3SAT

Approaches and Implementation
A Lightning Talk

Dylan D. Hunn
@dylhunn

Why 3SAT?

- “Canonical” NP-complete problem

Why 3SAT?

- “Canonical” NP-complete problem
- Fast heuristic-based solvers exist
  - Many other NPC problems are reduced to 3SAT in practice

Why 3SAT?

- “Canonical” NP-complete problem
- Fast heuristic-based solvers exist
  - Many other NPC problems are reduced to 3SAT in practice
- Interesting topic at the intersection of:
  - Boolean logic
  - Complexity theory
  - Algorithms

Why this talk?

- Survey the 3SAT problem

Why this talk?

- Survey the 3SAT problem
- See one interesting mapping reduction and solving algorithm

Why this talk?

- Survey the 3SAT problem
- See one interesting mapping reduction and solving algorithm
- Consider implementation details and try my custom solver

Why this talk?

- Survey the 3SAT problem
- See one interesting mapping reduction and solving algorithm
- Consider implementation details and try my custom solver
- Further implications

Why this talk?

- Survey the 3SAT problem
- See one interesting mapping reduction and solving algorithm
- Consider implementation details and try my custom solver
- Further implications

Lightning talk! All this in under 5 minutes! Let’s get started.

Follow along!

Project is open source on Github @dylhunn:

```
git clone https://github.com/dylhunn/257-max-3sat.git
cd 257-max-3sat/src
make test
```

Solve an arbitrary 3SAT problem:

```
make
./threesat ../test_data/test0.txt 1
```

Boolean satisfiability problem

Boolean formulae composed of clauses in conjunctive normal form

$(A \lor \neg A \lor A) \land (\neg B \lor \neg A \lor C)$

Is this formula satisfiable?

Boolean satisfiability problem

Boolean formulae composed of clauses in conjunctive normal form

$(A \lor \neg A \lor A) \land (\neg B \lor \neg A \lor C)$

Is this formula satisfiable?

Yes. One of several possible assignments:

`A: True`

`B: False`

`C: True`

Some terms

3CNF: A conjunctive normal form formula with 3 variables per clause

Some terms

3CNF: A conjunctive normal form formula with 3 variables per clause

3SAT

Decision problem: given such a formula, is it satisfiable?

Some terms

3CNF: A conjunctive normal form formula with 3 variables per clause

3SAT

Decision problem: given such a formula, is it satisfiable?

Assignment problem: find the satisfying assignment

Some terms

3CNF: A conjunctive normal form formula with 3 variables per clause

3SAT

Decision problem: given such a formula, is it satisfiable?

Assignment problem: find the satisfying assignment

MAX-3SAT: what assignment satisfies the largest number of clauses?

Naïve algorithm

- Enumerate the variables
- Recursively generate and check every assignment

Naïve algorithm

- Enumerate the variables
- Recursively generate and check every assignment
- Algorithm:

```
bool naïve_solve(formula f, solution s, int index) {
    if (index >= f.num_vars) {
        if (solution_is_valid(f, s)) return true;
        else return false;
    }
    for (int i = 0; i < 2; i++) {
        s[index] = i; // variable # index
        if (naïve_solve(f, s, index + 1)) return true;
    }
    return false;
}
```

Naïve algorithm

- Enumerate the variables
- Recursively generate and check every assignment
- Try it:

```
make
./threesat ../test_data/test0.txt 0
```

- It works:

```
[dylhunn@dylhunn0 src]$ make
[dylhunn@dylhunn0 src]$ ./threesat ../test_data/test0.txt 0
Solvable. Solution: [var1=false, var2=false, var3=false, var4=false, var5=true]
[dylhunn@dylhunn0 src]$ 
```

Naïve algorithm

- Enumerate the variables
- Recursively generate and check every assignment
- Try it:

```
make
./threesat ../test_data/test2.txt 0
```

- But...

