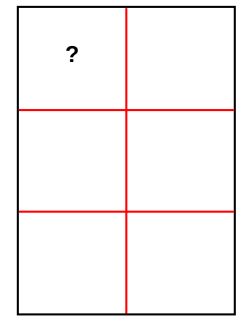
Consider $F = (a \vee b) \wedge (c \vee d) \wedge (\neg d \vee e)$



Solving the Puzzle: Systematic Search

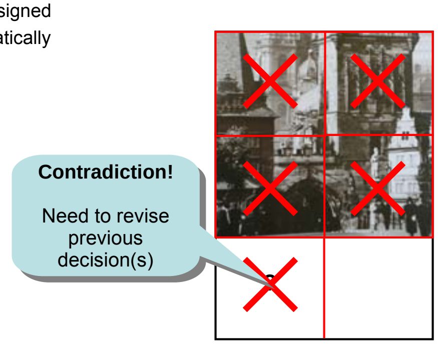
- **Search** for a solution **by backtracking**
  - Consistent but incomplete assignment
    - No constraints violated
    - Not all variables assigned
  - Choose values systematically





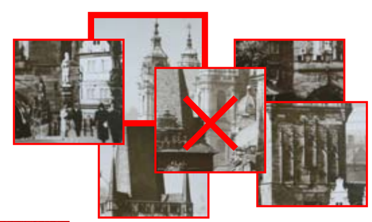
Solving the Puzzle: Systematic Search

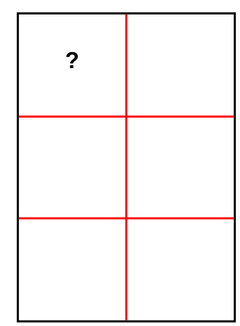
- **Search** for a solution **by backtracking**
  - Consistent but incomplete assignment
    - No constraints violated
    - Not all variables assigned
  - Choose values systematically



Solving the Puzzle: Systematic Search

- **Search** for a solution **by backtracking**
  - Consistent but incomplete assignment
    - No constraints violated
    - Not all variables assigned
  - Chose values systematically
  - Revise when needed





Solving the Puzzle: Systematic Search

- **Search** for a solution **by backtracking**
  - Consistent but incomplete assignment
    - No constraints violated
    - Not all variables assigned
  - Chose values systematically
  - Revise when needed
- Exhaustive search
  - Always finds a solution in the end (or shows there is none)
  - But it can take too long

Solving SAT: Local Search

- Search space: all 2^N truth assignments for F
- Goal: starting from an initial truth assignment A_0, compute assignments $A_1, A_2, \ldots, A_s$ such that A_s is a *satisfying* assignment for F

- A_{i+1} is computed by a "local transformation" to A_i

  e.g. A_1 = 000110111 green bit "flips" to red bit
  A_2 = 001110111
  A_3 = 001110101
  A_4 = 101110101
  …
  …
  A_s = 111010000 solution found!

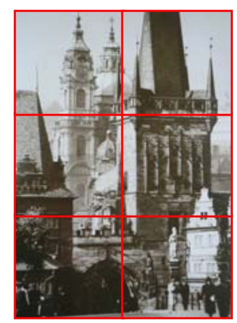
- *No proof* of unsatisfiability if F is unsat. ["incomplete method"]
- Several SAT solvers based on this approach, e.g. Walksat

Solving the Puzzle: Local Search

- **Search** for a solution by **local changes**
  - Complete but inconsistent assignment
    - All variables assigned
    - Some constraints violated
  - Start with a random assignment
  - With *local* changes try to find *globally* correct solution

Solving the Puzzle: Local Search

- **Search** for a solution by **local changes**
  - Complete but inconsistent assignment
    - All variables assigned
    - Some constraints violated
  - Start with a random assignment
  - With *local* changes try to find *globally* correct solution
- Randomized search
  - Often finds a solution quickly
  - But can get "stuck"



Tutorial Outline

1. **Introduction**
2. **Probabilistic inference using message passing**
3. **Survey Propagation**
4. **Solution clusters**
5. **Probabilistic inference for clusters**
6. **Advanced topics**

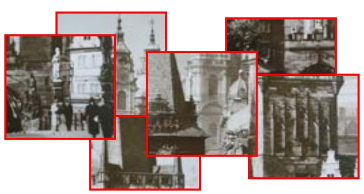
- Constraint satisfaction problems (CSPs)
  - SAT
  - Random graphs
- Random ensembles and satisfiability threshold
- Traditional approaches:
  - DPLL
  - Local search
- Probabilistic approaches:
  - Decimation
  - Reinforcement

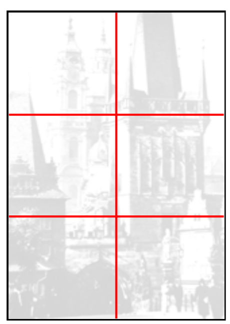
Solving SAT: Decimation

- "Search" space: all 2^N truth assignments for F
- Goal: attempt to construct a solution in "one-shot" by very carefully setting one variable at a time
- Decimation using Marginal Probabilities:
  - **Estimate** each variable's "marginal probability": *how often is it True or False in solutions?*
  - **Fix** the variable that is the most biased to its preferred value
  - **Simplify** F and **repeat**
- A method rarely used by computer scientists
  - Using #P-complete probabilistic inference to solve an NP-complete problem
- But has received tremendous success from the physics community
- No searching for solution, no backtracks
- No proof of unsatisfiability ["incomplete method"]

Solving the Puzzle: Decimation

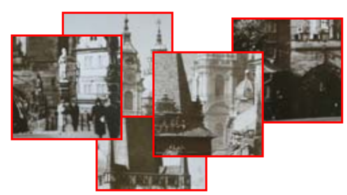
- Search by backtracking was pretty good
  - If only it didn't make wrong decisions
  - ⇒ Use some more "global" information
- **Construction:**
  - Spend a lot of effort on each decision
  - Hope you never need to revise: *a bold, greedy method*

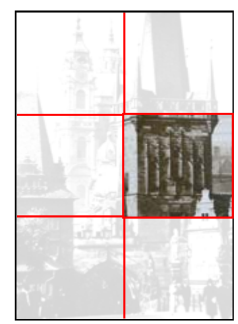




Solving the Puzzle: Decimation

- Search by backtracking was pretty good
  - If only it didn't make wrong decisions
  - ⇒ Use some more "global" information
- **Construction:**
  - Spend a lot of effort on each decision
  - Hope you never need to revise: *a bold, greedy method*





Solving the Puzzle: Decimation

- Search by backtracking was pretty good
  - If only it didn't make wrong decisions
  - ⇒ Use some more "global" information
- **Construction:**
  - Spend a lot of effort on each decision
  - Hope you never need to revise: *a bold, greedy method*

Solving SAT: (Reinforcement)

- Another way to using probabilistic information
  - If it works, it finds solutions faster
  - But more finicky than decimation

1. **Start** with uniform prior on each variable (no bias)
2. **Estimate** marginal probability, given this bias
3. **Adjust** prior ("reinforce")
4. **Repeat** until priors "point" to a solution

Not committing to a any particular value for any variable
Slowly evolving towards a consensus

Tutorial Outline

1. **Introduction**
2. **Probabilistic inference using message passing**
3. **Survey Propagation**
4. **Solution clusters**
5. **Probabilistic inference for clusters**
6. **Advanced topics**

CORNELL

Probabilistic Inference Using Message Passing

Tutorial Outline

1. **Introduction**
2. **Probabilistic inference using message passing**
3. **Survey Propagation**
4. **Solution clusters**
5. **Probabilistic inference for clusters**
6. **Advanced topics**

- Factor graph representation
- Inference using Belief Propagation (BP)
- BP inspired decimation

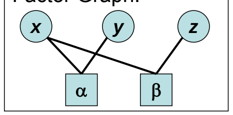
Encoding CSPs

- A CSP is a problem of finding a configuration (values of discrete variables) that is globally consistent (all constraints are satisfied)
- One can visualize the connections between variables and constraints in so called **factor graph**:
  - A bipartite undirected graph with two types of nodes:
    - Variables: one node per variable
    - Factors: one node per constraint
  - Factor nodes are connected to exactly variables from represented constraint

e.g. SAT Problem:

$$\underbrace{(\boldsymbol{x} \vee \boldsymbol{y})}_{\alpha} \wedge \underbrace{(\neg \boldsymbol{x} \vee \boldsymbol{z})}_{\beta}$$

Factor Graph:



Factor Graphs

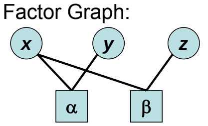
- Semantics of a factor graph:
  - Each variable node has an associated discrete domain
  - Each factor node α has an associated factor function $f_\alpha(x_\alpha)$, weighting the variable setting. For CSP, it =1 iff associated constraint is satisfied, else =0
  - Weight of the full configuration x: $F(\vec{x}) := \prod_\alpha f_\alpha(\vec{x}_\alpha)$
  - Summing weights of all configurations defines **partition function:**

$$\boxed{Z := \sum_{\vec{x}} \prod_\alpha f_\alpha(\vec{x}_\alpha)}$$

  - For CSPs the partition function computes the number of solutions

$$\underbrace{(\boldsymbol{x} \vee \boldsymbol{y})}_{\alpha} \wedge \underbrace{(\neg \boldsymbol{x} \vee \boldsymbol{z})}_{\beta}$$

Factor Graph:



x	y	z	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1
Z=			4

Probabilistic Interpretation

- Given a factor graph (with non-negative factor functions) the probability space is constructed as:
  - Set of possible worlds = configurations of variables
  - Probability mass function = normalized weights

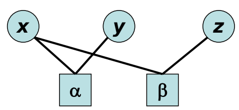
$$\Pr[X = \vec{x}] = \frac{1}{Z} F(\vec{x})$$

  - For CSP: Pr[X=x] is either 0 or 1/(number of solutions)
- Factor graphs appear in probability theory as a compact representation of factorizable probability distributions
  - Concepts like marginal probabilities naturally follow.
  - Similar to Bayesian Nets.

