The Traveling Salesman Problem for Cubic Graphs

David Eppstein

Univ. of California, Irvine
School of Information and Computer Science
What to do when a problem is NP-complete?
What to do when a problem is NP-complete?

Give up
What to do when a problem is NP-complete?

- Give up
- Use heuristics
What to do when a problem is NP-complete?

Give up

Use heuristics

Approximate the correct solution
What to do when a problem is NP-complete?

Give up

Use heuristics

Approximate the correct solution

Find easier special cases
What to do when a problem is NP-complete?

- Give up
- Use heuristics
- Approximate the correct solution
- Find easier special cases
- Do worst-case analysis with exponential time bounds
What to do when a problem is NP-complete?

- Give up
- Use heuristics
- Approximate the correct solution
- Find easier special cases
- Do worst-case analysis with exponential time bounds
Why study worst case time bounds for exponential algorithms?

Worst case time analysis is most important for slower compute-bound processes

Better time bounds improve solvable instance size by constant factor; Moore’s law much less effective

Less well-studied, so more interesting problems

Interesting gap between theory (exponential times) and practice (much smaller exponentials)
Problems studied in this paper

Hamiltonian cycle
Input: undirected, unweighted graph
Output: simple cycle containing all vertices, if one exists
Decision version is NP-complete

Traveling salesman problem (TSP)
Input: undirected graph with edge weights
Output: shortest Hamiltonian cycle, if one exists
Decision version is NP-complete

Cycle counting
Input: undirected, unweighted graph
Output: number of simple cycles containing all vertices
#P-complete

Weighted cycle counting
Input: undirected graph with edge weights
Output: Sum, over all Hamiltonian cycles, of product of weights of edges in cycle
Best previously-known TSP algorithm: dynamic programming

Choose arbitrary starting vertex $v$

For each pair $(S, w)$, where $S$ is any set of vertices that contains both $v$ and $w$, let $A(S, w)$ be the min weight of a path from $v$ to $w$ via $S$

Compute $A(S, w) = \min \{ A(S\setminus\{w\},x) + d(x,w) \mid x \in S\setminus\{v\} \}$

Global TSP length = $A(V(G), v)$

Total time: $O(2^n n^2)$
Total space: $O(2^n n^{1/2})$
[order subproblems by set size, only store problems with similar size]

Same approach works also for counting and weighted counting
New results

TSP and related problems for graphs with maximum degree at most three

Hamiltonian cycle and TSP

time: $O(2^n/3)$
space: linear

Cycle counting and weighted cycle counting

time: $O(2^{3n/8} n^{O(1)})$
space: polynomial
Main ideas of the new algorithms

**Generalize** the problem
(allow graphs with forced edges)

**Branch and simplify**
(complicated case analysis)

Improve analysis via **polynomial-time special case**
(reduction from TSP to minimum spanning tree)
Forced edges

Input = graph + set of "forced edges" (shown thick)
Output cycle(s) must include all forced edges

Example: $K_4$ with no forced edges has three Hamiltonian cycles
Forced edges

Input = graph + set of "forced edges" (shown thick)
Output cycle(s) must include all forced edges

Example: $K_4$ with one forced edge has only two Hamiltonian cycles
Forced edges

Equivalent in effect to inserting a vertex inside the edge

But, unlike new vertex, doesn’t count against total size of graph

More forced edges = fewer possible cycles = easier problem instance
Problem simplifications

Certain configurations force no Hamiltonian cycle to exist

Vertex with degree one

Three mutually incident forced edges

If encountered, immediately stop this branch of the search algorithm
Problem simplifications

Certain configurations allow additional edges to be forced or removed

- Vertex with degree two
  - both edges can be forced

- Two mutually incident forced edges
  - remove third edge (if it exists)
  - and contract to a single forced edge
Problem simplifications

Triangles in the input graph can be contracted (Delta-Y transformation)

Can be made to preserve weights of each Hamiltonian cycle and respect forced edges within the triangle
Branch and simplify

Standard general technique for exponential algorithms

Choose a variable (unforced edge)

- Force the edge
- Perform all possible simplifications
- Recurse on smaller instance
  - finds all Hamiltonian cycles containing the edge

- Restore original graph
- Try removing the edge
- Perform all possible simplifications
- Recurse on smaller instance
  - finds all Hamiltonian cycles not containing the edge

Choose variable to maximize simplification
(complex case analysis)
Analysis

Let $U = \#$ unforced edges in instance, $T(U) = \#$ subproblems in algorithm

Worst case:
Cycle of four unforced edges with all neighbors forced
Each subproblem removes or forces all edges in cycle

$$T(U) = 2T(U - 4)$$

Not-quite-as-bad case:
Cycle of four unforced edges with two neighbors forced
Removing side between forced neighbors forces two adjacent sides
Forcing side removes adjacent sides, forces three other edges

$$T(U) = T(U - 3) + T(U - 6)$$

Overall $T(U) = O(2^{\frac{U}{4}}) = O(2^{\frac{3n}{8}})$
Polynomial special case

If all components of unforced edges form 4-cycles, TSP can be transformed to minimum spanning tree

In general, perform branch and simplify until reaching polynomial special case

Removes worst case from previous algorithm

New analysis (using different variable for subproblem size):
Total time = total # subproblems = $O(2^{n/3})$
Conclusions

Significantly more efficient algorithms
for interesting special case of TSP, related problems

Open problems

Extend TSP-MST reduction to cycle counting?

Any hope of $o(2^n)$ for general case of TSP?

Generalize low degree to low total number of edges?

Other important NP-hard special cases not yet considered?