Investigating 3SAT

(Guide presentation for 380CT Coursework 2)

Kamal Bentahar

29th 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) given by

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

where each **clause** c_m has the form $\alpha \vee \beta \vee \gamma$, where each of α, β, γ is a **literal**: a variable x_i or its negation $\bar{x}_i$.

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

$$\{x_1, \bar{x}_1, x_2, \bar{x}_2, \ldots, x_n, \bar{x}_n\}$$

uniformly at random. Do not allow clauses including both x_i and $\bar{x}_i$ (tautological clauses). [2].

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

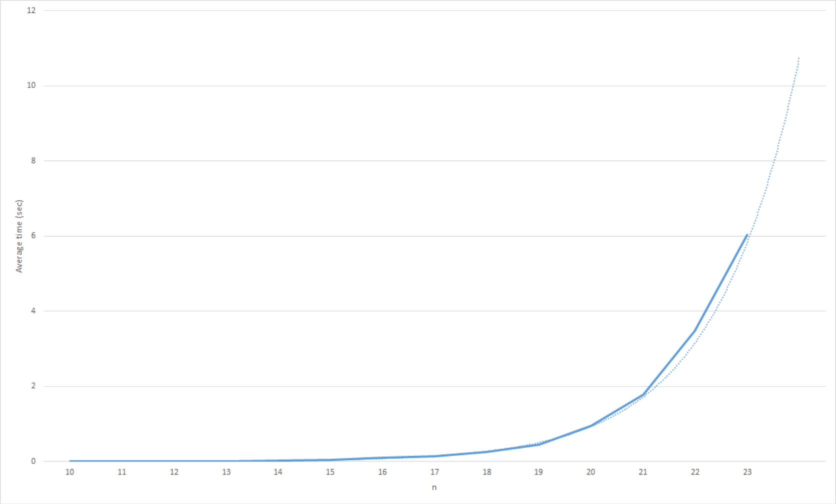
Exhaustive search – theory

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 for randomly generated instances with $\rho = 3$.
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

- Both exhaustive search and dynamic programming exhibit exponential running time:
- Exhaustive search is linear in ℓ but exponential in n. So it is useful when n is small even if ℓ is large.
- Dynamic Programming is constant in n [VERIFY!] but exponential in ℓ. So it is useful when ℓ is small even if n is large.
- ...

Greedy – theory

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

$L \leftarrow \emptyset$
for $w \in \{x_1, \bar{x}_1, \ldots, x_n, \bar{x}_n\}$ **do**
    Count occurrences of w in ϕ
    Append pair $(w, \text{count of occurrences of } w \text{ in } \phi)$ to L
Sort L with respect to the second component
for $(w, c) \in L$ **do**
    Set w to True ▷ If $w = \bar{x}_i$ 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.

GRASP – theory

$best\_candidate \leftarrow \emptyset$
while (termination condition is not met) **do**
  $greedy\_candidate \leftarrow \text{ConstructGreedyRandomizedSolution}()$
  $grasp\_candidate \leftarrow \text{LocalSearch}(greedy\_candidate)$
  if $f(grasp\_candidate) < f(best\_candidate)$ **then**
    $best\_candidate \leftarrow grasp\_candidate$
return $best\_candidate$

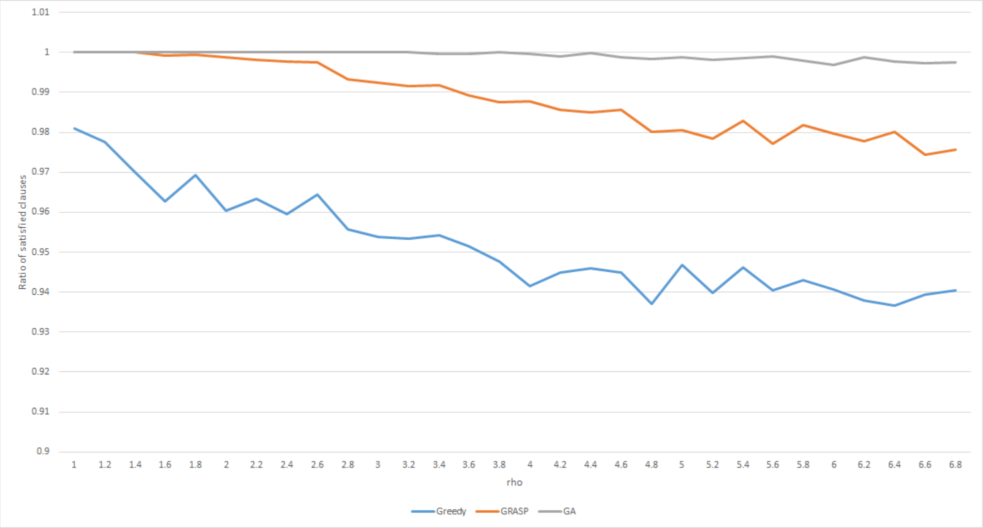
- "termination condition" is simply to repeat a fixed number of times, e.g. 100 times.
- f gives the ratio of unsatisfied clauses to ℓ. Objective is to minimize it.
- $ConstructGreedyRandomizedSolution()$ works like Greedy but shuffles L in blocks of a given size. [TODO: EXPLAIN MORE]
- $LocalSearch()$ works by flipping one variable's assignment at a time, and seeing if f improves. The best improvement is selected after trying all of them. [3]

Genetic Algorithm

determine initial population *p*
while termination criterion not satisfied **do**
    generate set *pr* of new candidate by recombination
    generate set *pm* of new candidates from *p* and *pr* by mutation
    select new population *p* from candidates in *p*, *pr*, *pm*
return fittest candidate found

- Initial population: generate 20 candidates by randomly assigning True/False values to the variables.
- termination criterion is to simply repeat 20 times.
- Recombination is done by cutting a pair of candidates at some point (*ab cd*) then creating the possible combinations *ad* and *cb*.
- Each candidate is mutated by flipping one random variable with probability 20%.
- The candidates are then sorted according to the objective function, and the fittest are kept.

Approximation – empirical results



Approximation – discussion of results

For $1 \leq \rho < 7$ we notice that:

- Greedy gets about 94-98% of the clauses satisfied.
- GRASP improves this to about 97-100%.
- GA performs better than GRASP at about 99-100%.
- For all of them, the quality decreases as ρ increases.
- ...

Special cases

1. $n = 1$. If x and $\bar{x}$ appear in the same clause then it becomes a tautology, and the clause can be ignored. Otherwise $x \vee x \vee x = x$ and $\bar{x} \vee \bar{x} \vee \bar{x} = \bar{x}$. 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 $\ell \leq n$?]
4. ...

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, GA, or other metaheuristics for approximate solutions.

Reflection

References

[TODO: format citations and list of references appropriately]

- Garey, S. and Johnson, D. (1979) **Computers and Intractability: A Guide to the Theory of NP-Completeness.** Freeman
- Hoos, H. and Stutzler, T. (2005) **Stochastic Local Search: Foundations and Applications.** Morgan Kaufmann
- Koutsoupias, E., & Papadimitriou, C. H. (1992). **On the greedy algorithm for satisfiability.** Information Processing Letters, 43(1), 53-55.