Search strategies and phase transition in the Random Boolean satisfiability problem

Heitor Pascoal de Bittencourt

São Carlos Institute of Physics
University of São Paulo

March 2021

SAT

Search strategies and phase transition in the Random Boolean satisfiability problem

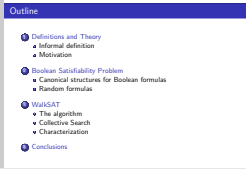
Heitor Pascoal de Bittencourt

São Carlos Institute of Physics
University of São Paulo

March 2021

1. Boa tarde a todos, meu nome é Heitor e vou apresentar meu trabalho de Mestrado
2. Estratégias de busca e transição de fase no problema da satisfatibilidade Booliana aleatório
3. Sob orientação do Prof. Fontanari
4. IFSC

Outline



1. Organizei esta apresentação da seguinte maneira:
2. Primeiro vou definir o problema
3. -> então vou dar alguns exemplos como motivação
4. Em seguida vamos ver duas formas canônicas de fórmulas bolianas
5. -> e o que são e características de fórmulas aleatórias
6. Após isso vamos ver uma estratégia poderosa para resolver este problema
7. -> vamos analisar a viabilidade de uma busca coletiva
8. -> caracterizar o algoritmo, e reanalisar o problema
9. Após tudo isso, vamos tirar algumas conclusões sobre este trabalho

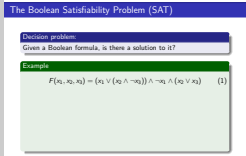
The Boolean Satisfiability Problem (SAT)

Decision problem:

Given a Boolean formula, is there a solution to it?

Example

$$F(x_1, x_2, x_3) = (x_1 \vee (x_2 \wedge \neg x_3)) \wedge \neg x_1 \wedge (x_2 \vee x_3) \qquad (1)$$



1. O que é o Prob da Satisfatibilidade Boliana?
2. Também chamado de SAT
3. É um problema de **decisão**:
4. Dada uma fórmula boliana, existe uma soluçao para ela?
5. Ou seja, é possível achar um conjunto de valores lógicos Verdadeiro/Falso de modo que satisfaça a fórmula?
6. Se existir pelo menos uma solução, então a resposta é "sim"

The Boolean Satisfiability Problem (SAT)

Decision problem:

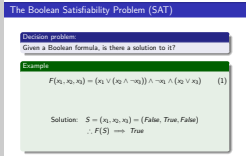
Given a Boolean formula, is there a solution to it?

Example

$$F(x_1, x_2, x_3) = (x_1 \vee (x_2 \wedge \neg x_3)) \wedge \neg x_1 \wedge (x_2 \vee x_3) \quad (1)$$

Solution: $S = (x_1, x_2, x_3) = (False, True, False)$

$\therefore F(S) \implies True$



1. Componentes de uma formula
2. -> literais são variáveis ou a sua negação
3. -> operadores bolianos $\wedge, \vee, \neg, (,)$

The Boolean Satisfiability Problem (SAT)

Decision problem:

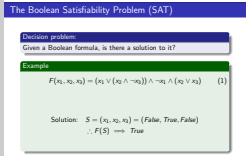
Given a Boolean formula, is there a solution to it?

Example

$$F(x_1, x_2, x_3) = (x_1 \vee (x_2 \wedge \neg x_3)) \wedge \neg x_1 \wedge (x_2 \vee x_3) \qquad (1)$$

Solution: $S = (x_1, x_2, x_3) = (False, True, False)$

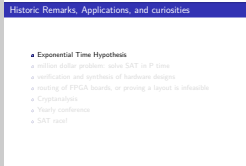
$\therefore F(S) \implies True$



1. Neste caso, há uma solução e dizemos que a fórmula é **satisfazível**
2. Poderia ter mais soluções

Historic Remarks, Applications, and curiosities

- Exponential Time Hypothesis
- million dollar problem: solve SAT in P time
- verification and synthesis of hardware designs
- routing of FPGA boards, or proving a layout is infeasible
- Cryptanalysis
- Yearly conference
- SAT race!



1. Agora que sabemos o que é o SAT, vamos ver algumas motivações
2. claro, além de toda importância teórica
3. **hipótese**, não foi provada
4. Impagliazzo and Paturi, in 1999
5. SAT não pode ser resolvido em tempo sub-exponencial no pior caso
6. então para uma fórmula com N variáveis, o tempo de execução de algo escala exponencialmente com N, no pior caso
7. sub-exponential time: $\mathcal{O}(2^{\delta n})$, for $\delta < 1$
8. ou seja, SAT pertence à classe NP

Historic Remarks, Applications, and curiosities

- Exponential Time Hypothesis
- million dollar problem: solve SAT in P time
- verification and synthesis of hardware designs
- routing of FPGA boards, or proving a layout is infeasible
- Cryptanalysis
- Yearly conference
- SAT race!

Historic Remarks, Applications, and curiosities
Exponential Time Hypothesis
million dollar problem: solve SAT in P time
verification and synthesis of hardware designs
routing of FPGA boards, or proving a layout is infeasible
Cryptanalysis
Yearly conference
SAT race!

1. Clay Mathematics Institute escolheu 7 problemas matemáticos como "Problemas do Milênio"
2. um desses problemas é: P vs NP
3. Resolver SAT em tempo P é equivalente a provar que $P = NP$
4. Quem fizer isso ganha 1 milhão de dólares

Historic Remarks, Applications, and curiosities

- Exponential Time Hypothesis
- million dollar problem: solve SAT in P time
- verification and synthesis of hardware designs
- routing of FPGA boards, or proving a layout is infeasible
- Cryptanalysis
- Yearly conference
- SAT race!

Historic Remarks, Applications, and curiosities
- Exponential Time Hypothesis
- million dollar problem: solve SAT in P time
- verification and synthesis of hardware designs
- routing of FPGA boards, or proving a layout is infeasible
- Cryptanalysis
- Yearly conference
- SAT race!

1. O SAT tbm tem usos/aplicações práticas
2. -> é usado para auxiliar o desenvolvimento de hardware, com roteamento e verificação de projets
3. -> é usado como ferramenta para criptoanálise
4. —> teste e análise de primitivas criptográficas
5. esses problemas podem ter desenas e até centenas de milhares de variáveis

Historic Remarks, Applications, and curiosities

- Exponential Time Hypothesis
- million dollar problem: solve SAT in P time
- verification and synthesis of hardware designs
- routing of FPGA boards, or proving a layout is infeasible
- Cryptanalysis
- Yearly conference
- SAT race!