Naïve algorithm

- Enumerate the variables
- Recursively generate and check every assignment
- Try it:

```
make
./threesat ../test_data/test2.txt 0
```

- It’s way too slow on large inputs (*obviously*):

```
[dylhunn@dylhunn0 src]$ ./threesat ../test_data/test2.txt 0
Solvable. Solution: [var1=false, var2=false, var3=true, var4=false, var5=false, var6=false, var7=false, v
ar8=false, var9=false, var10=false, var11=true, var12=false, var13=false, var14=true, var15=false, var16=
false, var17=false, var18=true, var19=false, var20=false, var21=false, var22=false, var23=false, var24=fa
lse, var25=true, var26=false, var27=false, var28=true, var29=false]
[dylhunn@dylhunn0 src]$
```

Runtime: ~10 s for 29 variables in 16 clauses

Bron-Kerbosch algorithm

Consider this formula:

$$(X \lor X \lor Y) \land (\neg X \lor \neg Y \lor \neg Y) \land (\neg X \lor Y \lor Y)$$

Bron-Kerbosch algorithm

Consider this formula:

$$(X \lor X \lor Y) \land (\neg X \lor \neg Y \lor \neg Y) \land (\neg X \lor Y \lor Y)$$

- Make each term a node

Bron-Kerbosch algorithm

Consider this formula:

$(X \lor X \lor Y) \land (\neg X \lor \neg Y \lor \neg Y) \land (\neg X \lor Y \lor Y)$

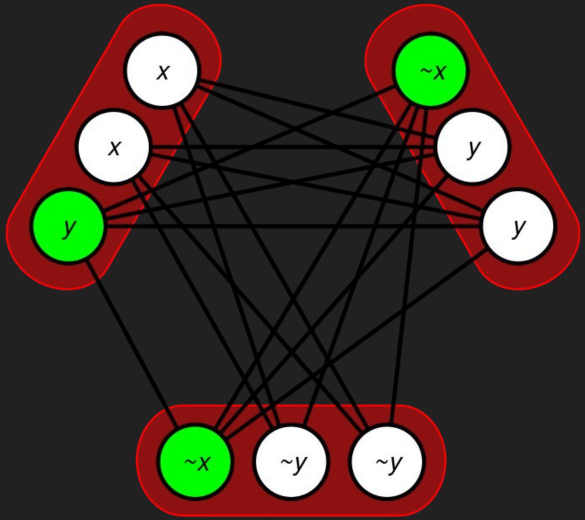
- Make each term a node
- Connect all nodes except:
  - nodes that share a clause
  - Incompatible nodes (e.g. $A \land \neg A$)

Bron-Kerbosch algorithm

Consider this formula:

(X ∨ X ∨ Y) ∧ (¬X ∨ ¬Y ∨ ¬Y) ∧ (¬X ∨ Y ∨ Y)

- Make each term a node
- Connect all nodes except:
  - nodes that share a clause
  - Incompatible nodes (e.g. A ∧ ¬A)



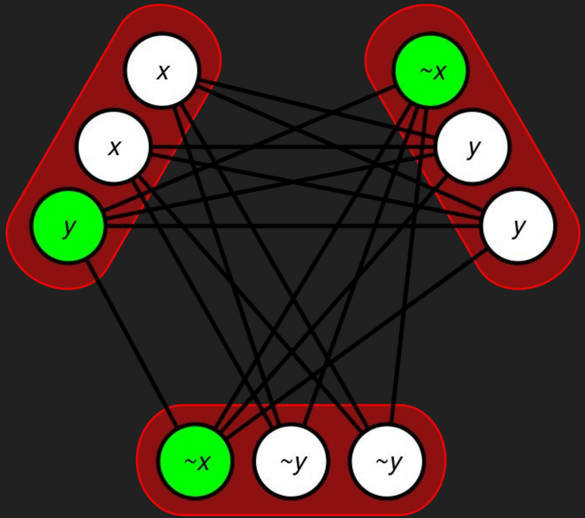
(Example illustration: Thore Husfeldt)

