# Relation between clustering and algorithmic phase transitions in the random k-XORSAT model and its NP-complete extensions

Fabrizio Altarelli, Rémi Monasson and Francesco Zamponi*

*Service de Physique Théorique, Orme des Merisiers, CEA Saclay, 91191 Gif-sur-Yvette Cedex, France

September 17, 2007

# Outline

# Introduction

## Random Constraint Satisfaction Problem

### Constraint Satisfaction Problem (CSP):

- Set of $N$ variables $\{x_1, \cdots, x_N\}$
- Set of $M = \alpha N$ constraints $\{c_1, \cdots, c_M\}$

### Example ($k$-SAT)

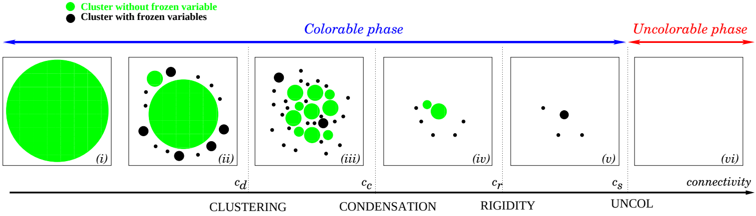
- $x_i \in \{0, 1\}$ Boolean variables
- $c_a : \{x_{i_1}, \cdots, x_{i_k}\} \neq \{z_1, \cdots, z_k\}$

### Random CSP:

Pick the constraints at random; e.g. uniformly at fixed $k$, $\alpha$

# Introduction
## Phase transitions in Random CSP

Structure of the solutions:




Phase transitions in Random CSP (see Krzakala's talk):

- $\alpha < \alpha_d$: Most of the solutions form a unique cluster
- $\alpha_d < \alpha < \alpha_c$: The solutions form many ($\sim e^{N\Sigma}$) clusters
- $\alpha_c < \alpha < \alpha_r$: A small number of clusters dominate
- $\alpha_r < \alpha < \alpha_s$: Frozen variables in a cluster
- $\alpha > \alpha_s$: No solutions (UNSAT)

$\alpha_d, \alpha_c, \alpha_r, \alpha_s$: Discontinuous jump of an "order parameter"

# Introduction

## Algorithmic phase transitions

Consider an algorithm attempting to find a solution

1. Decimation algorithm (DPLL, SP, ...); fixes sequentially all the variables
   $P_{sol}(\alpha)$= probability to find a solution:
   $$\lim_{N\to\infty} P_{sol}(\alpha) = \begin{cases} p > 0 & \alpha < \alpha_a \\ 0 & \alpha > \alpha_a \end{cases}$$
2. Local search algorithm (Walk-SAT, ...): samples configurations
   $T_{sol}(\alpha)$= average time to find a solution:
   $$T_{sol}(\alpha) \sim \begin{cases} N & \alpha < \alpha_a \\ e^{\gamma N} & \alpha > \alpha_a \end{cases}$$
   (similarly for decimation algorithms with backtracking)

Is $\alpha_a$ related to a static phase transition?

# Introduction

## Algorithmic phase transitions

Consider an algorithm attempting to find a solution

1. Decimation algorithm (DPLL, SP, ...); fixes sequentially all the variables
   $P_{sol}(\alpha)$= probability to find a solution:
   $$\lim_{N\to\infty} P_{sol}(\alpha) = \begin{cases} p > 0 & \alpha < \alpha_a \\ 0 & \alpha > \alpha_a \end{cases}$$
2. Local search algorithm (Walk-SAT, ...): samples configurations
   $T_{sol}(\alpha)$= average time to find a solution:
   $$T_{sol}(\alpha) \sim \begin{cases} N & \alpha < \alpha_a \\ e^{\gamma N} & \alpha > \alpha_a \end{cases}$$
   (similarly for decimation algorithms with backtracking)

Is $\alpha_a$ related to a static phase transition?