Historic Remarks, Applications, and curiosities
- Exponential Time Hypothesis
- million dollar problem: solve SAT in P time
- verification and synthesis of hardware designs
- routing of FPGA boards, or proving a layout is infeasible
- Cryptanalysis
- Yearly conference
- SAT race!

1. SAT 2021: Barcelona, em Julho

Normal Form

Two main general structures for Boolean formulas:

Normal Form
Two main general structures for Boolean formulas:

1. Vamos ver agora as formas canônicas de fórmulas bolianas

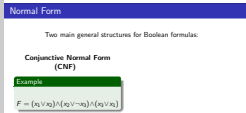
Normal Form

Two main general structures for Boolean formulas:

Conjunctive Normal Form (CNF)

Example

$F = (x_1 \vee x_2) \wedge (x_2 \vee \neg x_3) \wedge (x_3 \vee x_1)$



1. CNF: Conjunção (AND) de clausulas (ORs)
2. dentro dos parênteses só podemos ter OR

Normal Form

Two main general structures for Boolean formulas:

Conjunctive Normal Form (CNF)

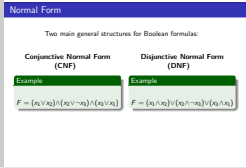
Example

$F = (x_1 \vee x_2) \wedge (x_2 \vee \neg x_3) \wedge (x_3 \vee x_1)$

Disjunctive Normal Form (DNF)

Example

$F = (x_1 \wedge x_2) \vee (x_2 \wedge \neg x_3) \vee (x_3 \wedge x_1)$



1. CNF: Conjunção (AND) de clausulas (ORs)
2. dentro dos parênteses só podemos ter OR
3. DNF: Disjunção (OR) de ANDs
4. são mais fáceis que CNF: basta satisfazer qualquer clausula
5. -> no CNF é necessário satisfazer TODAs as clausulas

Normal Form

Two main general structures for Boolean formulas:

Conjunctive Normal Form (CNF)

Example

$F = (x_1 \vee x_2) \wedge (x_2 \vee \neg x_3) \wedge (x_3 \vee x_1)$

Disjunctive Normal Form (DNF)

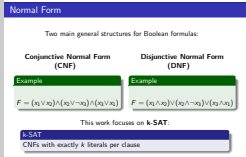
Example

$F = (x_1 \wedge x_2) \vee (x_2 \wedge \neg x_3) \vee (x_3 \wedge x_1)$

This work focuses on **k-SAT**:

k-SAT

CNFs with exactly k literals per clause



1. k-SAT é uma variação do SAT: CNFs com k literais por clausula
2. ou seja, em cada parênteses temos k variáveis
3. Qualquer fórmula Booliana pode ser transformada em CNF ou DNF
4. -> Transformada de Tseytin gera uma CNF, com um número linear de novas clausulas e variáveis
5. Transformar uma fórmula em DNF é possível, mas a nova fórmula é exponencialmente maior em N e M

How to generate?

This work focuses on **random 3**-SAT.

How to generate?
This work focuses on **random 3**-SAT.

1. não apenas k-SAT, mas k-SAT aleatório e com $k = 3$
2. 1-SAT e 2-SAT pertencem à classe de complexidade **P**: podem ser resolvidos em tempo polinomial
3. k-SAT com $k \geq 3$ pertence à classe **NP-C**
4. Por que 3-SAT?
5. -> estrutura mais simples que no caso geral
6. –> fácil de implementar código/algorithms
7. –> fácil de representar na memória

How to generate?

This work focuses on **random 3**-SAT.

3 parameters to generate a random formula:

- N: number of variables
- M: number of clauses
- $k = 3$: number of literals / clause

How to generate?
This work focuses on **random 3**-SAT.
3 parameters to generate a random formula:
- N: number of variables
- M: number of clauses
- $k = 3$: number of literals / clause

1. precisamos de 3 parâmetros para gerar uma fórmula
2. N
3. M
4. k
5. $k = 3$ nesse trabalho

How to generate?

This work focuses on **random 3**-SAT.

3 parameters to generate a random formula:

- N: number of variables
- M: number of clauses
- $k = 3$: number of literals / clause

A convenient definition

$$\alpha = M/N$$

How to generate?
This work focuses on **random 3**-SAT.
3 parameters to generate a random formula:
- N: number of variables
- M: number of clauses
- $k = 3$: number of literals / clause

A convenient definition
$\alpha = M/N$

1. É conveniente definir esse parâmetro α
2. número de clausulas / numero de vars
3. vamos bastante essa definição nesse trabalho

How to generate?

This work focuses on **random 3**-SAT.

3 parameters to generate a random formula:

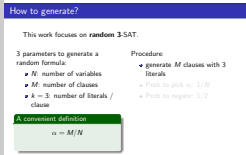
- N: number of variables
- M: number of clauses
- $k = 3$: number of literals / clause

Procedure:

- generate M clauses with 3 literals
- Prob to pick x_i: $1/N$
- Prob to negate: $1/2$

A convenient definition

$$\alpha = M/N$$



1. precisamos gerar M clausulas, então um laço para isso
2. E então gerar as $k = 3$ variáveis para cada clausula

How to generate?

This work focuses on **random 3**-SAT.

3 parameters to generate a random formula:

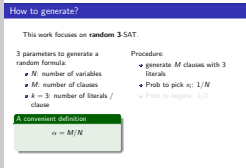
- N: number of variables
- M: number of clauses
- $k = 3$: number of literals / clause

Procedure:

- generate M clauses with 3 literals
- Prob to pick x_i: $1/N$
- Prob to negate: $1/2$

A convenient definition

$$\alpha = M/N$$



1. precisamos gerar M clausulas, então um laço para isso
2. E então gerar as $k = 3$ variáveis para cada clausula
3. escolhermos uma variável aleatoriamente

How to generate?

This work focuses on **random 3**-SAT.

3 parameters to generate a random formula:

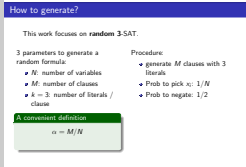
- N: number of variables
- M: number of clauses
- $k = 3$: number of literals / clause

Procedure:

- generate M clauses with 3 literals
- Prob to pick x_i: $1/N$
- Prob to negate: $1/2$

A convenient definition

$$\alpha = M/N$$



1. precisamos gerar M clausulas, então um laço para isso
2. E então gerar as $k = 3$ variáveis para cada clausula
3. escolhermos uma variável aleatoriamente
4. negamos esta variável também aleatoriamente

How to generate?