Bron-Kerbosch algorithm

Consider this formula:

(X ⋁ X ⋁ Y) ⋀ (¬X ⋁ ¬Y ⋁ ¬Y) ⋀ (¬X ⋁ Y ⋁ Y)

- Make each term a node
- Connect all nodes except:
  - nodes that share a clause
  - Incompatible nodes (e.g. A ⋀ ¬A)
- Any 3-clique is a satisfying assignment
  - (since there are 3 clauses)



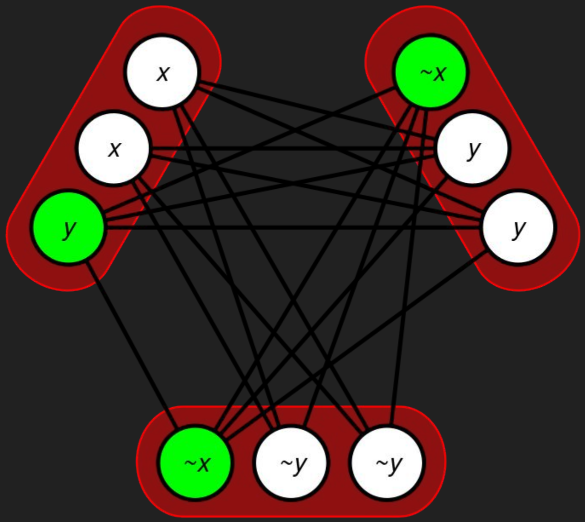
(Example illustration: Thore Husfeldt)

Bron-Kerbosch algorithm

Consider this formula:

(X ∨ X ∨ Y) ∧ (¬X ∨ ¬Y ∨ ¬Y) ∧ (¬X ∨ Y ∨ Y)

- Make each term a node
- Connect all nodes except:
  - nodes that share a clause
  - Incompatible nodes (e.g. `A ∧ ¬A`)
- Any 3-clique is a satisfying assignment
  - (since there are 3 clauses)
- Such an assignment is also a max-clique!



(Example illustration: Thore Husfeldt)

Bron-Kerbosch algorithm

- Max-clique finding and 3SAT are both NP-complete problems
  - Thus, there ***must*** be a polynomial-time mapping reduction

Bron-Kerbosch algorithm

- Max-clique finding and 3SAT are both NP-complete problems
  - Thus, there ***must*** be a polynomial-time mapping reduction
- There is: Bron-Kerbosch algorithm!

Bron-Kerbosch algorithm

- Max-clique finding and 3SAT are both NP-complete problems
  - Thus, there ***must*** be a polynomial-time mapping reduction
- There is: Bron-Kerbosch algorithm!

```
bron_kerbosch(set r, set p, set x) {
     if (p.empty() && x.empty())
          maxcliques_add(r);
     for (vertex v : p) {
          bron_kerbosch(r + v, insersect(p,
     neighbors(v)),
               intersect(x, neighbors(v));
          x = p - v;
          x = x + v;
     }
}
```

Some observations

- n-variable formula requires n assignments for solution, [1, n]

Some observations

- n-variable formula requires n assignments for solution, [1, n]
- Let’s use these variable numbers [1, n] as indices

Some observations

- n-variable formula requires n assignments for solution, [1, n]
- Let’s use these variable numbers [1, n] as indices
- The graph can be represented efficiently as an adjacency matrix

Some observations

- n-variable formula requires n assignments for solution, [1, n]
- Let’s use these variable numbers [1, n] as indices
- The graph can be represented efficiently as an adjacency matrix
- Bron-Kerbosch requires set operations

Some observations

- n-variable formula requires n assignments for solution, [1, n]
- Let’s use these variable numbers [1, n] as indices
- The graph can be represented efficiently as an adjacency matrix
- Bron-Kerbosch requires set operations
- But the set keys are uniformly distributed integers!

A Hashset!

