Exact Exponential Algorithms

CS 3AC3

Ryszard Janicki

Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada

This material is not covered by the textbook

Coping with NP-completeness

Q. Suppose I need to solve an NP-complete problem. What should I do?

A. Sacrifice one of three desired features.

1. Solve arbitrary instances of the problem.
2. Solve problem to optimality.
3. Solve problem in polynomial time.

Coping strategies.

1. Design algorithms for special cases of the problem.
2. Design approximation algorithms or heuristics.
3. **Design algorithms that may take exponential time.**

Exact exponential algorithms

- Complexity theory deals with worst-case behavior.
- Instances you want to solve may be “easy”.

Exact algorithms for 3-satisfiability

3-SAT. SAT where each clause contains exactly 3 literals (and each literal corresponds to a different variable).

$$\Phi = (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4)$$

yes instance: x_1 = true, x_2 = true, x_3 = false, x_4 = false

Theorem (Brute Force)

Given a 3-SAT instance with n variables and m clauses, the brute-force algorithm takes $O((m+n)2^n)$ time.

Proof.

- There are 2^n possible truth assignments to the n variables.
- We can evaluate a truth assignment in $O(m+n)$ time.

□

Exact algorithms for 3-satisfiability

A recursive framework. A 3-SAT formula Φ is either empty or the disjunction of a clause $(\ell_1 \vee \ell_2 \vee \ell_3)$ and a 3-SAT formula Φ' with one fewer clause.

$$\begin{aligned}
\Phi &= (\ell_1 \vee \ell_2 \vee \ell_3) \wedge \Phi' \\
&= (\ell_1 \wedge \Phi') \vee (\ell_2 \wedge \Phi') \vee (\ell_3 \wedge \Phi') \\
&= (\Phi' \mid \ell_1 = true) \vee (\Phi' \mid \ell_2 = true) \vee (\Phi' \mid \ell_3 = true)
\end{aligned}$$

Notation. $\Phi \mid x = true$ is the simplification of Φ by setting x to $true$.

Ex.

- $\Phi \quad = (x \vee y \vee \neg z) \wedge (x \vee \neg y \vee z) \wedge (w \vee y \vee \neg z) \wedge (\neg x \vee y \vee z)$.
- $\Phi' \quad = (x \vee \neg y \vee z) \wedge (w \vee y \vee \neg z) \wedge (\neg x \vee y \vee z)$.
- $(\Phi' \mid x = true) \quad = (w \vee y \vee \neg z) \wedge (y \vee z)$.

each clause has ≤ 3 literals

Exact algorithms for 3-satisfiability

A recursive framework. A 3-SAT formula Φ is either empty or the disjunction of a clause $(\ell_1 \vee \ell_2 \vee \ell_3)$ and a 3-SAT formula Φ' with one fewer clause.

```
3-SAT (Φ)
IF Φ is empty RETURN true.
(ℓ1 ∨ ℓ2 ∨ ℓ3) ∧ Φ' ← Φ.
IF 3-SAT(Φ' | ℓ1 = true) RETURN true.
IF 3-SAT(Φ' | ℓ2 = true) RETURN true.
IF 3-SAT(Φ' | ℓ3 = true) RETURN true.
RETURN false.
```

Theorem. The brute-force 3-SAT algorithm takes $O(\text{poly}(n)\, 3^n)$ time.

Pf. $T(n) \le 3T(n-1) + \text{poly}(n)$. ▪

$(\ell_1 \vee \ell_2 \vee \ell_3) \wedge \Phi' \leftarrow \Phi$ means '*represent* Φ as $(\ell_1 \vee \ell_2 \vee \ell_3) \wedge \Phi'$'.

Exact algorithms for 3-satisfiability

Key observation. The cases are not mutually exclusive. Every satisfiable assignment containing clause $(\ell_1 \vee \ell_2 \vee \ell_3)$ must fall into one of 3 classes:

- ℓ_1 is *true*.
- ℓ_1 is *false*; ℓ_2 is *true*.
- ℓ_1 is *false*; ℓ_2 is *false*; ℓ_3 is *true*.

```
3-SAT ($\Phi$)

IF $\Phi$ is empty RETURN true.
$(\ell_1 \vee \ell_2 \vee \ell_3) \wedge \Phi' \leftarrow \Phi$.
IF 3-SAT($\Phi' \mid \ell_1 = true$)                                  RETURN true.
IF 3-SAT($\Phi' \mid \ell_1 = false, \ell_2 = true$)                  RETURN true.
IF 3-SAT($\Phi' \mid \ell_1 = false, \ell_2 = false, \ell_3 = true$)  RETURN true.
RETURN false.
```

Exact algorithms for 3-satisfiability

Theorem. The brute-force algorithm takes $O(1.84^n)$ time.