This work focuses on **random 3**-SAT.

3 parameters to generate a random formula:

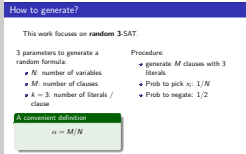
- N: number of variables
- M: number of clauses
- $k = 3$: number of literals / clause

Procedure:

- generate M clauses with 3 literals
- Prob to pick x_i: $1/N$
- Prob to negate: $1/2$

A convenient definition

$$\alpha = M/N$$



How to generate?

This work focuses on **random 3**-SAT.

3 parameters to generate a random formula:

- N: number of variables
- M: number of clauses
- $k = 3$: number of literals / clause

Procedure:

- generate M clauses with 3 literals
- Prob to pick x_i: $1/N$
- Prob to negate: $1/2$

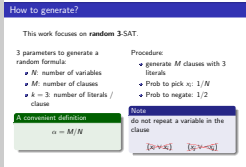
A convenient definition

$$\alpha = M/N$$

Note

do not repeat a variable in the clause

~~$(x_i \vee x_i)$~~ ~~$(x_j \vee \neg x_j)$~~



1. condição: nunca repetir um literal na clausula
2. se sortearmos um variável que já está lá, descartamos e escolhemos outra

Probability ρ that a random 3-CNF is satisfiable

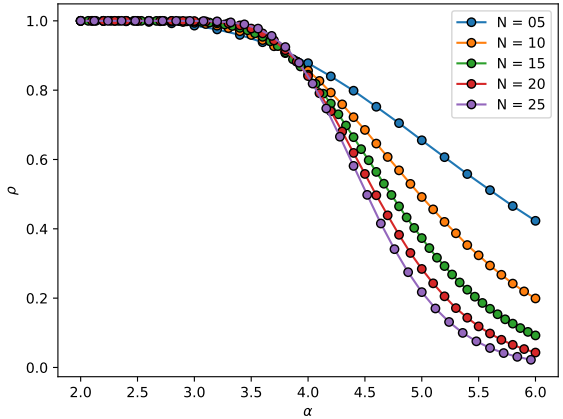
- Algorithm: for each (N, α) generate at least 10^5 random instances and check all possible solutions.

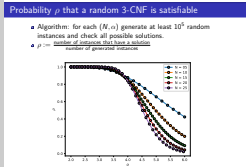
Probability ρ that a random 3-CNF is satisfiable
- Algorithm: for each (N, α) generate at least 10^5 random instances and check all possible solutions.

1. Nós começamos nossa análise estudando a probabilidade de um 3CNF aleatório ter solução
2. Usamos a Busca Exaustiva (também chamada de brute force)
3. mais de 100k fórmulas geradas

Probability ρ that a random 3-CNF is satisfiable

- Algorithm: for each (N, α) generate at least 10^5 random instances and check all possible solutions.
- $\rho := \frac{\text{number of instances that have a solution}}{\text{number of generated instances}}$





1. ρ é a probabilidade de um 3CNF aleatório ter solução
2. α pequeno: quase todas as fórmulas têm solução
3. α grandes: baixa prob de ter solução
4. existe um valor especial de α que muda esse comportamento

Number of solutions of a satisfiable random 3-CNF

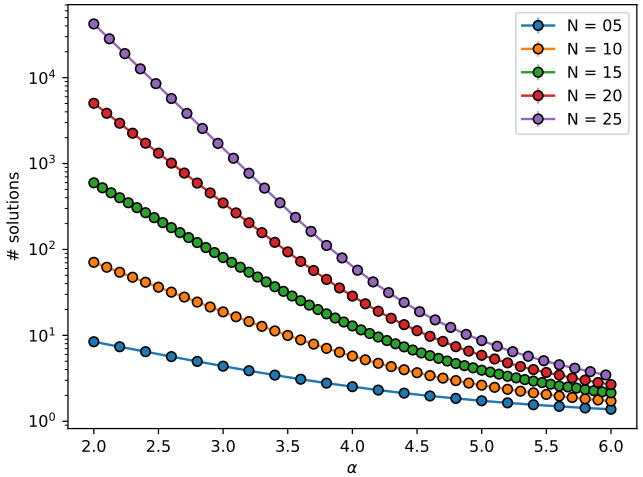
- Algorithm: for each (N, α) generate at least 10^5 instances and check all possible solutions.

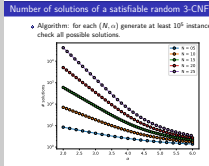
Number of solutions of a satisfiable random 3-CNF
- Algorithm: for each (N, α) generate at least 10^5 instances and check all possible solutions.

1. Agora que sabemos a probabilidade de ter solução, quantas soluções existem?
2. usamos o mesmo algoritmo anterior
3. -> **CAVEAT**: agora olhamos apenas para as instâncias que têm solução

Number of solutions of a satisfiable random 3-CNF

- Algorithm: for each (N, α) generate at least 10^5 instances and check all possible solutions.





1. as barras de erro são menores que os símbolos usados
2. α pequeno: número de soluções cresce exponencialmente com N
3. (again) the larger the α the harder the problem
4. o número de soluções tende a 1 conforme α cresce
5. Limitamos este trabalho a $N = 25$, porque a busca exaustiva é muito cara
6. um processador moderno consegue explorar todas as possíveis soluções com N até 30 em cerca de 1 segundo
7. -> extrapolating: $N = 40$: meia hora
8. -> extrapolating: $N = 50$: um mês
9. -> extrapolating: $N = 60$: 73 years
10. -> extrapolating: $N = 88$: 1.4 times the age of the Universe
11. Claramente precisamos de uma estratégia melhor

The WalkSAT

- local search strategy
- algorithm:

```
Assignment S ← random assignment;
for t ← 1 to T_max do
    if F(S) is True then
        return S;
    end
    C ← random unsatisfied clause from S;
    u ← UniformRandomNumber(0, 1);
    if u < P then
        x_i ← random variable in C;
    else
        x_i ← best variable to flip in C;
    end
    S[i] ← ¬x_i;
end
return No assignment found;
```

- P: noise parameter
- T_{max}: max number of iterations

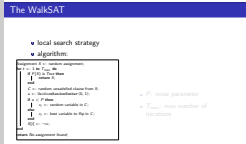
1. e a estratégia que escolhemos se chama WalkSAT
2. nome inspirado no caminhante aleatório
3. É um algo de busca local
4. -> simples
5. -> eficiente
6. -> robusto: consegue resolver problemas muito maiores

