Investigating 3SAT

(Guide presentation for 380CT Coursework 2)

Kamal Bentahar

19th December 2016

Notation

Let $x_1, x_2, \ldots, x_n$ be Boolean **variables**, and let ϕ be a Boolean formula written in 3-cnf (Conjunctive Normal Form)

$$\phi = c_1 \wedge c_2 \wedge \cdots \wedge c_\ell,$$

where each **clause** $c_m = x_i \vee x_j \vee x_k$, for some $i, j, k = 1, 2, \ldots, n$ and $m = 1, \ldots, \ell$.

A **literal** can be x_i or $\neg x_i$ for some $i = 1, 2, \ldots, n$.

The ratio ℓ/n is important for experiments, and will be denoted by ρ.

Definition of the problem

Decisional 3SAT

Decide if ϕ is satisfiable.

NP-complete.

Computational/Search 3SAT

If ϕ is satisfiable then find a satisfying assignment.

Optimization 3SAT (Max 3SAT)

Find an assignment that minimizes the number of non-satisfying clauses.

NP-hard.

Testing methodology

1. **Exhaustive search:** average time for instances with increasing n.
2. **Dynamic programming:** average time for instances with increasing ℓ.
3. **Greedy and meta-heuristics:** quality of approximation with increasing ρ .
   Quality of approximation is calculated as the ratio of satisfied clauses to ℓ.

Random instances sampling strategy

General 3SAT instances will be generated by selecting exactly 3 different literals from

$$\{x_1, \neg x_1, x_2, \neg x_2, \ldots, x_n \neg x_n\}$$

uniformly at random. Do not allow clauses including both x_i and $\neg x_i$ (tautological clauses). [1].

For 'yes' instances, a random variable assignment is fixed first, then clauses are randomly constructed making sure each is satisfiable.

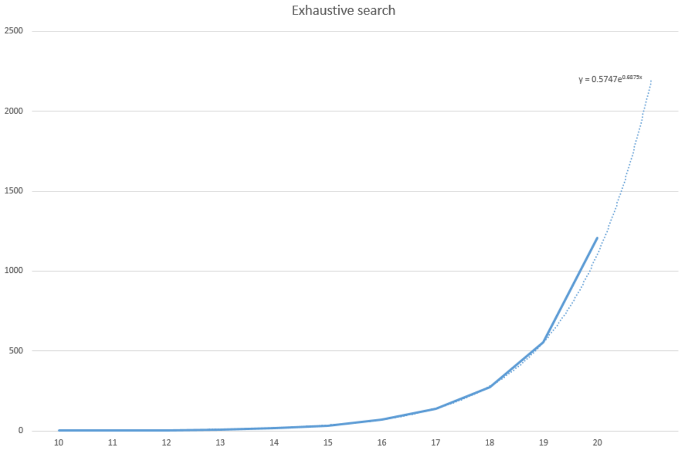
Exhaustive search – theory

for all possible variable assignments of $x_1, x_2, \ldots, x_n$ **do**
  if $\phi(x_1, x_2, \ldots, x_n)$ evaluates to True **then**
    return True
return False

There are 2^n possible assignments, and each evaluation of ϕ costs $O(\ell)$. So this algorithm costs

$$O(\ell\, 2^n).$$

Exhaustive search – empirical results



Average time in $100\times$ seconds [TODO: REDO EXPERIMENT] for randomly generated instances with $n = \ell$ for $n = 10, \ldots$.
Dotted line: fitted exponential curve.

Dynamic Programming

$\mathcal{A} \leftarrow \emptyset$ ▷ Set of possible assignments
for $k = 1, 2, \ldots, \ell$ **do**
    $S \leftarrow$ all the satisfying assignments of c_k ▷ 7 at most
    $update \leftarrow \emptyset$
    for $p \in \mathcal{A}$ **do**
        for $\sigma \in S$ **do**
            if σ and p do not clash **then**
                join p and σ and append to $update$
    $\mathcal{A} \leftarrow update$
return best candidate in $\mathcal{A}$

Cost: $O(\ell \times \max|\mathcal{A}|)$ time and $O(\max|\mathcal{A}|)$ space, but $|\mathcal{A}|$ can grow like 7^k in the worst case, we deduce that this algorithm can cost

$$O(\ell 7^\ell) \text{ time, and } O(7^\ell) \text{ space}$$

Only useful if ℓ is small. [TODO: VERIFY. CAN COST BE REDUCED?]

Dynamic Programming – empirical results

[TODO]

Exact methods – discussion of results

[TODO]