# Introduction

## Algorithmic phase transitions

Consider an algorithm attempting to find a solution

1. Decimation algorithm (DPLL, SP, ...); fixes sequentially all the variables
   $P_{sol}(\alpha)=$ probability to find a solution:
   $$\lim_{N\to\infty} P_{sol}(\alpha) = \begin{cases} p > 0 & \alpha < \alpha_a \\ 0 & \alpha > \alpha_a \end{cases}$$
2. Local search algorithm (Walk-SAT, ...): samples configurations
   $T_{sol}(\alpha)=$ average time to find a solution:
   $$T_{sol}(\alpha) \sim \begin{cases} N & \alpha < \alpha_a \\ e^{\gamma N} & \alpha > \alpha_a \end{cases}$$
   (similarly for decimation algorithms with backtracking)

Is $\alpha_a$ related to a static phase transition?

# Introduction

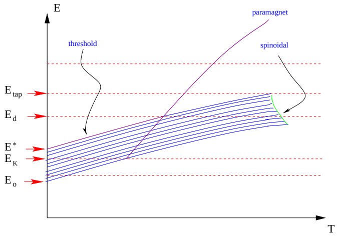
## Relation between static and algorithmic PT: fully connected

Sometimes yes...

### Fully connected spherical $p$-spin model

$$H = \sum_{i_1,\cdots,i_p} J_{i_1,\cdots,i_p} \sigma_{i_1} \cdots \sigma_{i_p} \ , \qquad \sum_i \sigma_i^2 = N$$

1. Clustering and dynamical equilibrium transitions at $T_d$
2. Find a configuration of energy $E$: clustering and algorithmic transitions at $E^*$




But already at the mean field level the situation is unclear for more complicated models

# Introduction

## Relation between static and algorithmic PT: diluted

For $k$-XORSAT Montanari and Semerjian proved the existence of a dynamical transition at temperature close to the clustering temperature

- No other rigorous or analytical results are known (to me)
- Many conjectures based on numerical results
- What it the role of the freezing transition? (Semerjian, arXiv:0705.2147; Kurchan, Krzakala, arXiv:cond-mat/0702546)

### Our result

We will discuss the relation between clustering transition and algorithmic transition for a class of decimation algorithms (DPLL) for XORSAT; we will show that $\alpha_a \leq \alpha_d$, i.e. these algorithms cannot find solutions in polynomial time in the clustered phase

# Introduction

## Relation between static and algorithmic PT: diluted

For $k$-XORSAT Montanari and Semerjian proved the existence of a dynamical transition at temperature close to the clustering temperature

- No other rigorous or analytical results are known (to me)
- Many conjectures based on numerical results
- What it the role of the freezing transition? (Semerjian, arXiv:0705.2147; Kurchan, Krzakala, arXiv:cond-mat/0702546)

**Our result**

We will discuss the relation between clustering transition and algorithmic transition for a class of decimation algorithms (DPLL) for XORSAT; we will show that $\alpha_a \leq \alpha_d$, i.e. these algorithms cannot find solutions in polynomial time in the clustered phase

# Random UE-CSP

$\rightarrow x_i \in \{0, \cdots, d-1\}; \; i = 1, \cdots, N.$

$\rightarrow$ *Uniquely Extendible (UE) Constraint* on $x_1, \cdots, x_k$:
if one fixes a subset of $k-1$ variables, there is only one value of the remaining variable that satisfies the constraint (clause)

$\rightarrow$ *Random $(k, d)$-UE-CSP formula*: A collection of $M = \alpha N$ different UE constraints taken uniformly at random

**Example**

For $d = 2$, $x_i \in 0, 1$ and $x_1 + \cdots + x_k = 0$ (XORSAT)

Note: for $d = 2$ XORSAT is in P; but...

**Theorem (Connamacher-Malloy)**

$(3, 4)$-*UE-CSP is NP-Complete*

# Random UE-CSP

$\rightarrow x_i \in \{0, \cdots, d-1\}$; $i = 1, \cdots, N$.

$\rightarrow$ *Uniquely Extendible (UE) Constraint* on $x_1, \cdots, x_k$:
if one fixes a subset of $k-1$ variables, there is only one value of the remaining variable that satisfies the constraint (clause)

$\rightarrow$ *Random $(k, d)$-UE-CSP formula*: A collection of $M = \alpha N$ different UE constraints taken uniformly at random

**Example**

For $d = 2$, $x_i \in 0, 1$ and $x_1 + \cdots + x_k = 0$ (XORSAT)

Note: for $d = 2$ XORSAT is in P; but...