The WalkSAT

- local search strategy
- algorithm:

```
Assignment S ← random assignment;
for t ← 1 to T_max do
    if F(S) is True then
        return S;
    end
    C ← random unsatisfied clause from S;
    u ← UniformRandomNumber(0, 1);
    if u < P then
        x_i ← random variable in C;
    else
        x_i ← best variable to flip in C;
    end
    S[i] ← ¬x_i;
end
return No assignment found;
```

- P: noise parameter
- T_{max}: max number of iterations



1. este é o algoritmo
2. entrada: Formula F, P, T_{max}
3. começa gerando uma tentativa de solução S, aleatoriamente
4. A cada iteração: avaliamos se encontramos uma solução, se sim $F(S) == True$, retornamos S e o algoritmo encerra
5. caso contrário, escolhemos uma clausula nao satisfeita aleatoriamente
6. e com probabilidade P: escolhemos uma variável aleatoriamente nesta clausula
7. com prabalidade $1 - P$: escolhemos a melhor variável na clausula
8. e então invertemos (flipamos) esta variável
9. A parte crítica do WalkSAT é escolher essa "melhor" variável para inverter
10. e fazemos da seguinte maneira: escolhemos a variável que minimiza o número de clausulas que são satisfeitas com S mas se tornariam insatisfeitas se a variável fosse invertida

The WalkSAT

- local search strategy
- algorithm:

```
Assignment S ← random assignment;
for t ← 1 to T_max do
    if F(S) is True then
        return S;
    end
    C ← random unsatisfied clause from S;
    u ← UniformRandomNumber(0, 1);
    if u < P then
        x_i ← random variable in C;
    else
        x_i ← best variable to flip in C;
    end
    S[i] ← ¬x_i;
end
return No assignment found;
```

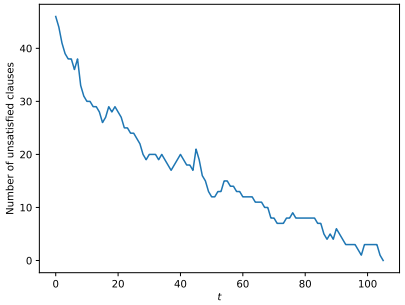
- P: noise parameter
- T_{max}: max number of iterations

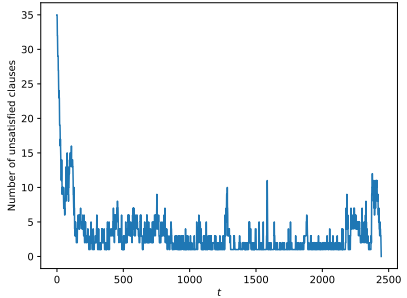
The WalkSAT
local search strategy
algorithm:
P: noise parameter
T_{max}: max number of iterations

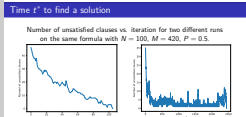
1. para ajudar o algo a sair de mínimos locais
2. para evitar loops infinitos

Time t^* to find a solution

Number of unsatisfied clauses vs. iteration for two different runs on the same formula with $N = 100$, $M = 420$, $P = 0.5$.



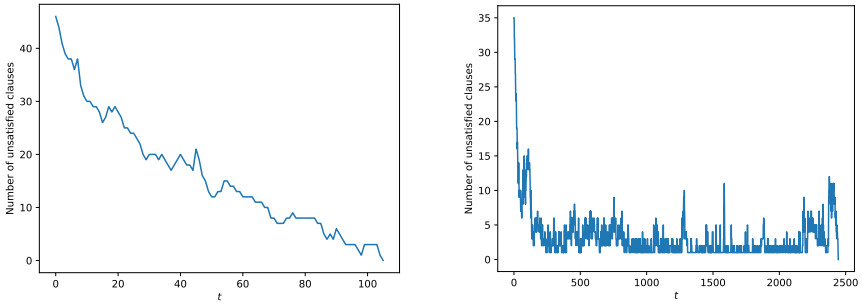




1. Vamos agora analizar o tempo que o WalkSAT leva para encontrar soluções
2. t^* é o tempo que ele leva para encontrar uma solução
3. os gráficos mostram o número de clausas não satisfeitas em função do tempo, para a mesma fórmula
4. WalkSAT pode ser muito rápido: em uma rodada precisou de cerca de 100 passos
5. mas o WalkSAT pode demorar, como na segunda rodada que precisou de cerca de 2500 passos
6. Note também a natureza estocástica deste algoritmo: algumas escolhas levam a um estado "pior"
7. Rapidamente o algo resolve a imensa maioria das 420 clausulas, e a maior parte do tempo é usado para resolver cerca de 10 clausulas
8. É interessante como inicialmente temos apenas 35 ou 45 clausulas para resolver. Isso é uma consequência de a atribuição inicial ser aleatória

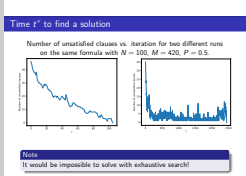
Time t^* to find a solution

Number of unsatisfied clauses vs. iteration for two different runs on the same formula with $N = 100$, $M = 420$, $P = 0.5$.



Note

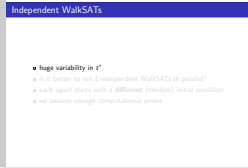
It would be impossible to solve with exhaustive search!