- n-variable formula requires n assignments for solution, [1, n]
- Let’s use these variable numbers [1, n] as indices
- The graph can be represented efficiently as an adjacency matrix
- Bron-Kerbosch requires set operations
- But the set keys are **uniformly distributed integers**!
- Performance boost: implement our own hashset
    - No generic types
    - **Identity hash function**, since the keys are uniformly distributed!

Performant results

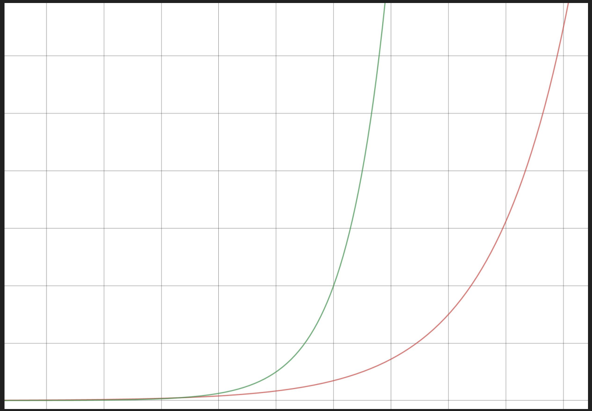
- Worst case: no n-vertex graph has more than $3^{(n/3)}$ max-cliques
  - Even better in practice: our nodes are connected to at most $(n-4)$ other nodes!

Performant results

- Worst case: no n-vertex graph has more than $3^{(n/3)}$ max-cliques
  - Even better in practice: our nodes are connected to at most $(n-4)$ other nodes!
- This is an improvement over the naive average case of 2^n

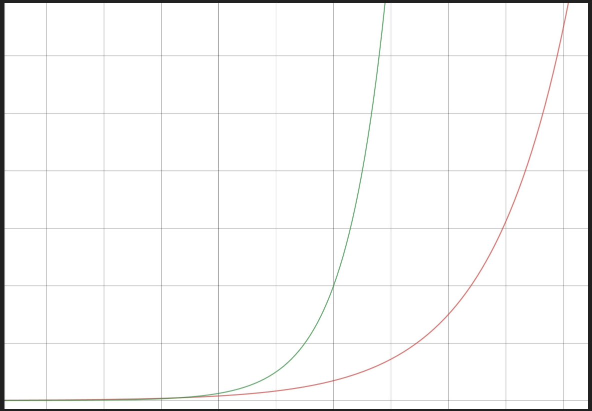
Performant results

- Worst case: no n-vertex graph has more than $3^{(n/3)}$ max-cliques
  - Even better in practice: our nodes are connected to at most $(n-4)$ other nodes!
- This is an improvement over the naive average case of 2^n
- Indeed:



Performant results

- Worst case: no n-vertex graph has more than $3^{(n/3)}$ max-cliques
  - Even better in practice: our nodes are connected to at most $(n-4)$ other nodes!
- This is an improvement over the naive average case of 2^n
- Indeed:



Barring a polynomial solution with P=NP, this much better.

Performant results

- Let’s try the large case again:

```
make
./threesat ../test_data/test2.txt 1
```

- Much better performance!

```
[dylhunn@dylhunn0 src]$ ./threesat ../test_data/test2.txt 1
Solvable. Solution: [var1=true, var2=false, var3=false, var4=true, var5=false, var6=false, var7=true, var
8=false, var9=false, var10=true, var11=false, var12=false, var13=true, var14=false, var15=false, var16=tr
ue, var17=false, var18=false, var19=false, var20=true, var21=false, var22=true, var23=false, var24=false,
 var25=true, var26=false, var27=false, var28=true, var29=false]
[dylhunn@dylhunn0 src]$
```

Runtime: < 0.1 s for 29 variables in 16 clauses

Generating test data

- 3CNF formulae to empirically verify this can be randomly generated

Generating test data

- 3CNF formulae to empirically verify this can be randomly generated
- Observation: the ratio of variables to terms affects satisfiability