Relation to Bayesian Networks

- Factor graphs are very similar to Bayesian Networks
  - Variables have uniform prior
  - Factors become auxiliary variables with {0,1} values
  - Conditional probability tables come from factor functions.
  - *F*(configuration *x*) ∝ Pr[configuration *x* | all auxiliary variables = 1]

Factor Graph:



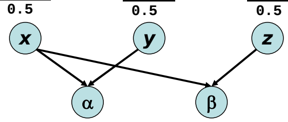
x y	$f_\alpha(x,y)$
0 0	0
0 1	1
1 0	1
1 1	1

x z	$f_\beta(x,z)$
0 0	1
0 1	1
1 0	0
1 1	1

≡

Bayesian Network:

P(x=1)	P(y=1)	P(z=1)
0.5	0.5	0.5



x y	P(α=1|x,y)
0 0	0
0 1	1
1 0	1
1 1	1

x z	P(β=1|x,z)
0 0	1
0 1	1
1 0	0
1 1	1

Querying Factor Graphs

- What is the value of the partition function *Z*?
  - E.g. count number of solutions in CSP.

$$\sum_{\vec{x}} \prod_\alpha f_\alpha(\vec{x}_\alpha)$$

- What is the configuration with maximum weight *F(x)*?
  - E.g. finds one (some) solution to a CSP.
  - Maximum Likelihood (ML) or Maximum A'Posteriori (MAP) inference

$$\mathrm{argmax}_{\vec{x}} \prod_\alpha f_\alpha(\vec{x}_\alpha)$$

- What are the **marginals** of the variables?
  - E.g. fraction of solutions in which a variable *i* is fixed to x_i.

$$p_i(x_i) = \frac{1}{Z}\sum_{\vec{x}_{-i}} \underbrace{\prod_\alpha f_\alpha(\vec{x}_\alpha)}_{=F(\vec{x})}$$

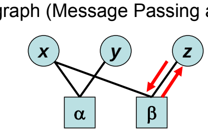
Notation: x_{-i} are all variables except x_i

Tutorial Outline

1. **Introduction**
2. **Probabilistic inference using message passing**
   - Factor graph representation
   - Inference using Belief Propagation (BP)
   - BP inspired decimation
3. **Survey Propagation**
4. **Solution clusters**
5. **Probabilistic inference for clusters**
6. **Advanced topics**

Inference in Factor Graphs

- Inference: answering the previous questions
- Exact inference is a #P-complete problem, so it does not take us too far
- Approximate inference is the way to go!
- A very popular algorithm for doing approximate inference is **Belief Propagation (BP),** sum-product algorithm
  - An algorithm in which an "agreement" is to be reached by sending "messages" along edges of the factor graph (Message Passing algorithm)



  - **PROS:** very scalable
  - **CONS:** finicky, exact only on tree factor graph, in general gives results of uncertain quality

Belief Propagation

- A famous algorithm, rediscovered many times and in many incarnations
  - Bethe's approximations in spin glasses [1935]
  - Gallager Codes [1963] (later Turbo codes)
  - Viterbi algorithm [1967]
  - BP for Bayesian Net inference [1988]

- Blackbox BP (for marginals):
  - Iteratively solve the following set of recursive equations in [0,1]

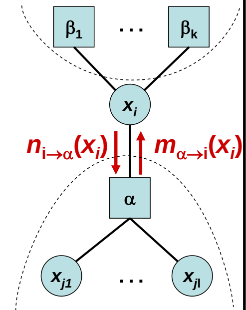
    $$n_{i\to\alpha}(x_i) \propto \prod_{\beta \ni i \backslash \alpha} m_{\beta\to i}(x_i)$$
    $$m_{\alpha\to i}(x_i) \propto \sum_{\vec{x}_{\alpha\backslash i} \in Dom^{|\alpha|-1}} f_\alpha(\vec{x}_\alpha) \prod_{j\in\alpha\backslash i} n_{j\to\alpha}(x_j)$$

  - Then compute marginal estimates (**beliefs**) as:

    $$b_i(x_i) \propto \prod_{\alpha \ni i} m_{\alpha\to i}(x_i)$$

BP Equations Dissected

- The "messages" are functions of the variable end of the edge:
  - Normalized to sum to a constant, e.g. 1
  - $n_{i\to\alpha}(x_i)$ = "Marginal probability of x_i without the whole downstream"
  - $m_{\alpha\to i}(x_i)$ = "Marginal probability of x_i without the rest of downstream"



$$n_{i\to\alpha}(x_i) \propto \prod_{\beta \ni i \backslash \alpha} m_{\beta\to i}(x_i)$$

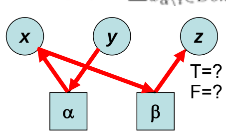
- Product across all factors with x_i except for α

$$m_{\alpha\to i}(x_i) \propto \sum_{\vec{x}_{\alpha\backslash i} \in Dom^{|\alpha|-1}} f_\alpha(\vec{x}_\alpha) \prod_{j\in\alpha\backslash i} n_{j\to\alpha}(x_j)$$

- Sum across all configurations of variables in α except x_i, of products across all variables in α except x_i

Belief Propagation as Message Passing

$$n_{i\to\alpha}(x_i) \propto \prod_{\beta \ni i \backslash \alpha} m_{\beta\to i}(x_i)$$
$$m_{\alpha\to i}(x_i) \propto \sum_{\vec{x}_{\alpha\backslash i} \in Dom^{|\alpha|-1}} f_\alpha(\vec{x}_\alpha) \prod_{j\in\alpha\backslash i} n_{j\to\alpha}(x_j)$$



$(x \vee y)$ $(\neg x \vee z)$

Solutions:

x	y	z
0	1	0
0	1	1
1	0	1
1	1	1

$n_{y\to\alpha}(T) = p_y^{upstream}(T) = 0.5$
$n_{y\to\alpha}(F) = p_y^{upstream}(F) = 0.5$

$m_{\alpha\to x}(T) = p_x^{upstream}(T) \propto 1$
$m_{\alpha\to x}(F) = p_x^{upstream}(F) \propto 0.5$

$n_{x\to\beta}(T) = p_x^{upstream}(T) = 0.66$
$n_{x\to\beta}(F) = p_x^{upstream}(F) = 0.33$

$m_{\beta\to z}(T) = p_z^{upstream}(T) \propto 1$
$m_{\beta\to z}(F) = p_z^{upstream}(F) \propto 0.33$

$b_z(T) = 0.75$
$b_z(F) = 0.25$

Basic Properties of BP

- Two main concerns are:
  - Finding the fixed point: do the iterations converge (completeness)?
  - Quality of the solution: how good is the approximation (correctness)?

- On factor graphs that are trees, BP always converges, and is exact
  - This is not surprising as the inference problems on trees are easy (polytime)

- On general factor graphs, the situation is worse:
  - **Convergence:** not guaranteed with simple iteration. But there are many ways to circumvent this, with various tradeoffs of speed and accuracy of the resulting fixed point (next slide)
  - **Accuracy:** not known in general, and hard to assess. But in special cases, e.g. when the factor graphs only has very few loops, can be made exact. In other cases BP is exact by itself (e.g. when it is equivalent to LP relaxation of a Totally Unimodular Problem)

Convergence of BP

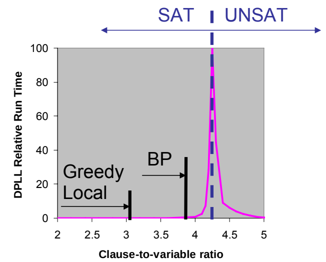
- The simplest technique: start with random messages and iteratively (a/synchronously) update until convergence might not work
  - In fact, does not work on many interesting CSP problems with "structure".
  - But on some (e.g. random) sparse factor graphs it works (e.g. decoding).
- Techniques to circumvent include:
  - Different solution technique:
    - E.g. Convex-Concave Programming: The BP equations can be cast as stationary point conditions for an optimization problem with objective function being sum of convex and concave functions. Provably convergent, but quite slow.
    - E.g. Expectation-Maximization BP: the minimization problem BP is derived from is solved by EM algo. Fast but very greedy.
  - Weak damping: make smaller steps in the iterations. Fast, but might not converge. $\kappa \in [0,1]$ is the damping parameter.

$$n_{i\to\alpha}^{new}(x_i) = (1-\kappa)\cdot n_{i\to\alpha}(x_i) + \kappa \cdot n_{i\to\alpha}^{old}(x_i)$$

  - Strong damping: fast and convergent, but does not solve the original equations $n_{i\to\alpha}^{new}(x_i) \propto (n_{i\to\alpha}(x_i))^{1-\kappa}$

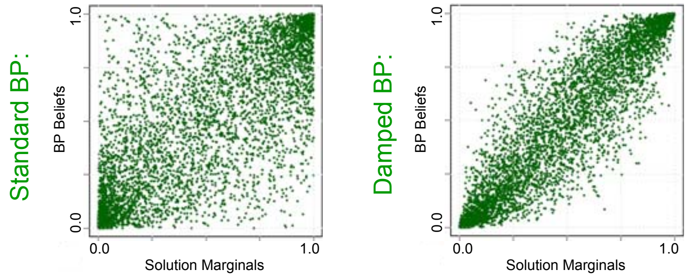
BP for Solving CSPs

- The maximum likelihood question is quite hard for BP to approximate
  - The convergence issues are even stronger.
  - "Finding the whole solution at once is too much to ask for."
- The way that SP was first used to solve hard random 3-SAT problems was via decimation guided by the marginal estimates.
- How does regular BP do when applied to random 3-SAT problems?
  - It does work, but only for $\alpha \leq 3.9$. The problems are 'easy', i.e. easily solvable by other techniques (e.g. advanced local search)



BP for Random 3-SAT

- What goes wrong with BP for random 3-SAT with $\alpha > 3.9$?
  1. It does not converge
  2. When made to converge, the results are not good enough for decimation

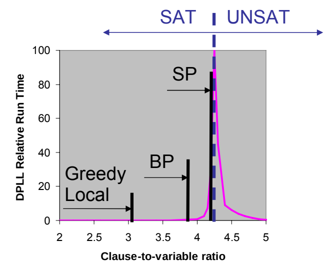


Survey Propagation

Tutorial Outline

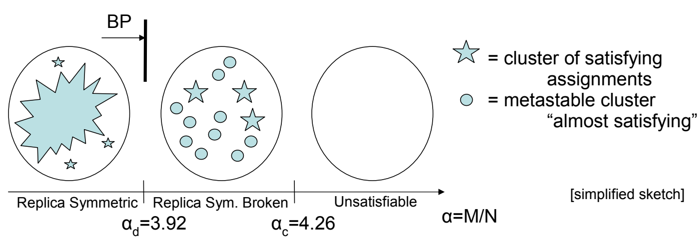
"Magic Solver for SAT"!