1. Vamos agora analizar o tempo que o WalkSAT leva para encontrar soluções
2. t^* é o tempo que ele leva para encontrar uma solução
3. os gráficos mostram o número de clausas não satisfeitas em função do tempo, para a mesma fórmula
4. WalkSAT pode ser muito rápido: em uma rodada precisou de cerca de 100 passos
5. mas o WalkSAT pode demorar, como na segunda rodada que precisou de cerca de 2500 passos
6. Note também a natureza estocástica deste algoritmo: algumas escolhas levam a um estado "pior"
7. Rapidamente o algo resolve a imensa maioria das 420 clausulas, e a maior parte do tempo é usado para resolver cerca de 10 clausulas
8. É interessante como inicialmente temos apenas 35 ou 45 clausulas para resolver. Isso é uma consequência de a atribuição inicial ser aleatória

Independent WalkSATs

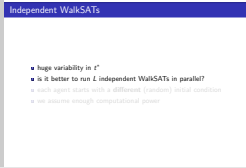
- huge variability in t^*
- is it better to run L independent WalkSATs in parallel?
- each agent starts with a **different** (random) initial condition
- we assume enough computational power



1. vemos que existe uma grande variabiliadde em t^*

Independent WalkSATs

- huge variability in t^*
- is it better to run L independent WalkSATs in parallel?
- each agent starts with a **different** (random) initial condition
- we assume enough computational power



1. Então, será que é vantajoso termos L walkSATs em paralelo?
2. e parar quando um deles encontrar uma solução?
3. essa ideia não foi explorada, anteriormente

Independent WalkSATs

- huge variability in t^*
- is it better to run L independent WalkSATs in parallel?
- each agent starts with a **different** (random) initial condition
- we assume enough computational power

Independent WalkSATs
- huge variability in t^*
- is it better to run L independent WalkSATs in parallel?
- each agent starts with a **different** (random) initial condition
- we assume enough computational power

1. então vamos ter cada um dos L agentes com uma condição inicial diferente
2. obviamente cada agente deve começar com uma condição inicial diferente
3. e ter seu próprio gerador de números aleatórios

Independent WalkSATs

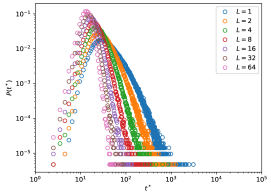
- huge variability in t^*
- is it better to run L independent WalkSATs in parallel?
- each agent starts with a **different** (random) initial condition
- we assume enough computational power

Independent WalkSATs
- huge variability in t^*
- is it better to run L independent WalkSATs in parallel?
- each agent starts with a **different** (random) initial condition
- we assume enough computational power

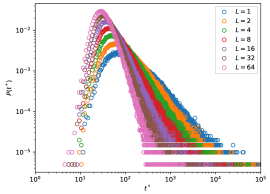
1. número suficiente de CPUs, nenhum custo adicional de paralelismo, e sincronia perfeita

Probability distribution of t^* for L independent WalkSATs

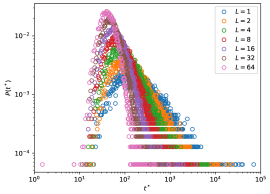
$N = 50,\ P = 0.5,\ T_{max} = 10^5$



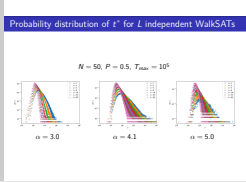
$\alpha = 3.0$



$\alpha = 4.1$



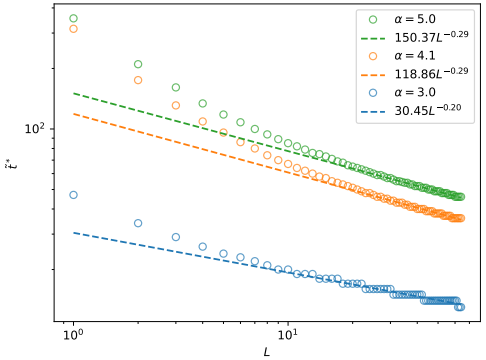
$\alpha = 5.0$

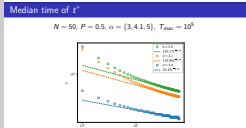


1. então vamos ver a distribuição de prob de t^* para vários WalkSATs em paralelo
2. para cada valor de L, os WalkSATs resolvem 10^5 fórmulas diferentes
3. mas não são quaisquer fórmulas, consideramos apenas aquelas que têm solução
4. A primeira coisa que podemos notar é que quanto mais WalkSATs, mais rápido resolvemos o problema
5. -> a largura das distribuições fica menor com Ls maiores
6. -> a cauda à direita é larga, e reduz com aumento de L
7. então aumentar o número de WalkSATs em paralelo aumenta a probabilidade de resolvermos o problema mais rapidamente
8. mas essa é uma análise qualitativa

Median time of t^*

$N = 50,\ P = 0.5,\ \alpha = \{3, 4.1, 5\},\ T_{max} = 10^5$

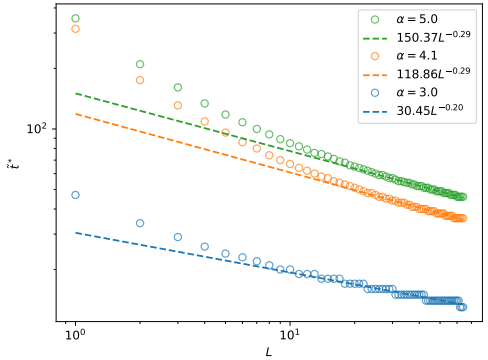




1. Para vermos se é vantajoso termos L WalkSATs em paralelo, precisamos comparar os tempos em função do número de agentes
2. -> e temos aqui um gráfico log log da mediana de t^* em função de L
3. —> o gráfico com a média está na dissertação
4. ——> a média não é muito confiável: como a distribuição de t^* apresenta uma cauda à direita muito larga, temos muito outliers e para evitar isso usamos a mediana
5. Novamente vemos que aumentar L reduz o tempo para encontrar uma solução

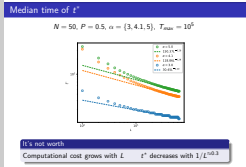
Median time of t^*

$N = 50,\ P = 0.5,\ \alpha = \{3, 4.1, 5\},\ T_{max} = 10^5$



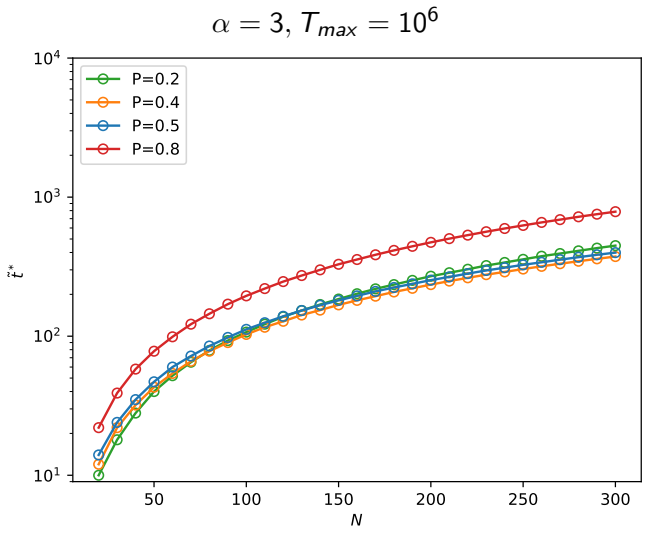
It's not worth

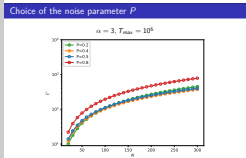
Computational cost grows with L $\quad$ t^* decreases with $1/L^{\approx 0.3}$