**Theorem (Connamacher-Malloy)**

*$(3, 4)$-UE-CSP is NP-Complete*

# Random UE-CSP

$\rightarrow x_i \in \{0, \cdots, d-1\};\ i = 1, \cdots, N.$

$\rightarrow$ *Uniquely Extendible (UE) Constraint* on $x_1, \cdots, x_k$:
if one fixes a subset of $k-1$ variables, there is only one value of the remaining variable that satisfies the constraint (clause)

$\rightarrow$ *Random $(k, d)$-UE-CSP formula*: A collection of $M = \alpha N$ different UE constraints taken uniformly at random

**Example**

For $d = 2$, $x_i \in 0, 1$ and $x_1 + \cdots + x_k = 0$ (XORSAT)

Note: for $d = 2$ XORSAT is in P; but...

**Theorem (Connamacher-Malloy)**

$(3, 4)$-*UE-CSP is NP-Complete*

# Search algorithms

## Definition

We consider a class of simple algorithms acting on a formula in an attempt to find solutions.

**Algorithm (DPLL)**

At each time step assign a variable according to the following rules:

- If there is at least one clause of length one then satisfy it (Unit Propagation)
- Else, choose a variable according to some heuristic rule and assign it at random

**Example**

$$\begin{cases} x_1 + x_2 + x_3 = 0 \\ x_1 + x_2 + x_4 = 1 \end{cases}$$

# Search algorithms

## Definition

We consider a class of simple algorithms acting on a formula in an attempt to find solutions.

### Algorithm (DPLL)

At each time step assign a variable according to the following rules:

- If there is at least one clause of length one then satisfy it (Unit Propagation)
- Else, choose a variable according to some heuristic rule and assign it at random

### Example

$$\begin{cases} x_1 + x_2 + x_3 = 0 \\ x_1 + x_2 + x_4 = 1 \end{cases}$$

# Search algorithms
## Definition

We consider a class of simple algorithms acting on a formula in an attempt to find solutions.

### Algorithm (DPLL)

At each time step assign a variable according to the following rules:

- If there is at least one clause of length one then satisfy it (Unit Propagation)
- Else, choose a variable according to some heuristic rule and assign it at random

### Example

$$\begin{cases} x_1 + x_2 + x_3 = 0 \\ x_1 + x_2 + x_4 = 1 \end{cases} \qquad \text{assign } x_2 = 0$$

# Search algorithms
## Definition

We consider a class of simple algorithms acting on a formula in an attempt to find solutions.

### Algorithm (DPLL)

At each time step assign a variable according to the following rules:

- If there is at least one clause of length one then satisfy it (Unit Propagation)
- Else, choose a variable according to some heuristic rule and assign it at random

### Example

$$\begin{cases} x_1 + x_3 = 0 \\ x_1 + x_4 = 1 \end{cases}$$

assign $x_3 = 1$

# Search algorithms

## Definition

We consider a class of simple algorithms acting on a formula in an attempt to find solutions.

### Algorithm (DPLL)

At each time step assign a variable according to the following rules:

- If there is at least one clause of length one then satisfy it (Unit Propagation)
- Else, choose a variable according to some heuristic rule and assign it at random

### Example

$$\begin{cases} x_1 = 1 \\ x_1 + x_4 = 1 \end{cases}$$

UP $x_1 = 1,\ x_4 = 0$

# Search algorithms
## Heuristics

After $T$ iterations, $C_j(T)$ = number of clauses of length $j$.
$\mathcal{C} = \{C_j\}_{j=1,\cdots,k}$.

We consider the following heuristic: pick a variable from a clause of length $j$ with probability $p_j(\mathcal{C})$, or completely at random; $\sum_{j=1}^{k} p_j \leq 1$
Unit Propagation implies that if $C_1 \neq 0 \rightarrow p_j = \delta_{1j}$

**Example**

- Unit Clause (UC): pick variables uniformly at random
- Generalized Unit Clause (GUC): always pick variables from the shortest clauses

This class of heuristics preserves the Poissonian distribution of variable occurrencies

# Search algorithms

## Heuristics

After $T$ iterations, $C_j(T) =$ number of clauses of length $j$.
$\mathcal{C} = \{C_j\}_{j=1,\cdots,k}$.