Pf. $T(n) \leq T(n-1) + T(n-2) + T(n-3) + O(m+n)$. ▪

largest root of $r^3 = r^2 + r + 1$

```
3-SAT ($\Phi$)

IF $\Phi$ is empty RETURN true.
$(\ell_1 \vee \ell_2 \vee \ell_3) \wedge \Phi' \leftarrow \Phi$.
IF 3-SAT($\Phi' \mid \ell_1 = true$)                              RETURN true.
IF 3-SAT($\Phi' \mid \ell_1 = false, \ell_2 = true$)                  RETURN true.
IF 3-SAT($\Phi' \mid \ell_1 = false, \ell_2 = false, \ell_3 = true$)      RETURN true.
RETURN false.
```

Theorem. There exists a $O(1.33334^n)$ deterministic algorithm for 3-SAT.

SAT Problem: Idea

- An example of a *Boolean formula*:

$$\Phi = (\overline{x} \wedge y) \vee (x \wedge \overline{z}),$$

where $\overline{x}$ means $\neg x$, so $x = 0 \iff \overline{x} = 1$ and $x = 1 \iff \overline{x} = 0$.

Definition

A Boolean formula Φ is **satisfiable** if so some assignment of 0's and 1's to the variables makes the formula to eveluate to 1.

- $(\overline{x} \wedge y) \vee (x \wedge \overline{z}) = 1$ if $x = 0, y = 1, z = 0$.
  This formula is satisfiable.
- $(\overline{x} \wedge y) \wedge (x \wedge \overline{z})$ is never 1, always 0.
  This formula is not satisfiable.

Exact algorithms for SAT

- DPPL algorithm. Highly-effective backtracking procedure.
  - Splitting rule: assign truth value to literal; solve both possibilities.
  - Unit propagation: clause contains only a single unassigned literal.
  - Pure literal elimination: if literal appears only negated or unnegated.
- Chaff. State-of-the-art SAT solver.
  - Solves real-world SAT instances with $\sim$ 10K variables.
- There are many other efficient SAT-solvers.

Exact algorithms for Traveling Salesman Problem (TSP) and Hamilton cycle

TSP. Given a set of n cities and a pairwise distance function $d(u, v)$, is there a tour of length $\leq D$?

HAM-CYCLE. Given an undirected graph $G = (V, E)$, does there exist a simple cycle Γ that contains every node in V ?

Theorem. The brute-force algorithm for TSP (or HAM-CYCLE) takes $O(n!)$ time.

Pf.

- There are $\frac{1}{2}(n-1)!$ tours.
- Computing the length of a tour takes $O(n)$ time. ▪

Note. The function $n!$ grows exponentially faster than 2^n.

- $2^{40} = 1099511627776 \sim 10^{12}$.
- $40! = 815915283247897734345611269596115894272000000000 \sim 10^{48}$.

Exact algorithms for TSP and Hamilton cycle

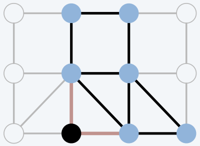
Theorem

There exists a $O(n^2 2^n)$ time algorithm for TSP (and HAMILTON-CYCLE).

Proof (Dynamic Programming).

- Define $c(s, v, X)$ = cost of cheapest path between s and v that visits every node in X exactly once (and uses only nodes in X).
- Observe $OPT = \min_{v \neq s} c(s, v, V) + c(v, s)$.
- There are $n\,2^n$ subproblems and they satisfy the recurrence:

$$c(s,v,X) = \begin{cases} c(s,v) & \text{if } |X| = 2 \\ \min_{u \in X \setminus \{s,v\}} c(s,u,X \setminus \{v\}) + c(u,v) & \text{if } |X| > 2. \end{cases}$$



- The values $c(s, v, X)$ can be computed increasing order of the cardinality of X. ▪

□

Exact algorithms for Hamilton cycle

Theorem

There exists a $O(1.657^n)$ time randomized algorithm for HAMILTON-CYCLE.

Euclidean traveling salesperson problem

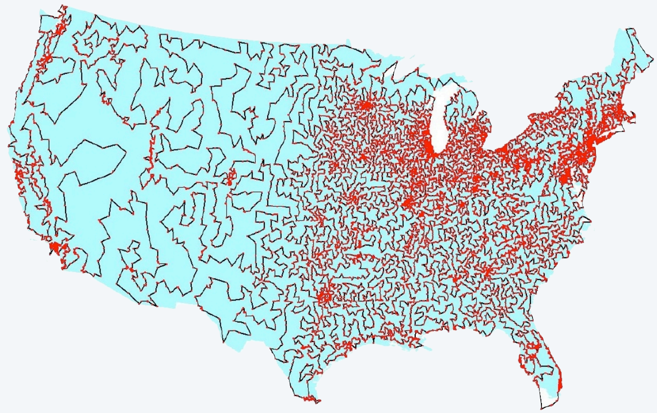
Euclidean TSP. Given n points in the plane and a real number L, is there a tour that visit every city exactly once that has distance $\leq L$?

Theorem

Given n points in the plane, for any constant $\varepsilon > 0$, there exists a poly-time algorithm to find a tour whose length is at most $(1 + \varepsilon)$ times that of the optimal tour.

Concorde TSP solver

- Concorde TSP solver is very efficient program that uses plenty of various techniques and heuristic to solve real life TSP problems. It solved all 110 TSP benchmarks.



13509 cities in the USA and an optimal tour