1. Mas não vale a pena: o custo computacional cresce linearmente com L
2. e a mediana decai com $1/L^{\approx 0.3}$
3. seria vantajoso se a mediana caisse mais rapidamente do que o custo aumenta

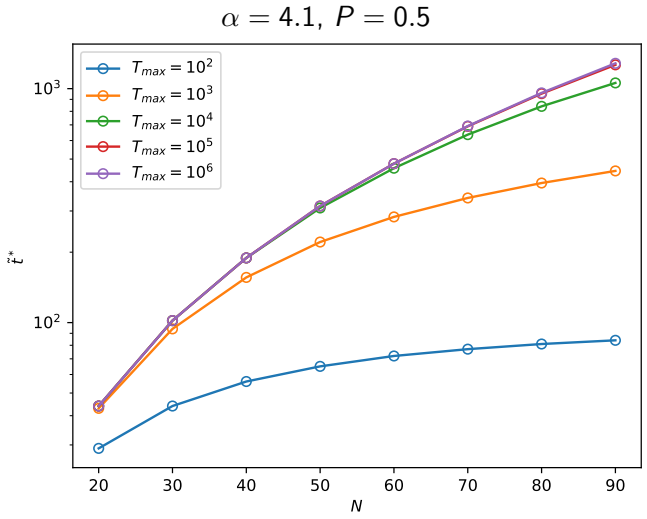
Choice of the noise parameter P

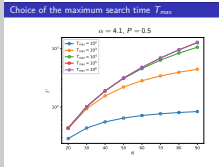




1. $P = .2$: 20% chance to get a random variable: many greedy/good choices
2. $P = .8$: 80% chance to get a random variable: few greedy/good choices
3. $P = 0.8$ is clearly the worst
4. $P = 0.2$ looks promising, but the number of outliers is huge: it finds solutions fast, but takes longer when it doesn't find it quickly
5. The median (shown here) cuts away those outliers
6. Plot with the average time is on the dissertation
7. $P = .4$ and $P = .5$ are very similar, as expected. The ideal P is around those values
8. we chose $P = .5$ for the remaining analysis
9. -> near the optimum value
10. -> good compromise between too greedy and non-greedy choices

Choice of the maximum search time T_{max}

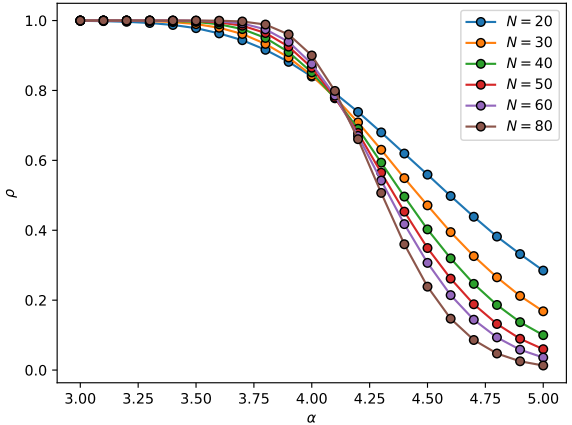




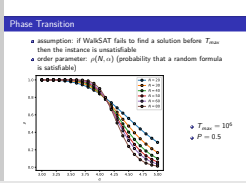
1. This plot shows the influence of T_{max} on the median search time
2. Very strong influence when $T_{max} = 10^2$
3. no significant difference between $T_{max} = 10^5$ and $T_{max} = 10^6$
4. $T_{max} = 10^4$ is very close as well
5. note that the maximum value of median time is considerably lower than T_{max}
6. T_{max} should be increased for higher values of N
7. -> Knuth suggests $T_{max} = 10000N$, for $N < 10^4$
8. -> Knuth suggests $T_{max} = 2500N$, for $N < 10^6$
9. we'll use 10^6 for safety

Phase Transition

- assumption: if WalkSAT fails to find a solution before T_{max} then the instance is unsatisfiable
- order parameter: $\rho(N, \alpha)$ (probability that a random formula is satisfiable)



- $T_{max} = 10^6$
- $P = 0.5$



1. we consider an instance to be unsatisfiable if WalkSAT cannot solve it in less than T_{max}
2. with this assumption, we can estimate the prob that a random formula is satisfiable
3. Similar to what we have done with the exhaustive search
4. We don't need to limit ourselves with $N < 25$!
5. generated 10^6 to 10^7 instances
6. -> $\rho := \frac{\text{number of instances that have a solution}}{\text{number of generated instances}}$
7. Clear threshold phenomenon at $\alpha_c \approx 4.1$, for larger N
8. In the limit $N \to \infty$: we expect $\rho \to 1$ for $\alpha < \alpha_c$
9. In the limit $N \to \infty$: we expect $\rho \to 0$ for $\alpha > \alpha_c$

How to characterize the phase transition

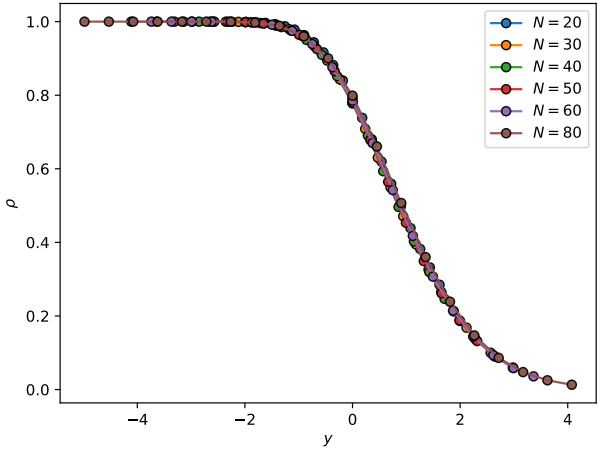
- critical point α_c
- critical exponent ν: sharpness of $\rho(N, \alpha)$ for finite N
- Finite-size scaling assumption for large N:
  - $\rho(N, \alpha) = f(y)$ with $y = N^{1/\nu}(\alpha - \alpha_c)/\alpha_c$
  - $\lim_{y\to\infty} f(y) \to 0$
  - $\lim_{y\to-\infty} f(y) \to 1$
  - ρ does not depend on N at $\alpha = \alpha_c$
  - sharpness of the transition $\partial\rho/\partial\alpha\,|_{\alpha=\alpha_c} = [f'(0)/\alpha_c]N^{1/\nu}$