We consider the following heuristic: pick a variable from a clause of length $j$ with probability $p_j(\mathcal{C})$, or completely at random; $\sum_{j=1}^{k} p_j \leq 1$

Unit Propagation implies that if $C_1 \neq 0 \rightarrow p_j = \delta_{1j}$

**Example**

- Unit Clause (UC): pick variables uniformly at random
- Generalized Unit Clause (GUC): always pick variables from the shortest clauses

This class of heuristics preserves the Poissonian distribution of variable occurrencies

# Search algorithms
## Heuristics

After $T$ iterations, $C_j(T) =$ number of clauses of length $j$.
$\mathcal{C} = \{C_j\}_{j=1,\cdots,k}$.

We consider the following heuristic: pick a variable from a clause of length $j$ with probability $p_j(\mathcal{C})$, or completely at random; $\sum_{j=1}^{k} p_j \leq 1$
Unit Propagation implies that if $C_1 \neq 0 \rightarrow p_j = \delta_{1j}$

**Example**

- Unit Clause (UC): pick variables uniformly at random
- Generalized Unit Clause (GUC): always pick variables from the shortest clauses

This class of heuristics preserves the Poissonian distribution of variable occurrencies

# Search algorithms
## Heuristics

After $T$ iterations, $C_j(T)$ = number of clauses of length $j$.
$\mathcal{C} = \{C_j\}_{j=1,\cdots,k}$.

We consider the following heuristic: pick a variable from a clause of length $j$ with probability $p_j(\mathcal{C})$, or completely at random; $\sum_{j=1}^{k} p_j \leq 1$
Unit Propagation implies that if $C_1 \neq 0 \rightarrow p_j = \delta_{1j}$

### Example

- Unit Clause (UC): pick variables uniformly at random
- Generalized Unit Clause (GUC): always pick variables from the shortest clauses

This class of heuristics preserves the Poissonian distribution of variable occurrencies

# Search algorithms
## Differential equations

We wish to analyze the performance of the algorithm on a random instance as function of $\alpha$ for $N \to \infty$
The average dynamics is described by a set of differential equations for $c_j(t) = \langle C_j(Nt) \rangle / N$:

$$\dot{c}_j = \frac{(j+1)c_{j+1} - jc_j}{1-t} - \rho_j(t)$$

$$\rho_j(t) = \lim_{\Delta T \to \infty} \lim_{N \to \infty} \frac{1}{\Delta T} \sum_{T=tN}^{tN+\Delta T-1} (p_j - p_{j+1})$$

It is easy to show that $\sum_{j=1}^{k} \rho_j \leq \sum_{j=1}^{k} j\rho_j \leq 1$

One has to solve these equations with initial datum $c_j(0) = \alpha\delta_{jk}$ to obtain $c_j(t)$; then

1. if at some time $c_2(t)/(1-t) = 1/2 \Rightarrow$ contradiction, $\alpha > \alpha_a$
2. else, at some time $c_j(t) = 0$ $\Rightarrow$ solution, $\alpha < \alpha_a$

# Search algorithms
## Differential equations

We wish to analyze the performance of the algorithm on a random instance as function of $\alpha$ for $N \to \infty$
The average dynamics is described by a set of differential equations for $c_j(t) = \langle C_j(Nt) \rangle / N$:

$$\dot{c}_j = \frac{(j+1)c_{j+1} - jc_j}{1-t} - \rho_j(t)$$

$$\rho_j(t) = \lim_{\Delta T \to \infty} \lim_{N \to \infty} \frac{1}{\Delta T} \sum_{T=tN}^{tN+\Delta T-1} (p_j - p_{j+1})$$

It is easy to show that $\sum_{j=1}^{k} \rho_j \leq \sum_{j=1}^{k} j\rho_j \leq 1$