- **Survey Propagation (SP) [2002]**
  - Developed in statistical physics community [Mezard, Parisi, Zecchina '02]
    - Using cavity method and replica symmetry breaking (1-RSB).
  - Using unexpected techniques, delivers unbelievable performance!
    - Using approximate probabilistic methods in SAT solving was previously unheard of. Indeed, one is tackling a #P-complete problem to solve NP-complete one!
    - Able to solve random SAT problems with 1,000,000s of variables in the hard region, where other solvers failed on 1,000s.
  - **Importantly:** sparkled renewed interest in pro probabilistic techniques for solving CSPs.



Preview of Survey Propagation

- SP was not invented with the goal of solving SAT problems in mind
  - It was devised to reason about spin glasses (modeling magnets) with many metastable and ground states.
- The principal observation behind the idea of SP is that the solutionspace of random k-SAT problems **breaks** into many well separated regions with high density of solutions ("clusters")



Preview of Survey Propagation

- The existence of many metastable states and clusters confuses SAT solvers and BP.
  - BP: does not converge due to strong attraction into many directions.
  - Local search: current state partly in once cluster, partly in another.
  - DPLL: each cluster has many variables that can only take one value.
- **Survey Propagation circumvents this by focusing on clusters, rather than on individual solutions.**

SP Demo

Survey Propagation Equations for SAT

- SP equations for SAT:

  The black part is exactly BP for SAT

$$
\begin{aligned}
\eta_{\alpha\to i} &= \prod_{j\in\alpha\setminus i} \frac{\pi^u_{j\to\alpha}}{\pi^u_{j\to\alpha}+\pi^s_{j\to\alpha}+\pi^*_{j\to\alpha}} \\
\pi^u_{i\to\alpha} &= \prod_{\beta\in V^s_\alpha(i)}(1-\eta_{\beta\to i})\left[1-\prod_{\beta\in V^u_\alpha(i)}(1-\eta_{\beta\to i})\right] \\
\pi^s_{i\to\alpha} &= \prod_{\beta\in V^u_\alpha(i)}(1-\eta_{\beta\to i})\left[1-\prod_{\beta\in V^s_\alpha(i)}(1-\eta_{\beta\to i})\right] \\
\pi^*_{i\to\alpha} &= \prod_{\beta\ni i\setminus\alpha}(1-\eta_{\beta\to i})
\end{aligned}
$$

  Notation:

  $V_\alpha^u(i)$ set of all clauses where x_i appears with opposite sign than in α.
  $V_\alpha^s(i)$ set of all clauses where x_i appears with the same sign than in α.

- SP inspired decimation:
  - Once a fixed point is reached, analogous equations are used to compute beliefs for decimation. $b_x(0/1)$ = "fraction of clusters where x is fixed to 0/1"
    $b_x(*)$ = "fraction of clusters where x is not fixed"
  - When the decimated problem becomes easy, calls another solver.

Survey Propagation and Clusters

- The rest of the tutorial describes ways to **reason about clusters**
  - Some do lead to exactly SP algorithm, some do not.
  - Focuses on combinatorial approaches, developed after SP's proven success, with more accessible CS terminology. Not the original statistical physics derivation.
- The goal is to approximate marginals of **cluster backbones**, that is variables that can only take one value in a cluster.
  - So that as many clusters as possible survive decimation.

Objective:
Understand how can solutionspace structure, like clusters, be used to improve problem solvers, ultimately moving from random to practical problems.

CORNELL

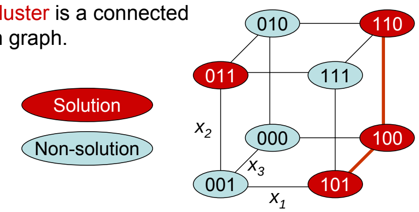
Solution Clusters

Tutorial Outline

1. **Introduction**
2. **Probabilistic inference using message passing**
3. **Survey Propagation**
4. **Solution clusters**
   - Cluster label & cover
   - Cluster filling & $Z_{(-1)}$
   - Cluster as a fixed point of BP
5. **Probabilistic inference for clusters**
6. **Advanced topics**

Clusters of Solutions

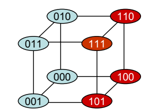
- **Definition:** A solution graph is an undirected graph where nodes correspond to solutions and are neighbors if they differ in value of only one variable.
- **Definition:** A solution cluster is a connected component of a solution graph.



- **Note:** this is not the only possible definition of a cluster, but the most 'combinatorial one'. Other possibilities include:
  - Solutions differing in constant fraction or *o(n)* of vars. are neighbors
  - Ground states: physics view

Thinking about Clusters

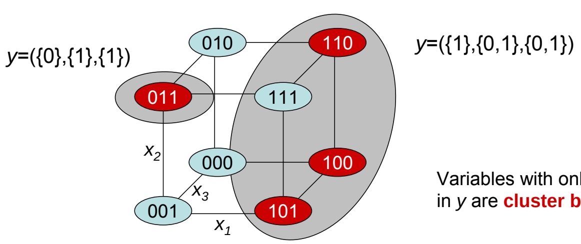
- Clusters are subsets of solutions, possibly exponential in size
  - Impractical to work with in this explicit form
- To compactly represent clusters, we need to trade off some expressive power for shorter representation
  - Will loose some details about the cluster, but will be able to work with it.
- **We will approximate clusters by hypercubes "from outside" and "from inside".**
  - **Hypercube:** Cartesian product of non-empty subsets of variable domains
    - E.g. *y*=({1},{0,1},{0,1}) is a 2-dimensional hypercube in 3-dim space



  - From outside: The (unique) minimal hypercube enclosing the whole cluster.
  - From inside: A (non-unique) maximal hypercube fitting inside the cluster.
  - The approximations are equivalent if clusters are indeed hypercubes.

Cluster Approximation from Outside

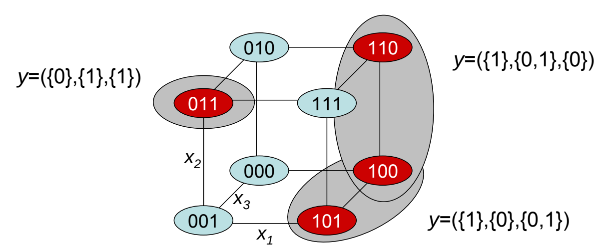
- Detailed **Cluster Label** for cluster C: The (unique) minimal hypercube *y* enclosing the whole cluster.
  1) No solution sticks out: setting any x_i to a value not in y_i cannot be extended to a solution from *C* $\quad \forall i \forall v_i \notin y_i\ \neg\exists \vec{x} : x_i = v_i \wedge \vec{x} \in \mathcal{C}$
  2) The enclosing is tight: setting any variable x_i to any value from y_i can be extended to a full solution from *C* $\quad \forall i \forall v_i \in y_i\ \exists \vec{x} : x_i = v_i \wedge \vec{x} \in \mathcal{C}$



Variables with only one value in *y* are **cluster backbones**

Cluster Approximation from Inside

- **Cluster Filling** for cluster C: A (non-unique) maximal hypercube fitting entirely inside the cluster.
  1) The hypercube *y* fits inside the cluster $\quad \vec{y} \subseteq \mathcal{C}$
  2) The hypercube cannot grow: extending the hypercube in any direction *i* sticks out of the cluster $\quad \forall i \forall v_i \notin y_i\ \exists \vec{x}_{-i} \in \vec{y}_{-i} : x_i = v_i \wedge \vec{x} \notin \mathcal{C}$



Difficulties with Clusters

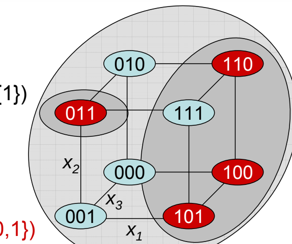
- **Even the simplest case is very hard!** Given *y* verify that *y* is the detailed cluster label (smallest enclosing hypercube) of a solutionspace with only one cluster.
  - We need to show that the enclosing does not leave out any solution. (coNP-style question)
  - Plus we need to show that the enclosing is tight. (NP-style question)
  - This means both NP and co-NP strength is needed even for verification!
- Now we will actually want to COUNT such cluster labels, that is solve the counting version of the decision problem!

Reasoning about clusters is hard!

Reasoning about Clusters

- We still have the explicit cluster *C* in those expressions, so need to simplify further to be able to reason about it efficiently.
  - Use only test for satisfiability instead of test for being in the cluster.
- **Simplified cluster label** (approximation from outside):

$$(1)\ \forall i \forall v_i \notin y_i\ \neg\exists \vec{x} : x_i = v_i \wedge \vec{x}_{-i} \in \cancel{C}\ \vec{y}_{-i} \wedge\ F(\vec{x}) = 1$$
$$(2)\ \forall i \forall v_i \in y_i\ \exists \vec{x} : x_i = v_i \wedge \vec{x}_{-i} \in \cancel{C}\ \vec{y}_{-i} \wedge\ F(\vec{x}) = 1$$

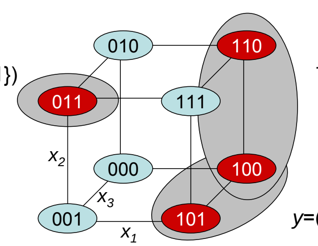


Note that *y* can now enclose multiple clusters!