How to characterize the phase transition
- critical point α_c
- critical exponent ν: sharpness of $\rho(N, \alpha)$ for finite N
- Finite-size scaling assumption for large N:
  - $\rho(N, \alpha) = f(y)$ with $y = N^{1/\nu}(\alpha - \alpha_c)/\alpha_c$
  - $\lim_{y\to\infty} f(y) \to 0$
  - $\lim_{y\to-\infty} f(y) \to 1$
  - ρ does not depend on N at $\alpha = \alpha_c$
  - sharpness of the transition $\partial\rho/\partial\alpha\,|_{\alpha=\alpha_c} = [f'(0)/\alpha_c]N^{1/\nu}$

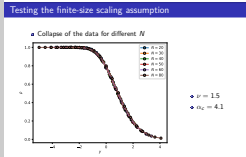
1. we now focus on characterization of the sharpness of the threshold
2. for that, we borrow the Finite Size Scaling technique from Statistical Physics to analyze it
3. Hipótese de escala finita
4. instead of ρ as function of α, we study ρ as order parameter
5. Quanto menor ν mais acentuada a transição

Testing the finite-size scaling assumption

- Collapse of the data for different N



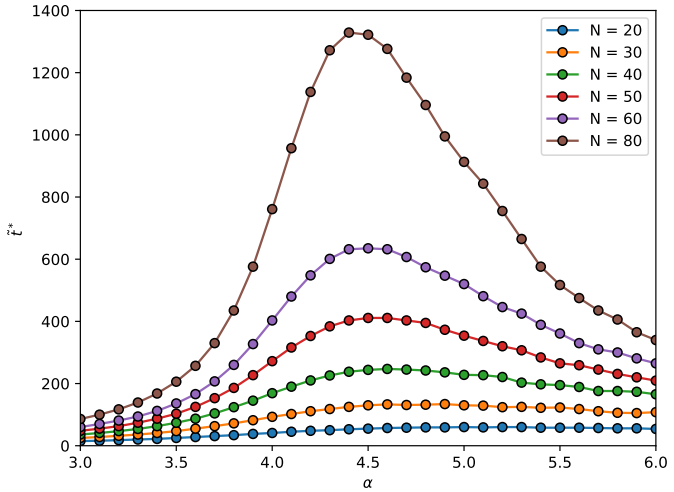
- $\nu = 1.5$
- $\alpha_c = 4.1$

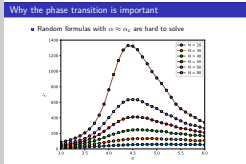


1. Todas as curvas, quando reescalas, coincidem
2. e temos uma curva universal para ρ
3. em 1994, Kirckpatrik & Selman publicaram uma Science com esse resultado. Nós reproduimos o resutlado deles usando um algoritmo completamente diferente, o WalkSAT. Eles usaram o Davis-Putnan
4. -> with $k = 2$: $\nu \approx 2.60$, $\alpha_c = 1$
5. -> with $k = 3$: $\nu \approx 1.50$, $\alpha_c \approx 4.2$
6. -> with $k = 4$: $\nu \approx 1.25$, $\alpha_c \approx 9.8$
7. -> with $k = 5$: $\nu \approx 1.10$, $\alpha_c \approx 20.9$
8. -> with $k = 6$: $\nu \approx 1.05$, $\alpha_c \approx 43.2$

Why the phase transition is important

- Random formulas with $\alpha \approx \alpha_c$ are hard to solve





1. walksat takes much longer to solve problems near the critical point than problems far from it, even for larger alphas
2. $T_{max} = 10^6$ and $P = 0.5$

Closing remarks

- 3-SAT is NP-Complete (ETH)
- Local search algorithm WalkSAT
- Random Boolean formulas
  - Phase transition at $\alpha_c \approx 4.1$
  - Random formulas near α_c are harder to solve
- Running L **independent** WalkSATs in parallel does make it faster, but not fast enough to compensate for the extra computational effort.

Closing remarks
3-SAT is NP-Complete (ETH)
Local search algorithm WalkSAT
Random Boolean formulas
Running L independent WalkSATs in parallel does make it faster, but not fast enough to compensate for the extra computational effort.

1. Vamos agora para nossas conclusões
2. SAT é um problema NP-C
3. então usar força bruta só é viável para $N < 30$
4. e precisamos de um algoritmo mais poderoso

Closing remarks

- 3-SAT is NP-Complete (ETH)
- Local search algorithm WalkSAT
- Random Boolean formulas
  - Phase transition at $\alpha_c \approx 4.1$
  - Random formulas near α_c are harder to solve
- Running L **independent** WalkSATs in parallel does make it faster, but not fast enough to compensate for the extra computational effort.

Closing remarks
- 3-SAT is NP-Complete (ETH)
- Local search algorithm WalkSAT
- Random Boolean formulas
  - Phase transition at $\alpha_c \approx 4.1$
  - Random formulas near α_c are harder to solve
- Running L **independent** WalkSATs in parallel does make it faster, but not fast enough to compensate for the extra computational effort.

Closing remarks

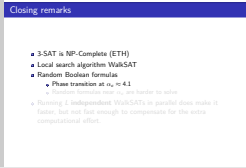
- 3-SAT is NP-Complete (ETH)
- Local search algorithm WalkSAT
- Random Boolean formulas
  - Phase transition at $\alpha_c \approx 4.1$
  - Random formulas near α_c are harder to solve
- Running L **independent** WalkSATs in parallel does make it faster, but not fast enough to compensate for the extra computational effort.

Closing remarks
- 3-SAT is NP-Complete (ETH)
- Local search algorithm WalkSAT
- Random Boolean formulas
  - Phase transition at $\alpha_c \approx 4.1$
  - Random formulas near α_c are harder to solve
- Running L **independent** WalkSATs in parallel does make it faster, but not fast enough to compensate for the extra computational effort.

1. estudamos e caracterizamos fórmulas aleatórias

Closing remarks

- 3-SAT is NP-Complete (ETH)
- Local search algorithm WalkSAT
- Random Boolean formulas
  - Phase transition at $\alpha_c \approx 4.1$
  - Random formulas near α_c are harder to solve
- Running L **independent** WalkSATs in parallel does make it faster, but not fast enough to compensate for the extra computational effort.