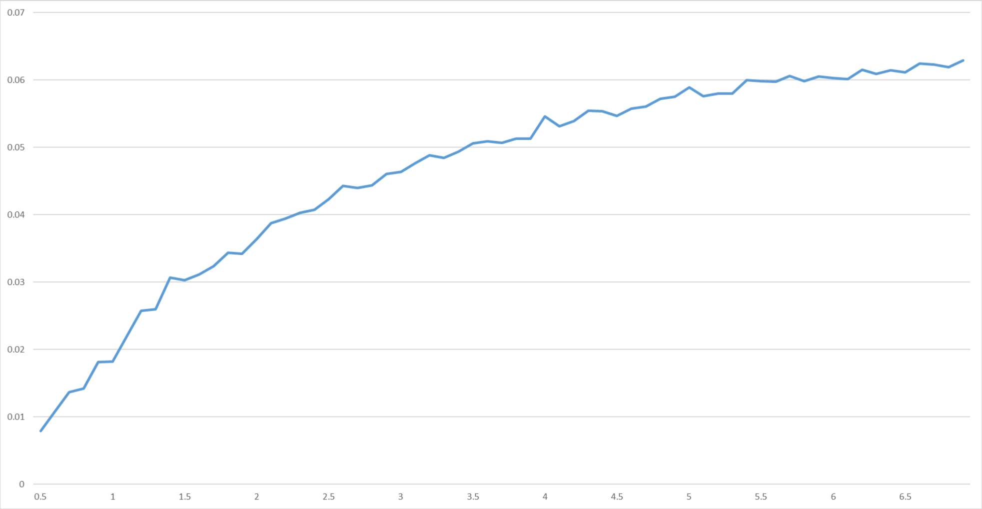
Greedy – theory

Find the variable that appears most often and assign it accordingly to maximize …

```
L ← ∅
for w ∈ {x_1, ¬x_1, ..., x_n, ¬x_n} do
    Count occurrences of w in φ
    Append pair (w, count of occurrences of w in φ) to L
Sort L with respect to the second component
for (w, c) ∈ L do
    Set w to True            ▷ If w = ¬x_i then set x_i to False
return count of satisfied clauses
```

Cost: $O(n \log n)$ assuming the use of an $O(n \log n)$ sorting algorithm.

Greedy method – empirical results



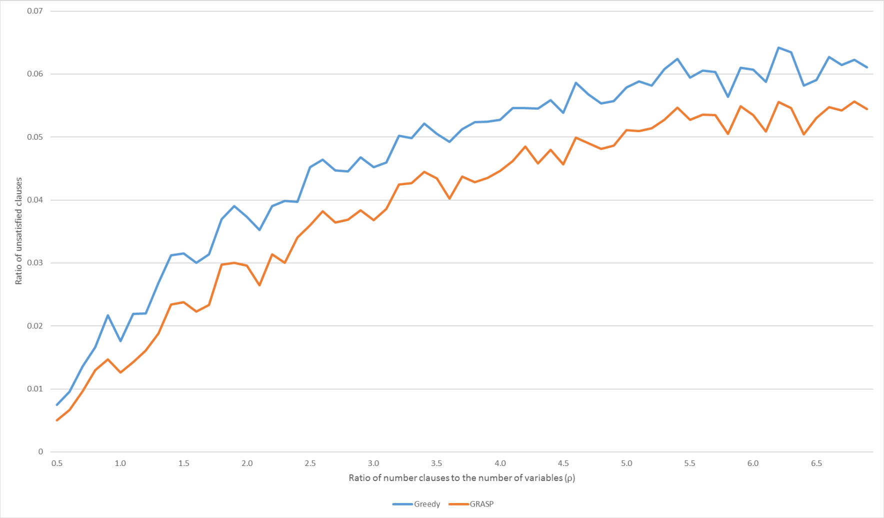
Average ratio of clauses unsatisfied by Greedy for $\rho = 0.5, \ldots, 7$.

GRASP – theory

best_candidate $\leftarrow \emptyset$
while (termination condition is not met) **do**
    greedy_candidate $\leftarrow$ ConstructGreedyRandomizedSolution()
    grasp_candidate $\leftarrow$ LocalSearch(*greedy_candidate*)
    if f(*grasp_candidate*) $<$ f(*best_candidate*) **then**
        best_candidate $\leftarrow$ *grasp_candidate*
return *best_candidate*

- "termination condition" is simply to repeat a fixed number of times, e.g. 100 times.
- f gives the ratio of unsatisfied clauses to ℓ. Objective is to minimize it.
- *ConstructGreedyRandomizedSolution*() works like Greedy but shuffles L in blocks of a given size. [TODO: EXPLAIN MORE]
- *LocalSearch*() works by flipping the variables' assignment.

GRASP – empirical results



Results for $\rho = 0.5, \ldots, 7$ and $n = 20$.

Ant Colony Optimization – theory

initialize weights
while termination criterion is not satisfied **do**
    generate population *sp* of candidate solutions using subsidiary randomized constructive search
    perform subsidiary local search on *sp*
    adapt weights based on *sp*
return *s*

The subsidiary constructive search uses weights (pheromone trails) and heuristic information.

Ant Colony Optimization – empirical results

Approximation methods – discussion of results

[TODO]

Special cases

1. $n = 1$. If x and $\neg x$ appear in the same clause then it becomes a tautology, and the clause can be ignored. Otherwise $x \vee x \vee x = x$ and $\neg x \vee \neg x \vee \neg x = \neg x$. So ϕ simplifies to a conjunction of terminals, whose satisfiability is easy to establish. [TODO: details?]
2. $n = 2$. We get 2-SAT which is in **P**. [TODO: details?]
3. $\ell = 1$. Always satisfiable. [TODO: true for $\ell \leq n$?]

Conclusion

- If instance is a special case then can be solved in polynomial time.
- Exhaustive search useful when n is small.
- Dynamic programming useful when ℓ is small.
- Otherwise, use GRASP, ACO, or other metaheuristics for approximate solutions.

Reflection

References

- Hoos, H. and Stutzler, T. (2005) **Stochastic Local Search: Foundations and Applications.** Morgan Kaufmann
- Garey, S. and Johnson, D. (1979) **Computers and Intractability: A Guide to the Theory of NP-Completeness.** Freeman