One has to solve these equations with initial datum $c_j(0) = \alpha\delta_{jk}$ to obtain $c_j(t)$; then

1. if at some time $c_2(t)/(1-t) = 1/2 \Rightarrow$ contradiction, $\alpha > \alpha_a$
2. else, at some time $c_j(t) = 0 \Rightarrow$ solution, $\alpha < \alpha_a$

# Search algorithms

## Differential equations

We wish to analyze the performance of the algorithm on a random instance as function of $\alpha$ for $N \to \infty$

The average dynamics is described by a set of differential equations for $c_j(t) = \langle C_j(Nt) \rangle / N$:

$$\dot{c}_j = \frac{(j+1)c_{j+1} - jc_j}{1-t} - \rho_j(t)$$

$$\rho_j(t) = \lim_{\Delta T \to \infty} \lim_{N \to \infty} \frac{1}{\Delta T} \sum_{T=tN}^{tN+\Delta T-1} (p_j - p_{j+1})$$

It is easy to show that $\sum_{j=1}^{k} \rho_j \leq \sum_{j=1}^{k} j\rho_j \leq 1$

One has to solve these equations with initial datum $c_j(0) = \alpha\delta_{jk}$ to obtain $c_j(t)$; then

1. if at some time $c_2(t)/(1-t) = 1/2 \Rightarrow$ contradiction, $\alpha > \alpha_a$
2. else, at some time $c_j(t) = 0 \Rightarrow$ solution, $\alpha < \alpha_a$

# Phase transitions
## Leaf removal

If a variable occurs only in one clause ⇒ the clause can always be satisfied

**Algorithm (Leaf Removal)**

1. Remove one of the clauses containing a uniquely occurring (UO) variable
2. If (there are UO variables) goto 1; else stop

This procedure corresponds to eliminate subsequently the leaves in the factor graph representing the formula.

**Example**

$$\begin{cases} x_1 + x_2 + x_3 = 0 \\ x_1 + x_2 + x_4 = 1 \end{cases}$$ eliminate the first clause ($x_3$)

# Phase transitions

## Leaf removal

If a variable occurs only in one clause $\Rightarrow$ the clause can always be satisfied

### Algorithm (Leaf Removal)

1. Remove one of the clauses containing a uniquely occurring (UO) variable
2. If (there are UO variables) goto 1; else stop

This procedure corresponds to eliminate subsequently the leaves in the factor graph representing the formula.

### Example

$$\begin{cases} x_1 + x_2 + x_3 = 0 \\ x_1 + x_2 + x_4 = 1 \end{cases}$$

eliminate the first clause ($x_3$)

# Phase transitions
## Leaf removal

If a variable occurs only in one clause ⇒ the clause can always be satisfied

**Algorithm (Leaf Removal)**

1. Remove one of the clauses containing a uniquely occurring (UO) variable
2. If (there are UO variables) goto 1; else stop

This procedure corresponds to eliminate subsequently the leaves in the factor graph representing the formula.

**Example**

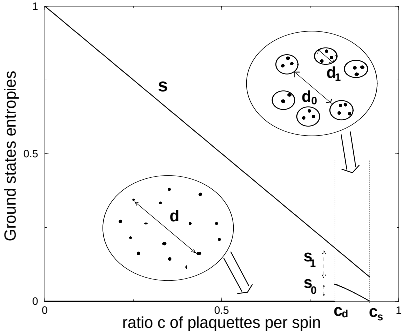
$$\begin{cases} \\ x_1 + x_2 + x_4 = 1 \end{cases}$$

eliminate the second clause

# Phase transitions

## Leaf removal

The leaf removal algorithm can be analyzed using differential equations.