Generating test data

- 3CNF formulae to empirically verify this can be randomly generated
- Observation: the ratio of variables to terms affects satisfiability
  - Low ratio: $(\neg A \lor \neg A \lor B) \land (\neg B \lor \neg A \lor \neg A) \land (A \lor A \lor A)$
    - **Unsatisfiable** *(and easy to construct)*

Generating test data

- 3CNF formulae to empirically verify this can be randomly generated
- Observation: the ratio of variables to terms affects satisfiability
  - Low ratio: $(\neg A \lor \neg A \lor B) \land (\neg B \lor \neg A \lor \neg A) \land (A \lor A \lor A)$
    - **Unsatisfiable** *(and easy to construct)*
  - High ratio: $(A \lor B \lor A) \land (\neg C \lor \neg B \lor \neg D) \land (\neg A \lor E \lor F)$
    - **Satisfiable** *(and easy to construct)*

Generating test data

- 3CNF formulae to empirically verify this can be randomly generated
- Observation: the ratio of variables to terms affects satisfiability
  - Low ratio: **$(\neg A \lor \neg A \lor B) \land (\neg B \lor \neg A \lor \neg A) \land (A \lor A \lor A)$**
    - **Unsatisfiable** *(and easy to construct)*
  - High ratio: **$(A \lor B \lor A) \land (\neg C \lor \neg B \lor \neg D) \land (\neg A \lor E \lor F)$**
    - **Satisfiable** *(and easy to construct)*
- In the **3SAT decision problem**, this can be used as a heuristic!

Generating test data

- 3CNF formulae to empirically verify this can be randomly generated
- Observation: the ratio of variables to terms affects satisfiability
  - Low ratio: $(\neg A \lor \neg A \lor B) \land (\neg B \lor \neg A \lor \neg A) \land (A \lor A \lor A)$
    - Unsatisfiable *(and easy to construct)*
  - High ratio: $(A \lor B \lor A) \land (\neg C \lor \neg B \lor \neg D) \land (\neg A \lor E \lor F)$
    - Satisfiable *(and easy to construct)*
- In the **3SAT decision problem**, this can be used as a heuristic!
- In high ratio cases, the naive solver often performs quite well
  - Might terminate after only exploring one assignment!
  - No graph construction overhead

Generating test data

- 3CNF formulae to empirically verify this can be randomly generated
- Observation: the ratio of variables to terms affects satisfiability
  - Low ratio: **$(\neg A \lor \neg A \lor B) \land (\neg B \lor \neg A \lor \neg A) \land (A \lor A \lor A)$**
    - **Unsatisfiable** *(and easy to construct)*
  - High ratio: **$(A \lor B \lor A) \land (\neg C \lor \neg B \lor \neg D) \land (\neg A \lor E \lor F)$**
    - **Satisfiable** *(and easy to construct)*
- In the **3SAT decision problem**, this can be used as a heuristic!
- In high ratio cases, the naive solver often performs quite well
  - Might terminate after only exploring one assignment!
  - No graph construction overhead
- Thus, empirical verification of the Big-O graph is challenging
  - Must find the 50/50 Satisfiable / Unsatisfiable ratio

Solving MAX-3SAT

- Bron-Kerbosch finds *all* max-cliques

Solving MAX-3SAT

- Bron-Kerbosch finds *all* max-cliques
- If the largest max clique < `n/3` nodes, no satisfying assignment

Solving MAX-3SAT

- Bron-Kerbosch finds *all* max-cliques
- If the largest max clique < `n/3` nodes, no satisfying assignment
- But it is still the MAX-3SAT solution!

Solving MAX-3SAT

- Bron-Kerbosch finds *all* max-cliques
- If the largest max clique < `n/3` nodes, no satisfying assignment
- But it is still the MAX-3SAT solution!
- This is an obvious next step for my implementation

3SAT

Approaches and Implementation
A Lightning Talk

Questions?