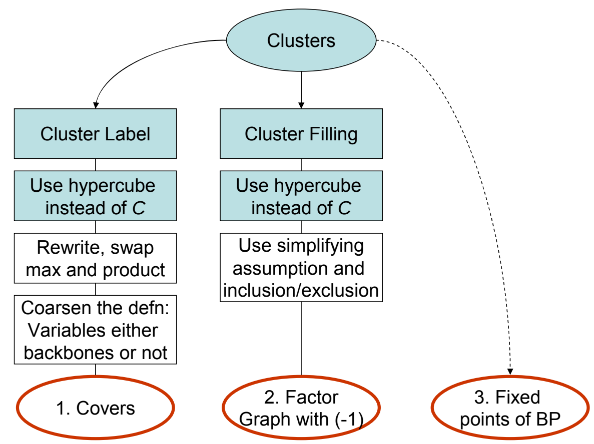
Reasoning about Clusters

- **Simplified cluster filling** (approximation from inside):

$$(1)\ \cancel{\vec{y} \subseteq C}\ \forall \vec{x} \in \vec{y} : F(\vec{x}) = 1$$
$$(2)\ \forall i \forall v_i \notin y_i\ \exists \vec{x}_{-i} \in \vec{y}_{-i} : x_i = v_i \wedge \cancel{\vec{x} \notin C}\ F(\vec{x}) = 0$$



The Approximations of Clusters



The Cover Story

- Rewrite the conditions for simplified cluster label

$$(1)\ \forall i \forall v_i \notin y_i\ \neg\exists \vec{x} : x_i = v_i \wedge \vec{x}_{-i} \in \vec{y}_{-i} \wedge F(\vec{x}) = 1$$
$$(2)\ \forall i \forall v_i \in y_i\ \exists \vec{x} : x_i = v_i \wedge \vec{x}_{-i} \in \vec{y}_{-i} \wedge F(\vec{x}) = 1$$

as:

$$(1)\ \forall i \forall v_i \notin y_i : \max_{\vec{x}_{-i} \in \vec{y}_{-i}, x_i = v_i} \prod_\alpha f_\alpha(\vec{x}_\alpha) = 0$$
$$(2)\ \forall i \forall v_i \in y_i : \max_{\vec{x}_{-i} \in \vec{y}_{-i}, x_i = v_i} \prod_\alpha f_\alpha(\vec{x}_\alpha) = 1$$

- Swapping **max** and **product**:

$$(1)\ \forall i \forall v_i \notin y_i : \prod_\alpha \max_{\vec{x}_{\alpha \setminus i} \in \vec{y}_{\alpha \setminus i}, x_i = v_i} f_\alpha(\vec{x}_\alpha) = 0$$
$$(2)\ \forall i \forall v_i \in y_i : \prod_\alpha \max_{\vec{x}_{\alpha \setminus i} \in \vec{y}_{\alpha \setminus i}, x_i = v_i} f_\alpha(\vec{x}_\alpha) = 1$$

  - This makes it efficient: **from exponential to polynomial complexity**
  - But this approximation changes semantics a lot as discussed later.

The Cover Story

$$(1)\ \forall i \forall v_i \notin y_i : \prod_\alpha \max_{\vec{x}_{\alpha \setminus i} \in \vec{y}_{\alpha \setminus i}, x_i = v_i} f_\alpha(\vec{x}_\alpha) = 0$$
$$(2)\ \forall i \forall v_i \in y_i : \prod_\alpha \max_{\vec{x}_{\alpha \setminus i} \in \vec{y}_{\alpha \setminus i}, x_i = v_i} f_\alpha(\vec{x}_\alpha) = 1$$

- Finally, we will only focus on variables that are cluster backbones (when $|y_i|=1$), and will use $*$ to denote variables that are not (if $|y_i|>1$)

- A **cover** is a vector z of domain values or $*$

$$z_i = \begin{cases} v_i & \text{if } y_i = \{v_i\} \\ * & \text{if } |y_i| > 1 \end{cases}$$

  - Cover is polynomial to verify
  - Hypercube enclosing whole clusters, but not necessarily the minimal one (not necessarily all cluster backbone variables are identified).

The Cover Story for SAT

- The above applied to SAT yields this characterization of a cover:
  - Generalized {0,1,$*$} assignments ($*$ means "undecided") such that
    1. Every clause has a satisfying literal or ≥ 2 $*$s
    2. Every non-$*$ variable has a "certifying" clause in which all other literals are false
  - E.g. the following formula has 2 covers: ($*$ $*$ $*$) and (000)
    This is actually correct, as there are exactly two clusters

$$F = \underbrace{(\neg x \vee y \vee z)}_{\alpha} \wedge \underbrace{(x \vee \neg y \vee z)}_{\beta} \wedge \underbrace{(x \vee y \vee \neg z)}_{\gamma}$$

- We arrived at a **new combinatorial object**
  - Number of covers gives an approximation to number of clusters
  - Cover marginals approximate cluster backbone marginals.

Properties of Covers for SAT

- Covers represent solution clusters
  - $*$ generalizes both 0 and 1
  - Clusters have unique covers.

Solutions	Hierarchy of covers	
1 0 0 0	1 * 0 *	* * * *
1 0 0 1		
1 1 0 0		
0 0 1 0	0 0 1 *	
0 0 1 1		

- Some covers do not generalize any solution (false covers)

- Every formula (sat or unsat) without unit clauses has the trivial cover, all stars $***$

- Set of covers for a given formula depends on both semantics (set of satisfying assignments) and syntax (the particular set of clauses used to define the solutionspace)

Properties of Covers for SAT

- Covers provably exist in k-SAT for k≥9
  - For k=3 they are very hard to find (much harder than solutions!) but empirically also exist.
- Unlike finding solutions, finding covers is not a self-reducible problem
  - covers cannot be found by simple decimation
  - e.g. if we guess that in some cover x=0, and use decimation:

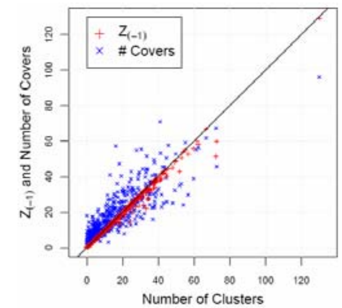
$$F = \cancel{(\neg x \vee y \vee z)} \wedge (\cancel{x} \vee \neg y \vee z) \wedge (\cancel{x} \vee y \vee \neg z)$$

$$F' = (\neg y \vee z) \wedge (y \vee \neg z)$$

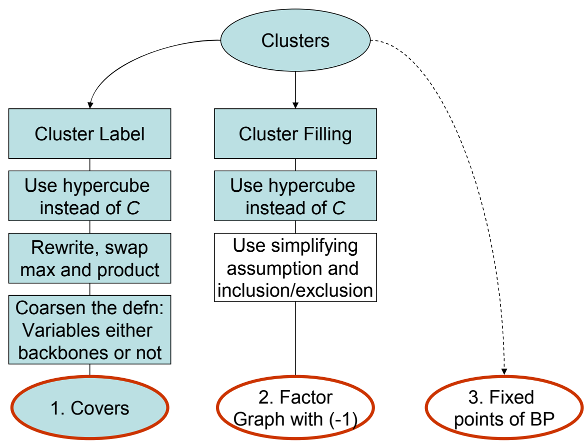
(11) is a cover for F' but (011) is not a cover for F

Empirical Results: Covers for SAT

Random 3-SAT, n=90, α=4.0
One point per instance



The Approximations of Clusters



The (-1) Story

- **Simplified cluster filling** (approximation from inside):

$$(1)\ \forall \vec{x} \in \vec{y} : F(\vec{x}) = 1$$
$$(2)\ \forall i \forall v_i \notin y_i\ \exists \vec{x}_{-i} \in \vec{y}_{-i} : x_i = v_i \wedge F(\vec{x}) = 0$$

- With *F(y)* being the natural extension of *F(x)* to hypercubes (condition (1)), we can rewrite the conditions above as the indicator function for simplified cluster filling:

$$\chi(\vec{y}) = \overbrace{F(\vec{y})}^{(1)} \cdot \overbrace{\prod_i \prod_{v_i \notin y_i} \left(1 - F(\vec{y}[y_i \leftarrow y_i \cup \{v_i\}])\right)}^{(2)}$$
$$= F(\vec{y}) \left(\sum_{\vec{y}' \in (\mathcal{P}(Dom))^n \setminus \vec{y}} (-1)^{\#_o(\vec{y}')} \prod_i \prod_{v_i \in y'_i} F(\vec{y}[y_i \leftarrow y_i \cup \{v_i\}]) \right)$$

Notation: $\#_o(y)$ is the number of odd-sized elements of *y*

The (-1) Story

- Now summing $\chi(y)$ across all candidate cluster fillings:

$$\vec{y} \in DomExt^n$$
$$DomExt := \mathcal{P}(\{c_1, \ldots, c_k\}) \setminus \emptyset$$

and using a simplifying assumption, we derive the following approximation of number of clusters:

$$\boxed{Z_{(-1)} = \sum_{\vec{y} \in DomExt^n} (-1)^{\#_e(\vec{y})} \prod_{\alpha} f_\alpha(\vec{y}_\alpha)} \qquad f_\alpha(\vec{y}_\alpha) = \min_{\vec{x}_\alpha \in \vec{y}_\alpha} f_\alpha(\vec{x}_\alpha)$$

Notation: ($\#_e(y)$ is the number of even-sized elements of y)

- Syntactically very similar to standard Z, which computes exactly number of solutions

$$Z = \sum_{\vec{x} \in Dom^n} \prod_{\alpha} f_\alpha(\vec{x}_\alpha)$$

Properties of $Z_{(-1)}$ for SAT

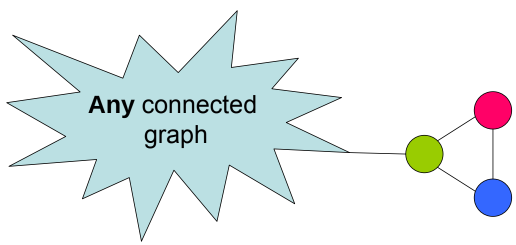
- $Z_{(-1)}$ is a function of the solutionspace only (semantics of the problem), does not depend on the way it is encoded (syntax)

On what kind of solutiospaces does $Z_{(-1)}$ count number of clusters exactly?

- A theoretical framework can be developed to tackle the question. E.g. if a solutionspace satisfies certain properties (we call such solutionspaces k-simple), the $Z_{(-1)}$ is exact, and also gives exact backbone marginals:
- **Theorem**: if the solutionspace decomposes into 0-simple subspaces, then $Z_{(-1)}$ is exact.
  - (empirically, solutionspace of random 3-SAT formulas decompose to "almost 0-simple" spaces)
- **Theorem**: if the solutionspace decomposes into 1-simple subspaces, then marginal sums of $Z_{(-1)}$ correctly capture information about cluster backbones

Properties of $Z_{(-1)}$ for COL

- **Theorem**: If every connected component of graph G has at least one triangle, then the $Z_{(-1)}$ corresponding to 3-COL problem on G is exact.