Two possible outputs:

1. The algorithm removes all the clauses $\Rightarrow$ no clusters
2. A 2-core of variables remains $\Rightarrow$ clusters

The phase diagram is simpler than $k$-SAT: $\alpha_d = \alpha_f$, $\alpha_c = \alpha_s$

# Phase transitions

## The potential for the backbone

For a system defined by $\mathcal{C}(t) = \{c_j(t)\}_{j=2,\cdots,k}$, define

$G(b,t) = \sum_{j=2}^{k} c_j(t) b^j$

$V(b,t) = -\frac{G(b,t)}{1-t} + b + (1-b)\log(1-b)$

The phase space structure is manifested by the shape of the potential:

# Phase transitions

## The potential for the backbone

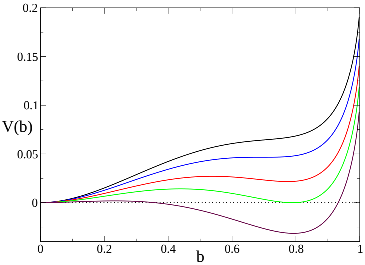
For a system defined by $\mathcal{C}(t) = \{c_j(t)\}_{j=2,\cdots,k}$, define

$G(b,t) = \sum_{j=2}^{k} c_j(t) b^j$

$V(b,t) = -\frac{G(b,t)}{1-t} + b + (1-b)\log(1-b)$

The phase space structure is manifested by the shape of the potential:

1. $V$ has a single minimum in $b = 0 \;\Rightarrow$ no clusters
2. $V$ has a secondary minimum in $b^* \neq 0 \Rightarrow$ clusters
3. $V(b^*) \leq 0 \quad \Rightarrow$ UNSAT



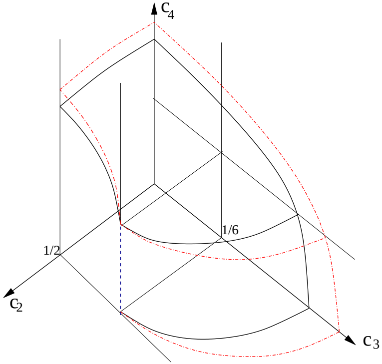

Note: $d_0 = 1/2, d_1 = (1-b^*)/2$

# Phase transitions

## The phase diagram

Phase diagram for $k = 4$

1. Clustering transition surface $\Sigma_d$ (black)
2. Sat/Unsat transition surface $\Sigma_s$ (red)
3. Contradiction surface $\Sigma_q$, $c_2 = 1/2$
4. Critical line (blue), $\Sigma_{crit} = \{c_2 = 1/2, c_3 = 1/6\}$

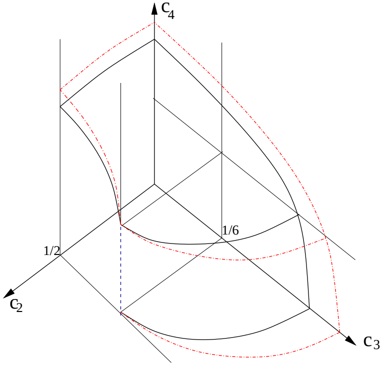



Trajectories start on the axis $c_4$ at $c_4 = \alpha$ and evolve according to the differential equations $\dot{c}_j = \frac{(j+1)c_{j+1} - jc_j}{1-t} - \rho_j(t)$

# Phase transitions
## The phase diagram

Phase diagram for $k = 4$

1. Clustering transition surface $\Sigma_d$ (black)
2. Sat/Unsat transition surface $\Sigma_s$ (red)
3. Contradiction surface $\Sigma_q$, $c_2 = 1/2$
4. Critical line (blue), $\Sigma_{crit} = \{c_2 = 1/2, c_3 = 1/6\}$

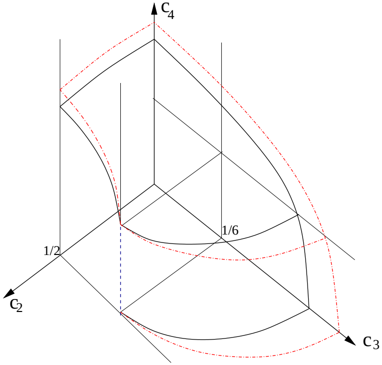



Trajectories start on the axis $c_4$ at $c_4 = \alpha$ and evolve according to the differential equations $\dot{c}_j = \frac{(j+1)c_{j+1} - jc_j}{1-t} - \rho_j(t)$