1. Note que a transição de fase é uma característica das fórmulas aleatórias, e não do algoritmo usado

Closing remarks

- 3-SAT is NP-Complete (ETH)
- Local search algorithm WalkSAT
- Random Boolean formulas
  - Phase transition at $\alpha_c \approx 4.1$
  - Random formulas near α_c are harder to solve
- Running L **independent** WalkSATs in parallel does make it faster, but not fast enough to compensate for the extra computational effort.

Closing remarks
- 3-SAT is NP-Complete (ETH)
- Local search algorithm WalkSAT
- Random Boolean formulas
  - Phase transition at $\alpha_c \approx 4.1$
  - Random formulas near α_c are harder to solve
- Running L independent WalkSATs in parallel does make it faster, but not fast enough to compensate for the extra computational effort.

1. e fórmulas com α próximo do valor crítico são mais difíceis de serem resolvidas

Closing remarks

- 3-SAT is NP-Complete (ETH)
- Local search algorithm WalkSAT
- Random Boolean formulas
  - Phase transition at $\alpha_c \approx 4.1$
  - Random formulas near α_c are harder to solve
- Running L **independent** WalkSATs in parallel does make it faster, but not fast enough to compensate for the extra computational effort.

Closing remarks
- 3-SAT is NP-Complete (ETH)
- Local search algorithm WalkSAT
- Random Boolean formulas
  - Phase transition at $\alpha_c \approx 4.1$
  - Random formulas near α_c are harder to solve
- Running L **independent** WalkSATs in parallel does make it faster, but not fast enough to compensate for the extra computational effort.

1. Vimos também que termos vários agentes em parelo, reduz o tempo de busca, mas não o suficiente para compensar o custo computacional extra

Acknowledgements

Thanks for the attention!

CAPES
Committee
Prof. Dr. José Fontanari & Dr. Sandro Reia
Family, girlfriend, friends, colleagues

Acknowledgements

Thanks for the attention!

CAPES
Committee
Prof. Dr. José Fontanari & Dr. Sandro Reia
Family, girlfriend, friends, colleagues

1. Agradeço a atenção e estou aberto a perguntas

Appendices

Appendices

Exponential Time Hypothesis

Random Initial State

Reductions

Exponential Time Hypothesis

Hypothesis

Unproven!

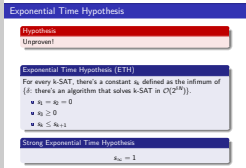
Exponential Time Hypothesis (ETH)

For every k-SAT, there's a constant s_k defined as the infimum of $\{\delta$: there's an algorithm that solves k-SAT in $\mathcal{O}(2^{\delta N})\}$.

- $s_1 = s_2 = 0$
- $s_3 \geq 0$
- $s_k \leq s_{k+1}$

Strong Exponential Time Hypothesis

$$s_\infty = 1$$



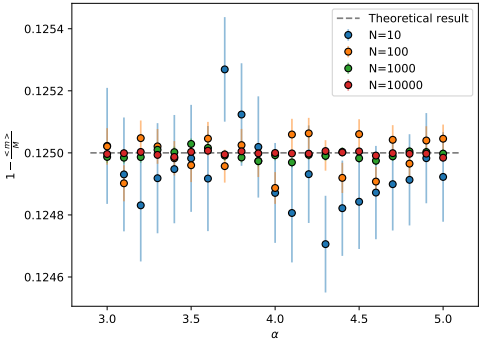
1. my thesis: sec 2.3.1
2. so far, no evidence against it
3. proving it is true or false solves the P vs NP problem
4. ETH published in 1999 by Impagliazzo and Paturi
5. infimum: largest lower bound

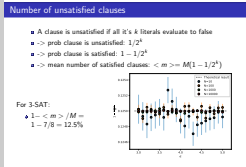
Number of unsatisfied clauses

- A clause is unsatisfied if all it's k literals evaluate to false
- -> prob clause is unsatisfied: $1/2^k$
- -> prob clause is satisfied: $1 - 1/2^k$
- -> mean number of satisfied clauses: $< m >= M(1 - 1/2^k)$

For 3-SAT:

- $1 - < m > /M = 1 - 7/8 = 12.5\%$





1. $N = 10, \alpha = 3$: $12.5\%30 = 3.75$
2. $N = 10, \alpha = 4.3$: $12.5\%43 = 5.375$
3. fração de clausulas não satisfeitas

What are reductions?

A polynomial-time **reduction** from problem X to problem Y:

- it is an algorithm R
- notation: $X \leq Y$
- meaning: any input for problem X can be transformed in an input for problem Y
- constraint: solving Y on the modified input yields the same answer as X on x
- mathematically: $X(x) = Y(R(x))$

What are reductions?
A polynomial-time **reduction** from problem X to problem Y:
- it is an algorithm R
- notation: $X \leq Y$
- meaning: any input for problem X can be transformed in an input for problem Y
- constraint: solving Y on the modified input yields the same answer as X on x
- mathematically: $X(x) = Y(R(x))$

1. example: reduce SAT to Traveling Salesman Problem
2. -> $SAT(x) = TSP(R(x))$
3. -> come up with an algo R that transform input for SAT in input for TSP: transform a list of Booleans in a list of cities and the distance between them
4. -> R runs in polynomial time: proves TSP is NP-C
5. otherwise, if you can solve TSP in poly-time, you could use it as subroutine to solve SAT in poly-time

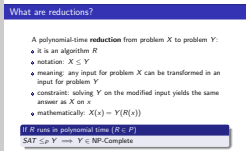
What are reductions?

A polynomial-time **reduction** from problem X to problem Y:

- it is an algorithm R
- notation: $X \leq Y$
- meaning: any input for problem X can be transformed in an input for problem Y
- constraint: solving Y on the modified input yields the same answer as X on x
- mathematically: $X(x) = Y(R(x))$

If R runs in polynomial time ($R \in P$)

$SAT \leq_P Y \implies Y \in$ NP-Complete



1. example: reduce SAT to Traveling Salesman Problem
2. -> $SAT(x) = TSP(R(x))$
3. -> come up with an algo R that transform input for SAT in input for TSP: transform a list of Booleans in a list of cities and the distance between them
4. -> R runs in polynomial time: proves TSP is NP-C
5. otherwise, if you can solve TSP in poly-time, you could use it as subroutine to solve SAT in poly-time