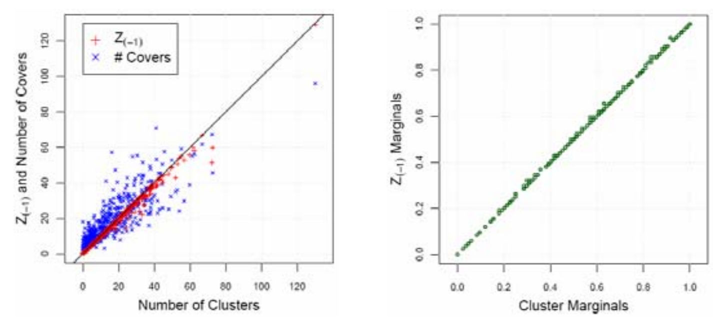


- **Corollary**: On random graphs with at least constant average degree, $Z_{(-1)}$ counts exactly the number of solution clusters of 3-COL with high probability.

Empirical Results: $Z_{(-1)}$ for SAT

Random 3-SAT, n=90, α=4.0
One point per instance

Random 3-SAT, n=200, α=4.0
One point per variable
One instance



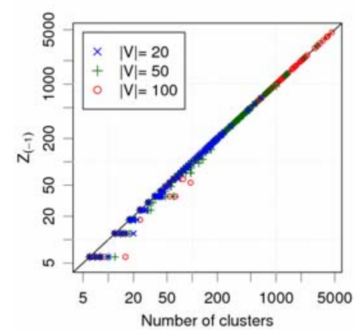
Empirical Results: $Z_{(-1)}$ for SAT

- $Z_{(-1)}$ is remarkably accurate even for many structured formulas (formulas encoding some real-world problem):

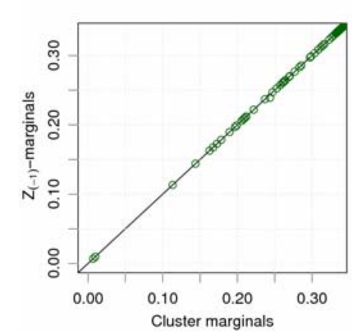
Instance Name	# solutions	# clusters	$Z_{(-1)}$
v32r250p1	52081218	6	6
v32r500p5	1543304664	6	6
driverlog1_ks99i	856152	338100	338100
rovers1_ks99i	17850294	15	15
rovers1_v01a	83200608	46	46
rovers1_v01i	266000	15	15
rovers2_ks99i	531360	8	8
rovers2_v01a	52107696	316	308
rovers2_v01i	21504	8	8
rovers4_ks99i	13794198600	11	11
rovers4_v01a	2592794880	22	22
rovers4_v01i	28447200	11	11

Empirical Results: $Z_{(-1)}$ for COL

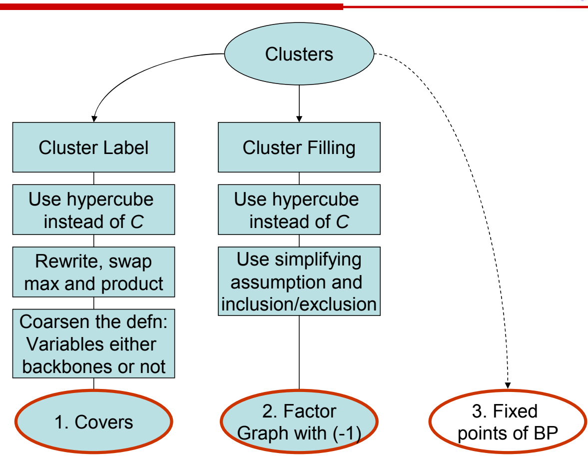
Random 3-COL, various sizes, avg.deg 1.0-4.7
One point per instance, log-log



Random 3-COL, n=100
One point per variable
One instance

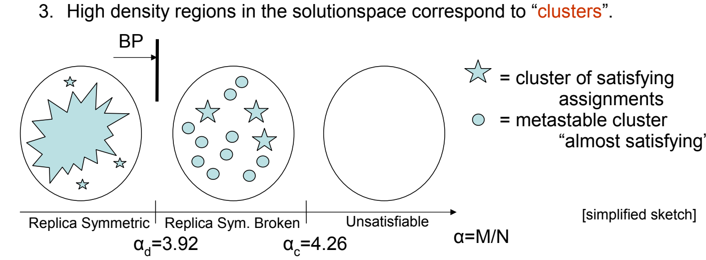


The Approximations of Clusters



Clusters as Fixed Points of BP

- Coming from physics intuition (for random problems):
  1. BP equations have multiple fixed points, resulting in multiple sets of beliefs
  2. Each set of beliefs corresponds to a region with "high density of solutions"
  3. High density regions in the solutionspace correspond to "clusters".



- This notion of a cluster is closely related to the cover object, and counting number of fixed points of BP to counting number of covers.
  - As we will see later.

Coming up next….

- We have 3 ways at approximately characterize clusters
- We want to be able to count them and find marginal probabilities of cluster backbones
- We will use Belief Propagation to do approximate inference on all three cluster characterizations
  - Which is where the Survey Propagation algorithm will come from.

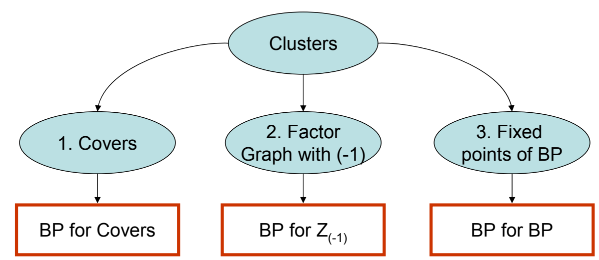
Probabilistic Inference for Clusters

CORNELL

Tutorial Outline

1. **Introduction**
2. **Probabilistic inference using message passing**
3. **Survey Propagation**
4. **Solution clusters**
5. **Probabilistic inference for clusters**
   - BP for covers
   - BP for $Z_{(-1)}$
   - BP for fixed points of BP
     - The origin of SP
6. **Advanced topics**

Belief Propagation for Clusters

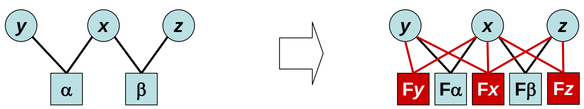


- For SAT, they all boil down to the same algorithm ≡ Survey Propagation
- For COL, (-1) and BP fixed points differ (uncertain for covers)

In general, Survey Propagation is "BP for fixed points of BP"

BP for Covers

- Reminder: cover for SAT
  - Generalized {0,1,∗} assignments (∗ means "undecided") such that
    1. Every clause has a satisfying literal or ≥ 2 ∗s
    2. Every non-∗ variable has a "certifying" clause in which all other literals are false
- Applying BP directly on the above conditions creates a very dense factor graph
  - Which is not good because BP works best on low density factor graphs.
  - The problem is the second condition: the verifying factor not only needs to be connected to the variable, but also to all its neighbors at distance 2.



- We will define a more local problem equivalent to covers, and apply BP

BP for Covers in SAT

- Covers of a formula are in one-to-one correspondence to fixed points of **discrete** Warning Propagation (WP):
  - **Request**∈{0,1}: from clause to variable, with meaning "you better satisfy me!"….because no other variable will.

$$(r_{\alpha\to i}=1) \text{ iff } (\forall j\in\alpha\setminus i : w_{j\to\alpha}=1)$$

  - **Warning**∈{0,1}: from variable to clause with meaning "I cannot satisfy you!"….because I received a request from at least one opposing clause.

$$(w_{i\to\alpha}=1) \text{ iff } (\exists\beta\in V^u_\alpha(i)\setminus i : r_{\beta\to i}=1)$$

Notation: $V_\alpha^u(i)$ set of all clauses where x_i appears with opposite sign than in α.

Equivalence of Covers and WP solutions

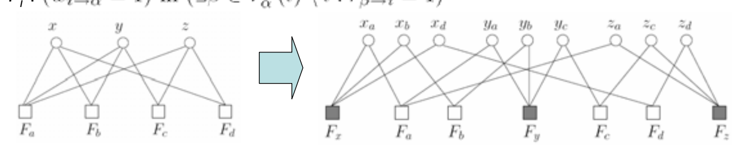
$$(r_{\alpha\to i}=1) \text{ iff } (\forall j\in\alpha\setminus i : w_{j\to\alpha}=1)$$
$$(w_{i\to\alpha}=1) \text{ iff } (\exists\beta\in V^u_\alpha(i)\setminus i : r_{\beta\to i}=1)$$

- Once a WP solution is found, variable is:
  - **1** if it receives a request from a clause where it is positive
  - **0** if it receives a request from a clause where it is positive
  - **∗** if it does not receive any request at all
  - Variable cannot be receiving conflicting requests in a solution.
- This assignment is a cover:
  1. Every clause has satisfying literal or ≥ 2 ∗s
     - Because otherwise the clause would send a warning to some variable
  2. Every non-∗ variable has a "certifying" clause
     - Because otherwise the variable would not receive a request

Applying BP to Solutions of WP

- A factor graph can be build to represent the WP constraints, with variables being request-warning pairs between a variable and a clause $(r,w)\in\{(0,0),(0,1),(1,0)\}$

F_α: $(r_{\alpha\to i}=1) \text{ iff } (\forall j\in\alpha\setminus i : w_{j\to\alpha}=1)$
F_i: $(w_{i\to\alpha}=1) \text{ iff } (\exists\beta\in V^u_\alpha(i)\setminus i : r_{\beta\to i}=1)$



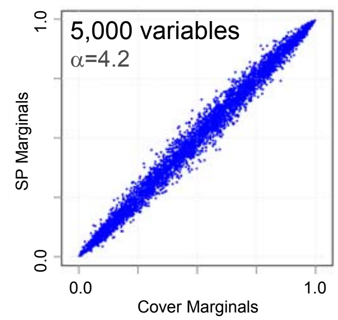
  - The cover factor graph has the same topology as the original.
- Applying standard BP to this modified factor graph, after some simplifications, yields the SP equations.
  - This construction shows that SP is an instance of the BP algorithm

SP must compute a loopy approximation to cover marginals

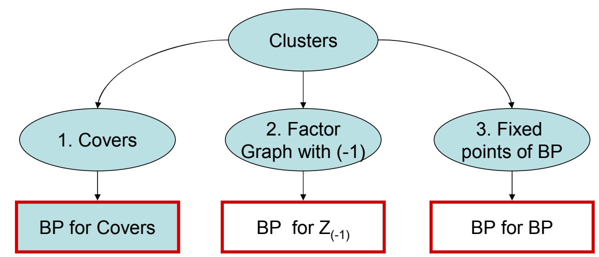
SP as BP on Covers: Results for SAT

Experiment:

1. sample many covers using local search in one large formula
2. compute cover magnetization from samples (x-axis)
3. compare with SP marginals (y-axis)



Belief Propagation for Clusters



BP with (-1)

- Recall that the number of clusters is very well approximated by

$$Z_{(-1)} = \sum_{\vec{y} \in DomExt^n} (-1)^{\#_e(\vec{y})} \prod_{\alpha} f_\alpha(\vec{y}_\alpha)$$

- This expression is in a form that is very similar to the standard partition function of the original problem, which we can approximate with BP.
- **$Z_{(-1)}$ can also be approximated with "BP"**: the factor graph remains the same, only the semantics is generalized:
  - Variables: $\vec{y} \in DomExt^n$

    $DomExt := \mathcal{P}(\{c_1, \dots, c_k\}) \setminus \emptyset$
  - Factors: $f_\alpha(\vec{y}_\alpha) = \min_{\vec{x}_\alpha \in \vec{y}_\alpha} f_\alpha(\vec{x}_\alpha)$
- And we need to adapt the BP equations to cope with (-1).