# Phase transitions
## The phase diagram

Phase diagram for $k = 4$

1. Clustering transition surface $\Sigma_d$ (black)
2. Sat/Unsat transition surface $\Sigma_s$ (red)
3. Contradiction surface $\Sigma_q$, $c_2 = 1/2$
4. Critical line (blue), $\Sigma_{crit} = \{c_2 = 1/2, c_3 = 1/6\}$



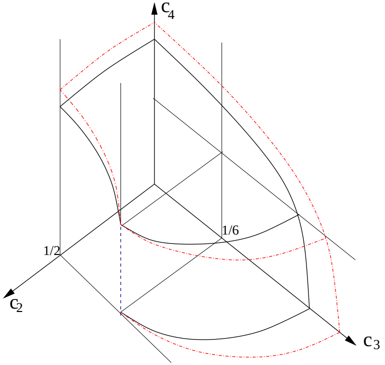

Trajectories start on the axis $c_4$ at $c_4 = \alpha$ and evolve according to the differential equations $\dot{c}_j = \frac{(j+1)c_{j+1} - jc_j}{1-t} - \rho_j(t)$

# Phase transitions

## The phase diagram

Phase diagram for $k = 4$

1. Clustering transition surface $\Sigma_d$ (black)
2. Sat/Unsat transition surface $\Sigma_s$ (red)
3. Contradiction surface $\Sigma_q$, $c_2 = 1/2$
4. Critical line (blue), $\Sigma_{crit} = \{c_2 = 1/2, c_3 = 1/6\}$



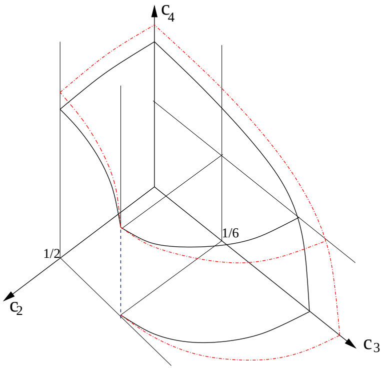

Trajectories start on the axis $c_4$ at $c_4 = \alpha$ and evolve according to the differential equations $\dot{c}_j = \frac{(j+1)c_{j+1} - jc_j}{1-t} - \rho_j(t)$

# Phase transitions

## The phase diagram

Phase diagram for $k = 4$

1. Clustering transition surface $\Sigma_d$ (black)
2. Sat/Unsat transition surface $\Sigma_s$ (red)
3. Contradiction surface $\Sigma_q$, $c_2 = 1/2$
4. Critical line (blue), $\Sigma_{crit} = \{c_2 = 1/2, c_3 = 1/6\}$




Trajectories start on the axis $c_4$ at $c_4 = \alpha$ and evolve according to the differential equations $\dot{c}_j = \frac{(j+1)c_{j+1} - jc_j}{1-t} - \rho_j(t)$

# Dynamic phase diagram

## Transition lines

For a given heuristic, define $t_d(\alpha)$, $t_s(\alpha)$, $t_q(\alpha)$ as the times where the trajectory starting at $c_k = \alpha$ crosses the phase boundary surfaces

# Dynamic phase diagram

## Transition lines

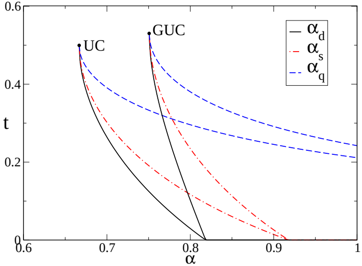
For a given heuristic, define $t_d(\alpha)$, $t_s(\alpha)$, $t_q(\alpha)$ as the times where the trajectory starting at $c_k = \alpha$ crosses the phase boundary surfaces

Equations for the lines follow from the equation of motion and the potential:

$$\frac{d\alpha_d}{dt} = - \left. \frac{1-F'(b,t)}{\partial_\alpha G'(b,t)} \right|_{\alpha_d,b_d}$$

$$\frac{d\alpha_s}{dt} = - \left. \frac{b-F(b,t)}{\partial_\alpha G(b,t)} \right|_{\alpha_s,b_s}$$

where $F(b,t) = \sum_{j=1}^{k} \rho_j(t) b^j$



# Dynamic phase diagram

## Transition lines

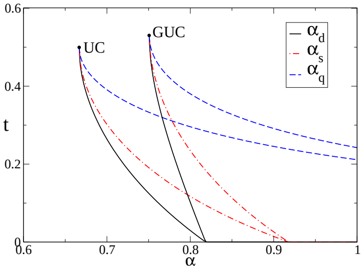
For a given heuristic, define $t_d(\alpha)$, $t_s(\alpha)$, $t_q(\alpha)$ as the times where the trajectory starting at $c_k = \alpha$ crosses the phase boundary surfaces

Equations for the lines follow from the equation of motion and the potential:

$$\frac{d\alpha_d}{dt} = - \left. \frac{1-F'(b,t)}{\partial_\alpha G'(b,t)} \right|_{\alpha_d, b_d}$$

$$\frac{d\alpha_s}{dt} = - \left. \frac{b-F(b,t)}{\partial_\alpha G(b,t)} \right|_{\alpha_s, b_s}$$

where $F(b,t) = \sum_{j=1}^{k} \rho_j(t) b^j$




We have $F(b,t)/b \leq F'(b,t) \leq 1$ as a consequence of $\sum_{j=1}^{k} j\rho_j \leq 1$

**Main result**

The trajectories cannot escape from the clustered phase $\Rightarrow \alpha_a \leq \alpha_d$

# Dynamic phase diagram
## Optimal algorithm

The best algorithm in this class can reach $\alpha_a = \alpha_d$ if $\frac{d\alpha_d}{dt} \equiv 0$

One can try to optimize the performances of the algorithm, e.g. by minimizing $\frac{d\alpha_d}{dt}$
*Global* optimization difficult for finite $k$

The problem is simple for $k \to \infty$: one can argue that for GUC $\dot{\alpha}_d \to 0$

**Optimality of GUC**

In the limit $k \to \infty$, $\alpha_a^{GUC} \sim \alpha_d \sim \frac{\log k}{k}$; therefore GUC is optimal in the class of Poissonian search algorithms

This result is supported by numerical simulations (direct solution of the differential equations and finite size scaling for $k \to \infty$) that show a correction $\sim 1/k$

# Dynamic phase diagram
## Optimal algorithm

The best algorithm in this class can reach $\alpha_a = \alpha_d$ if $\frac{d\alpha_d}{dt} \equiv 0$

One can try to optimize the performances of the algorithm, e.g. by minimizing $\frac{d\alpha_d}{dt}$
*Global* optimization difficult for finite $k$

The problem is simple for $k \to \infty$: one can argue that for GUC $\dot{\alpha}_d \to 0$

### Optimality of GUC

In the limit $k \to \infty$, $\alpha_a^{GUC} \sim \alpha_d \sim \frac{\log k}{k}$; therefore GUC is optimal in the class of Poissonian search algorithms

This result is supported by numerical simulations (direct solution of the differential equations and finite size scaling for $k \to \infty$) that show a correction $\sim 1/k$

# Conclusions

1. DPLL algorithms that preserve the uniform distribution cannot find solutions above the clustering transition
2. In the limit $k \to \infty$ GUC is optimal within this class and works up to $\alpha_d$
3. We could not prove that GUC is optimal for finite $k$. What is the optimal heuristic?

but...

1. For UE-CSP, clustering coincides with the appearance of a finite fraction of frozen variables in each cluster. These phenomena are in generally distinct. Which one is relevant?
2. The generalization of this work to $k$-SAT could solve this problem; however in that case an algorithm able to identify the clusters similarly to the Leaf Removal is not known

# Conclusions

1. DPLL algorithms that preserve the uniform distribution cannot find solutions above the clustering transition
2. In the limit $k \to \infty$ GUC is optimal within this class and works up to $\alpha_d$
3. We could not prove that GUC is optimal for finite $k$. What is the optimal heuristic?

but...

1. For UE-CSP, clustering coincides with the appearance of a finite fraction of frozen variables in each cluster. These phenomena are in generally distinct. Which one is relevant?
2. The generalization of this work to $k$-SAT could solve this problem; however in that case an algorithm able to identify the clusters similarly to the Leaf Removal is not known