BP Adaptation for (-1)

- Standard BP equations can be derived as stationary point conditions for continuous constrained optimization problem (variational derivation).
- The BP adaptation for $Z_{(-1)}$ follows exactly the same path, and generalizes where necessary.
  - The following intermezzo goes through the derivation

One can derive a message passing algorithm for inference in factor graphs with (-1)

- We call this adaptation $BP_{(-1)}$

(Intermezzo: Deriving BP$_{(-1)}$)

1. We have a target function *p(y)* with real domain that is known up to a normalization constant and unknown marginals, and we seek trial function *b(y)* with known marginals to approximate *p(y)*

$$p(\vec{y}) := \frac{1}{Z_{(-1)}}(-1)^{\#_e(\vec{y})}\prod_\alpha f_\alpha(\vec{y}_\alpha)$$

To do this, we will search through a space of possible *b(y)* that have a special form, so that only polynomial number of parameters is needed. The parameters are marginal sums of *b(y)* for each variable and factor.

(Intermezzo: Deriving BP$_{(-1)}$)

2. The **standard** assumptions we have about *b(y)* are: (assumption is legitimate if the same condition holds for *p(y)*)

- Marginalization $b_i(y_i) = \sum_{\vec{y}_{-i}} b(\vec{y})$ and $b_\alpha(\vec{y}_\alpha) = \sum_{\vec{y}_{-\alpha}} b(\vec{y})$
  - Legitimate but not enforceable
- Normalization $\sum_{y_i} b_i(y_i) = \sum_{\vec{y}_\alpha} b_\alpha(\vec{y}_\alpha) = 1$
  - Legitimate, and explicitly enforced
- Consistency $\forall \alpha, i \in \alpha, y_i : b_i(y_i) = \sum_{\vec{y}_{\alpha \setminus i}} b_\alpha(\vec{y}_\alpha)$
  - Legitimate and explicitly enforced
- Tree-like decomposition (d_i is degree of variable *I*)

$$|b(\vec{y})| = \frac{\prod_\alpha |b_\alpha(\vec{y}_\alpha)|}{\prod_i |b_i(y_i)|^{d_i - 1}}$$

  - Not legitimate, and built-in

(Intermezzo: Deriving BP$_{(-1)}$)

Two **additional** assumptions are needed to deal with (-1)

- Sign-correspondence *b(y)* and *p(y)* have the same signs
  - Legitimate and built-in
- Sign-alternation $b_i(y_i)$ is negative iff $|y_i|$ is even, and $b_\alpha(y_\alpha)$ is negative iff $\#_e|y_\alpha|$ is odd
  - May or may not be legitimate, built-in

- The Sign-alternation assumption can be viewed as a application of inclusion-exclusion principle
  - Whether or not it is legitimate depends on the solutionspace of a particular problem.

Theorem: if a k-SAT problem has a k-simple solutionspace, then Sign-alternation is legitimate

(Intermezzo: Deriving BP$_{(-1)}$)

3. The Kullback-Leibler divergence
   - The function that is being minimized in BP derivation
   - Traditionally defined to measure difference between prob. Distributions
   - Need to generalize to allow for non-negative functions (with Sign-correspondence)

Lemma: Let b(.) and p(.) be (possibly negative) weight functions on the same domain. If they agree on signs and sum to the same constant, then the KL-divergence D(*b*||*p*) satisfies: D(*b*||*p*) ≥ 0 and =0 iff *b*≡*p* .

4. Minimizing D(*b*||*p*)
   - Writing *p(y)=sign(p(y))* |*p(y)*| and *p(y)=sign(b(y))* |*b(y)*| allows to isolate the signs and minimization follows analogous steps as in the standard BP
   - At the end, we implant the signs back using Sign-alternation assumption

The Resulting $BP_{(-1)}$

- The $BP_{(-1)}$ iterative equations:

The black part is exactly BP

$$n_{i\to\alpha}(y_i) \propto \prod_{\beta\ni i\setminus\alpha} m_{\beta\to i}(y_i)$$

$$m_{\alpha\to i}(y_i) \propto \sum_{\vec{y}_{\alpha\setminus i}\in DomExt^{|\alpha|-1}} f_\alpha(\vec{y}_\alpha) \prod_{j\in\alpha\setminus i} (-1)^{\delta(|y_j| \text{ is even})} n_{j\to\alpha}(y_j)$$

- The beliefs (estimates of marginals):

$$b_i(y_i) \propto (-1)^{\delta(|y_i| \text{ is even})} \prod_{\alpha\ni i} m_{\alpha\to i}(y_i)$$

$$b_\alpha(\vec{y}_\alpha) \propto (-1)^{\#_e(\vec{y}_\alpha)} f_\alpha(\vec{y}_\alpha) \prod_{i\in\alpha} n_{i\to\alpha}(y_i)$$

- The $Z_{BP(-1)}$ (the estimate of $Z_{(-1)}$):

$$\log Z_{BP_{(-1)}} := -\sum_\alpha \sum_{\vec{y}_\alpha} b_\alpha(\vec{y}_\alpha) \log |b_\alpha(\vec{y}_\alpha)| + \sum_i (d_i - 1) \sum_{y_i} b_i(y_i) \log |b_i(y_i)|$$

Relation of $BP_{(-1)}$ to SP

- **For SAT: $BP_{(-1)}$ is equivalent to SP**
  - The instantiation of the equations can easily be rewritten as SP equations
  - This is shown in the following intermezzo.
- **For COL: $BP_{(-1)}$ is NOT equivalent to SP**
  - $BP_{(-1)}$ estimates the total number of clusters
  - SP estimates the number of most numerous clusters
  - While **$BP_{(-1)}$** computes the total number of clusters (and thus the marginals of cluster backbones), it **does not perform well in decimation.**
    - It stops converging on the decimated problem
  - **SP**, focusing on computing "less information", **performs well in decimation**

(Intermezzo: $BP_{(-1)}$ for SAT is SP)

- Using a simple substitution one can rewrite the $BP_{(-1)}$ equations into a form equivalent with SP equations:

$$n_{i\to\alpha}(y_i) = \prod_{\beta\ni i\setminus\alpha} m_{\beta\to i}(y_i)$$

$$m_{\alpha\to i}(y_i) \propto \sum_{\vec{y}_{\alpha\setminus i}\in DomExt^{|\alpha|-1}} f_\alpha(\vec{y}_\alpha) \prod_{j\in\alpha\setminus i} (-1)^{\delta(|y_j| \text{ is even})} n_{j\to\alpha}(y_j)$$

  - y_i={T,F} means x_i=$*$
  - Move around the (-1) term: $\tilde{n}_{i\to\alpha}(x_i) := (-1)^{\delta(x_i=*)} n_{i\to\alpha}(x_i)$
  - Plug-in SAT factors:

$$m_{\alpha\to i}(x_i) = \begin{cases} 1 & \text{if } x_i = sign(\alpha,i) \\ 1 - \prod_{j\in\alpha\setminus i}(\tilde{n}_{j\to\alpha}(-sign(\alpha,j)) + \tilde{n}_{j\to\alpha}(*)) & \text{otherwise} \end{cases}$$

  - Define a message for "P[no other variable than *i* will satisfy α]"

$$\eta_{\alpha\to i} := \prod_{j\in\alpha\setminus i}(\tilde{n}_{j\to\alpha}(-sign(\alpha,j)) + \tilde{n}_{j\to\alpha}(*))$$

(Intermezzo: $BP_{(-1)}$ for SAT is SP)

- Define messages (analog of '*n*' messages) to denote, reps., *i* is forced to satisfy α, *i* is forced to unsatisfy α, and *i* is not forced either way:

$$\pi^s_{i\to\alpha} := \tilde{n}_{i\to\alpha}(sign(\alpha,i)) + \tilde{n}_{i\to\alpha}(*)$$
$$\pi^u_{i\to\alpha} := \tilde{n}_{i\to\alpha}(-sign(\alpha,i)) + \tilde{n}_{i\to\alpha}(*)$$
$$\pi^*_{i\to\alpha} := -\tilde{n}_{i\to\alpha}(*)$$

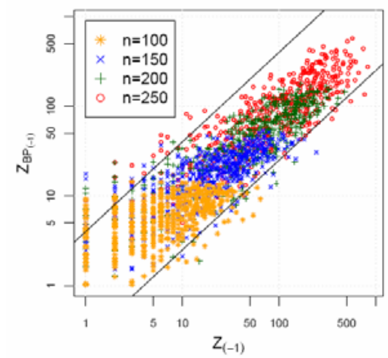
- Putting it together, we get the **SP equations**:

$$\eta_{\alpha\to i} = \prod_{j\in\alpha\setminus i} \frac{\pi^u_{j\to\alpha}}{\pi^u_{j\to\alpha}+\pi^s_{j\to\alpha}+\pi^*_{j\to\alpha}}$$
$$\pi^u_{i\to\alpha} = \left[1 - \prod_{\beta\in V^u_\alpha(i)}(1-\eta_{\beta\to i})\right] \prod_{\beta\in V^s_\alpha(i)}(1-\eta_{\beta\to i})$$
$$\pi^s_{i\to\alpha} = \left[1 - \prod_{\beta\in V^s_\alpha(i)}(1-\eta_{\beta\to i})\right] \prod_{\beta\in V^u_\alpha(i)}(1-\eta_{\beta\to i})$$
$$\pi^*_{i\to\alpha} = \prod_{\beta\ni i\setminus\alpha}(1-\eta_{\beta\to i})$$

$BP_{(-1)}$: Results for SAT

Experiment: approximating $Z_{(-1)}$

1. count exact $Z_{(-1)}$ for many small formulas at α=4.0 (x-axis)
2. compare with $BP_{(-1)}$'s estimate of partition function $Z_{BP(-1)}$ (y-axis)



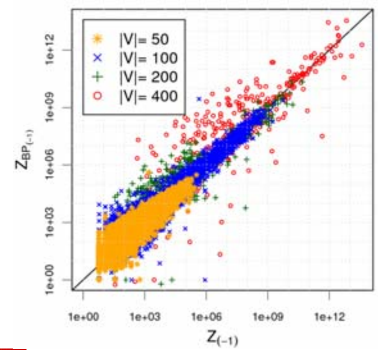
Plot is on log-log scale
The lines are *y=4x* and *y=¼x*

The estimate is good only for $\alpha \geq 3.9$ It is =1 for lower ratios.

$BP_{(-1)}$: Results for COL

Experiment: approximating $Z_{(-1)}$

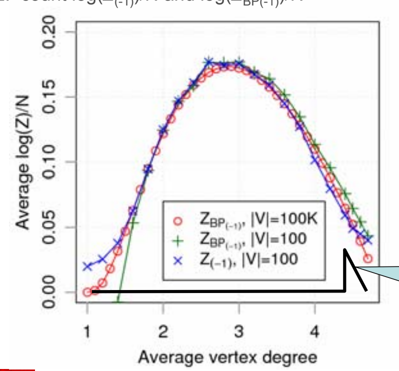
1. count exact $Z_{(-1)}$ for many small graphs with avg.deg.$\in$[1,4.7] (x-axis)
2. compare with $BP_{(-1)}$'s estimate of partition function $Z_{BP(-1)}$ (y-axis)



$BP_{(-1)}$: Results for COL

Experiment: rescaling number of clusters and $Z_{(-1)}$

1. for graphs with various average degrees (x-axis)
2. count $\log(Z_{(-1)})/N$ and $\log(Z_{BP(-1)})/N$ (y-axis)



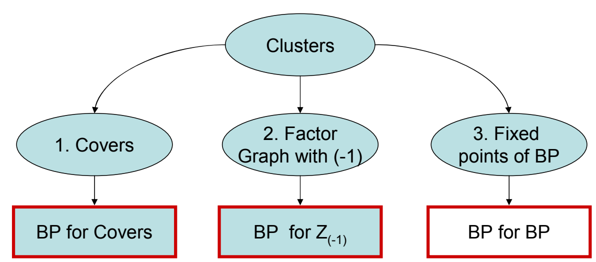
The rescaling assumes that #clusters=exp(N Σ(c))

Σ(c) is so called **complexity** and is instrumental in various physics-inspired approaches to cluster counting (will see later)

Sketch of SP results:

Nonzero only between 4.42 and 4.69

Belief Propagation for Clusters



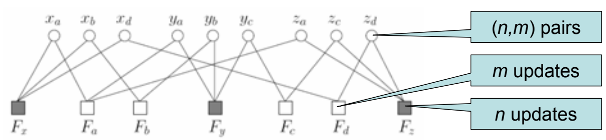
BP for Fixed Points of BP

- The task of finding fixed points of BP equations:

$$n_{i\to\alpha}(x_i) \;\propto\; \prod_{\beta\ni i\setminus\alpha} m_{\beta\to i}(x_i)$$

$$m_{\alpha\to i}(x_i) \;\propto\; \sum_{\vec{x}_{\alpha\setminus i}\in Dom^{|\alpha|-1}} f_\alpha(\vec{x}_\alpha)\prod_{j\in\alpha\setminus i} n_{j\to\alpha}(x_j)$$

  can be cast as finding solutions for a **constrained problem** (the equations) with **continuous variables** (the messages)
- One can thus construct a factor graph with continuous variables for the problem. Its partition function *Z* is the number of fixed points of BP.
  - The factor graph is topologically equivalent with the one for covers (WP).



BP for Fixed Points of BP

- The new BP messages *N((n,m))* and *M((n,m))* are now **functions on continuous domains**
  - The sum in the update rule is replaced by an integral
- To make the new equations computationally tractable, we can **discretize the values of *n* and *m*** to {0,1,∗} as follows:
  - If the value is 0 or 1, the discretized value is also **0** or **1**
  - If the value is ∈(0,1), the discretized value is **∗**
- We can still recover some information about **cluster backbones**
  - $m_{\alpha\to i}(v_i)$=**1**: x_i is a v_i-backbone, according to α, in a BP fixed point.
  - $m_{\alpha\to i}(v_i)$=**∗**: x_i is not a v_i-backbone, according to α, in a BP fixed point.

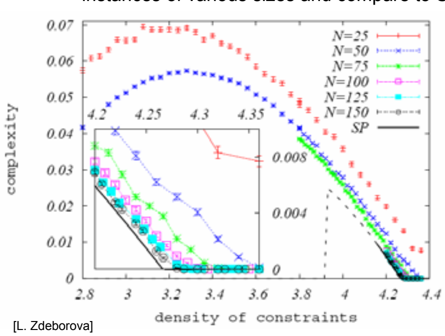
> "BP for fixed points of discretized BP" computes
> the fraction of fixed points where x_i is a v_i-backbone.

- **This leads to** equations analogous to Warning Propagation, and thus to **SP** through the same path as for covers.

BP for BP: Results for SAT

Experiment: counting number of solution clusters with SP

1. random 3-SAT for various α (x-axis)
2. compute avg. complexity (log(#clusters)/N) for 'median' instances of various sizes and compare to SP (y-axis)



[L. Zdeborova]

The plot is smooth because only median (out of 999) instances are considered.

Coming up Next….

- Reasoning about clusters on solutonspaces of random problems can be done efficiently with BP
  - But what is it all good for?
  - Can BP be used for more practical problems?
- We will show how extensions of these techniques can be used to finely trace changes in solutionspace geometry for large random problems
- We will show how BP can be utilized to approximate and bound solution counts of various real-world problems.

Advanced Topics

Tutorial Outline

1. **Introduction**
2. **Probabilistic inference using message passing**
3. **Survey Propagation**
4. **Solution clusters**
5. **Probabilistic inference for clusters**
6. **Advanced topics**

- Clustering in solutionspace of random problems
- Solution counting with BP

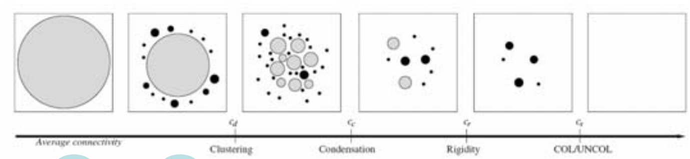
Understanding Solution Clusters

Solution cluster related concepts

- **Dominating clusters**:
  a minimal set of clusters that contains "almost all" solutions
  - How many dominating clusters are there? Exponentially many? Constant?
- **Frozen/backbone variable** v in a cluster C:
  v takes only one value in all solutions in C
  - Do clusters have frozen variables?
- **Frozen cluster** C:
  a constant fraction of variables in C are frozen
  - The key quantity estimated by SP! *(how many clusters have x frozen to T?)*

Cluster Structure of Random CSPs

k-COL problems, with increasing graph density ("connectivity")



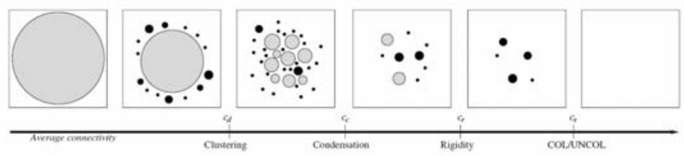
1 2

[F.Krzakala, et al.]

1. Very low density:
   - all solutions essentially in a single dominating cluster
   - the dominating cluster is not frozen
2. Low density:
   - a single large cluster still dominates (i.e., has "almost all" solutions)
   - small frozen clusters appear

Cluster Structure of Random CSPs

k-COL problems, with increasing graph density

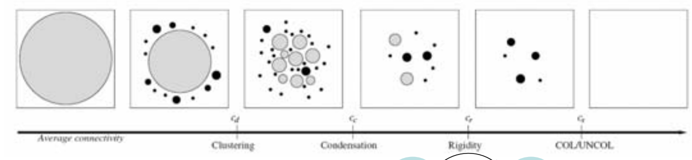


3

3. Dynamical/clustering transition threshold c_d
   - solutions split into an exponential number of small dominating clusters
   - no single giant dominating cluster
   - some clusters are frozen, some are not

Cluster Structure of Random CSPs

k-COL problems, with increasing graph density

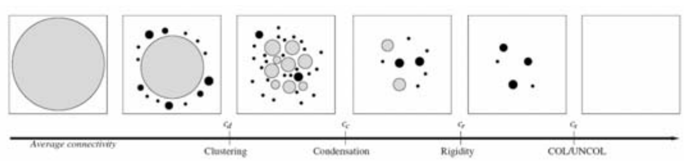


4 5

4. Condensation transition threshold c_c
   - solutions condense into a constant number of small dominating clusters
   - some clusters are frozen, some are not
5. Rigidity transition threshold c_r
   - all dominating clusters are frozen
   - could occur before or after condensation

Cluster Structure of Random CSPs

k-COL problems, with increasing graph density



6

6. Satisfiability/colorability transition threshold c_s
   - all solutions disappear

Where do these transitions come from?

The "Sigma vs. s curve"

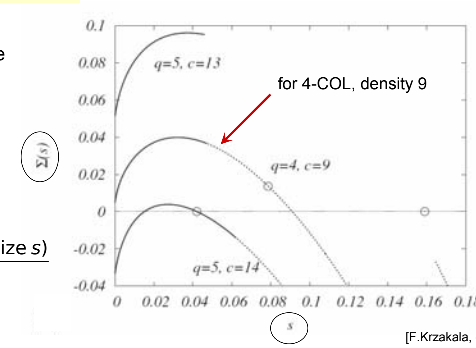
Histogram of clusters, grouped by cluster size

s: Normalized cluster size

$$s = \frac{\log(\text{cluster size})}{N}$$

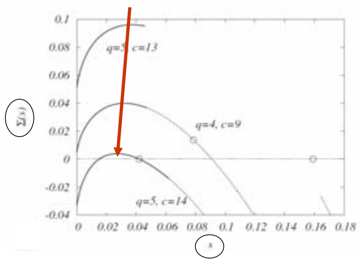
Σ: Complexity

$$\Sigma(s) = \frac{\log(\#\text{clusters of size } s)}{N}$$



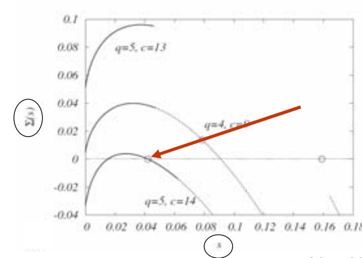
[F.Krzakala, et al.]

Satisfiability Threshold



- ◻ Clusters size occurring most often:
  - Select a cluster uniformly at random, what size does it have?
  - Corresponds to slope=0 in the plot.
  - Standard SP works with these clusters.
- ◻ Is used to predict the satisfiability threshold c_s
  - *When the point with slope=0 goes below zero*

Condensation Threshold



- ◻ # of solutions in clusters of (norm.) size *s* = $e^{Ns}e^{N\Sigma(s)} = \boxed{e^{N(s+\Sigma(s))}}$
- ◻ Cluster size representing most solutions:
  - Select a solution uniformly at random, what is the size of the cluster it is in?
  - Corresponds to slope=1 in the plot.
- ◻ Is used to predict the condensation threshold c_c

Tutorial Outline

1. **Introduction**
2. **Probabilistic inference using message passing**
3. **Survey Propagation**
4. **Solution clusters**
5. **Probabilistic inference for clusters**
6. **Advanced topics**

- ◻ Clustering in solutionspace of random problems
- ◻ Solution counting with BP

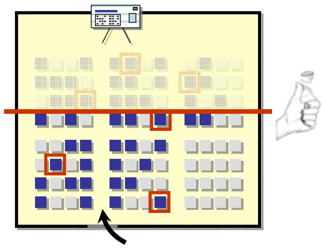
Solution counting with BP

- ◻ **Solution counting / Model counting** is a problem with far reaching consequences and intensive research activity.
- ◻ BP by itself is able to estimate the value of the partition function, i.e. the number of solutions of a CSP:

$$Z := \sum_{\vec{x}} \prod_{\alpha} f_\alpha(\vec{x}_\alpha)$$

- ◻ **Issue:** Just like in the case with finding a solution, where doing Maximum Likelihood itself did not work well (we needed decimation), this estimate is not robust.
- ◻ **Way out:** Use decimation again, keeping track of some additional information that will be used to construct a more robust estimate.

Solution Count Estimation with Guarantees



Idea:

- Identify a "balanced" row split or column split (roughly equal number of solutions on each side)
  - Use marginals for estimate
- Pick one side *at random*
- Count on that side recursively
- Multiply result by 2

[similar to decimation]

This provably yields the true count on average!

- Even when an unbalanced row/column is picked accidentally for the split, e.g. even when marginals are incorrect
- Provides probabilistic correctness guarantees on the estimated count
- Surprisingly good in practice

Experiments: BPCount

Always very fast!

Sometimes better

Sometimes not

Instance	# of vars	True Count (if known)	SampleCount (99% confidence) LWR-bound	SampleCount Time	BPCount (99% confidence) LWR-bound	BPCount Time
CIRCUIT SYNTH.						
2bitmax_6	252	2.1×10^{29}	$\geq 2.4 \times 10^{28}$	29 sec	$\geq 2.8 \times 10^{28}$	5 sec
RANDOM *k*-CNF						
wff-3-3.5	150	1.4×10^{14}	$\geq 1.6 \times 10^{13}$	4 min	$\geq 1.6 \times 10^{11}$	3 sec
wff-3-1.5	100	1.8×10^{21}	$\geq 1.6 \times 10^{20}$	4 min	$\geq 1.0 \times 10^{20}$	1 sec
wff-4-5.0	100	—	$\geq 8.0 \times 10^{15}$	2 min	$\geq 2.0 \times 10^{15}$	2 sec
LATIN SQUARE						
ls8-norm	301	5.4×10^{11}	$\geq 3.1 \times 10^{10}$	19 min	$\geq 1.9 \times 10^{10}$	12 sec
ls9-norm	456	3.8×10^{17}	$\geq 1.4 \times 10^{15}$	32 min	$\geq 1.0 \times 10^{16}$	11 sec
ls10-norm	657	7.6×10^{24}	$\geq 2.7 \times 10^{21}$	49 min	$\geq 1.0 \times 10^{23}$	22 sec
ls11-norm	910	5.4×10^{33}	$\geq 1.2 \times 10^{30}$	69 min	$\geq 6.4 \times 10^{30}$	1 min
ls12-norm	1221	—	$\geq 6.9 \times 10^{37}$	50 min	$\geq 2.0 \times 10^{41}$	70 sec
ls13-norm	1596	—	$\geq 3.0 \times 10^{49}$	67 min	$\geq 4.0 \times 10^{54}$	6 min
ls14-norm	2041	—	$\geq 9.0 \times 10^{60}$	44 min	$\geq 1.0 \times 10^{67}$	4 min
LANGFORD PROBS.						
lang-2-12	576	1.0×10^{5}	$\geq 4.3 \times 10^{3}$	32 min	$\geq 2.3 \times 10^{3}$	50 sec
lang-2-15	1024	3.0×10^{7}	$\geq 1.0 \times 10^{6}$	60 min	$\geq 5.5 \times 10^{5}$	1 min
lang-2-16	1024	3.2×10^{8}	$\geq 1.0 \times 10^{6}$	65 min	$\geq 3.2 \times 10^{5}$	1 min
lang-2-19	1444	2.1×10^{11}	$\geq 3.3 \times 10^{9}$	62 min	$\geq 4.7 \times 10^{7}$	26 min
lang-2-20	1600	2.6×10^{12}	$\geq 5.8 \times 10^{9}$	54 min	$\geq 7.1 \times 10^{4}$	22 min
lang-2-23	2116	3.7×10^{15}	$\geq 1.6 \times 10^{11}$	85 min	$\geq 1.5 \times 10^{5}$	15 min
lang-2-24	2304	—	$\geq 4.1 \times 10^{13}$	80 min	$\geq 8.9 \times 10^{7}$	18 min

Tutorial Outline

1. **Introduction**
2. **Probabilistic inference using message passing**
3. **Survey Propagation**
4. **Solution clusters**
5. **Probabilistic inference for clusters**
6. **Advanced topics**

Some Research Directions

- Challenge for computer scientists:
  Can one design an algorithm that is better than SP for random CSPs?
- Challenge posed by statistical physicists:
  Can one solve random CSPs beyond the rigidity threshold, c_r?
- Can the properties of solution clusters in large structured problems be efficiently estimated?
- What other interesting solutionspace structures (other than clusters) can be identified and used for solving problems?
- Can the fast probabilistic methods be combined with complete DPLL-style solution technique to create efficient hybrid methods?

Selected References

- Factor Graphs and Belief Propagation:
  - F. R. Kschischang, B. J. Frey, and H. A. Loeliger. **Factor graphs and the sum-product algorithm.** *IEEE Transactions on Information Theory*, 47(2):498{519, 2001.
  - J. S. Yedidia, W. T. Freeman, and Y. Weiss. **Constructing free-energy approximations and generalized belief propagation algorithms.** *IEEE Transactions on Information Theory*, 51(7):2282–2312, 2005.
- Variants of BP:
  - Yuille, A. L. **CCCP algorithms to minimize the Bethe and Kikuchi free energies: Convergent alternatives to belief propagation.** *Neural Comput.* 14(7):1691–1722.
  - M. J. Wainwright, T. Jaakkola and A. S. Willsky. **Tree-reweighted belief propagation and approximate ML estimation by pseudo-moment matching.** *9th Workshop on Artificial Intelligence and Statistics*, Key West, Florida; January, 2003
  - E. I. Hsu and S. A. McIlraith. **Characterizing propagation methods for boolean satisfiabilty.** In *9th SAT*, vol. 4121 of *LNCS*, pages 325.338, Seattle,WA, August 2006.

Selected References

- Survey Propagation:
  - M. Mezard, G. Parisi, and R. Zecchina. **Analytic and Algorithmic Solution of Random Satisfiability Problems.** *Science*, 297(5582):812.815, 2002.
  - A. Braunstein and R. Zecchina. **Survey propagation as local equilibrium equations.** *J. Stat. Mech.*, P06007, 2004.
  - A. Braunstein, R. Mulet, A. Pagnani, M. Weigt, and R. Zecchina. **Polynomial iterative algorithms for coloring and analyzing random graphs.** *Physical Review E*, 68:036702, 2003.
- Covers and $Z_{(-1)}$:
  - D. Achlioptas and F. Ricci-Tersenghi. **On the solution-space geometry of random constraint satisfaction problems.** In *38th STOC*, pages 130.139, Seattle,WA, 2006.
  - E. N. Maneva, E. Mossel, and M. J. Wainwright. **A new look at survey propagation and its generalizations**. *In 16th SODA*, pp. 1089-1098, Vancouver, Canada, 2005.
  - L. Kroc, A. Sabharwal, and B. Selman. **Survey propagation revisited.** In *23rd UAI*, Vancouver, July 2007.
  - L. Kroc, A. Sabharwal, and B. Selman. **Counting solution clusters using Belief Propagation.** *Submitted to 22nd NIPS*, 2008.

Selected References

- Solutionspace structure of random CSPs:
  - S. Kirkpatrick and B. Selman. **Critical behavior in the satisfiability of random boolean expressions**, *Science* 264 (1994), 1297-1301.
  - F. Krzakala, A. Montanari, F. Ricci-Tersenghi, G. Semerjian and L. Zdeborova. **Gibbs states and the set of solutions of random constraint satisfaction problems,** *Proc. Natl. Acad. Sci.* 104, 10318, 2007.

CORNELL

Thank you for attending!

Tutorial slides: http://www.cs.cornell.edu/~sabhar/tutorials/AAAI08-BPSP