CS 473: Algorithms

Chandra Chekuri
chekuri@cs.illinois.edu
3228 Siebel Center

University of Illinois, Urbana-Champaign

Fall 2009

Part I

Heuristics

Coping with Intractability

Some general things that people do.

- Consider special cases of the problem which may be tractable.
- Run inefficient algorithms (for example exponential time algorithms for NP-hard problems) augmented with (very) clever heuristics
  - stop algorithm when time/resources run out
  - use massive computational power
- Exploit properties of instances that arise in practice which may be much easier. Give up on hard instances, which is ok.
- Settle for sub-optimal solutions, especially for optimization problems

NP and *EXP*

EXP: all problems that have an exponential time algorithm.

Proposition

$NP \subseteq EXP$

NP and *EXP*

EXP: all problems that have an exponential time algorithm.

Proposition

$NP \subseteq EXP$

Proof.

Let $X \in NP$ with certifier C. To prove $X \in EXP$, here is an algorithm for X. Given input s,

- For every t, with $|t| \leq p(|s|)$ run $C(s, t)$; answer "yes" if any one of these calls returns "yes", otherwise say "no". □

Every problem in NP has a brute-force "try all possibilities" algorithm that runs in exponential time.

Examples

- SAT: try all possible truth assignment to variables
- Independent set: try all possible subsets of vertices
- Vertex cover: try all possible subsets of vertices

Improving brute-force via intelligent backtracking

- Backtrack search: enumeration with bells and whistles to "heuristically" cut down search space.
- Works quite well in practice for several problems, especially for small enough problem sizes.

Backtrack Search Algorithm for SAT

Input: CNF Formula φ on n variables $x_1, \ldots, x_n$ and m clauses
Output: Is φ satisfiable or not.

1. Pick a variable x_i
2. φ' is CNF formula obtained by setting $x_i = 0$ and simplifying
3. Run a simple (heuristic) check on φ': returns "yes", "no" or "not sure"
   - If "not sure" recursively solve φ'
   - If φ' is satisfiable, return "yes"
4. φ'' is CNF formula obtained by setting $x_i = 1$
5. Run simple check on φ'': returns "yes", "no" or "not sure"
   - If "not sure" recursively solve φ''
   - If φ'' is satisfiable, return "yes"
6. Return "no"

Certain part of the search space is pruned.

Example

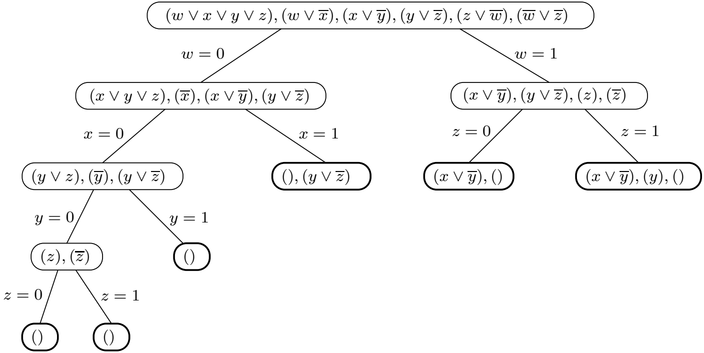


Figure: Backtrack search. Formula is not satisfiable.

Figure from Dasgupta etal book.

Backtrack Search Algorithm for SAT

How do we pick the order of variables?

Backtrack Search Algorithm for SAT

How do we pick the order of variables? Heuristically! Examples:

- pick variable that occurs in most clauses first
- pick variable that appears in most size 2 clauses first
- ...

Backtrack Search Algorithm for SAT

How do we pick the order of variables? Heuristically! Examples:

- pick variable that occurs in most clauses first
- pick variable that appears in most size 2 clauses first
- ...

What are quick tests for satisfiability?

Backtrack Search Algorithm for SAT

How do we pick the order of variables? Heuristically! Examples:

- pick variable that occurs in most clauses first
- pick variable that appears in most size 2 clauses first
- ...

What are quick tests for satisfiability? Depends on known special cases and heuristics. Examples.

- Obvious test: return "no" if empty clause, "yes" if no clauses left and otherwise "not sure"
- Run obvious test and in addition if all clauses are of size 2 then run 2-SAT polynomial time algorithm
- ...

Branch-and-Bound: backtracking for optimization probs

Intelligent backtracking can be used also for optimization problems. Consider a minimization problem.
Notation: for instance I, $opt(I)$ is optimum value on I.

P_0 initial instance of given problem.

- Keep track of the best solution value B found so far. Initialize B to be crude upper bound on $opt(I)$.
- Let P be a subproblem at some stage of exploration.
- If P is a complete solution, update B.
- Else use a lower bounding heuristic to quickly/efficiently find a lower bound b on $opt(P)$.
  - If $b \geq B$ then prune P
  - Else explore P further by breaking it into subproblems and recurse on them.
- Output best solution found.

Example: Vertex Cover

Given $G = (V, E)$, find a minimum sized vertex cover in G.

- Initialize $B = n - 1$.
- Pick a vertex u. Branch on u: either choose u or discard it.
- Let b_1 be a lower bound on $G_1 = G - u$.
- If $1 + b_1 < B$, recursively explore G_1
- Let b_2 be a lower bound on $G_2 = G - u - N(u)$ where $N(u)$ is the set of neighbours of u.
- If $|N(u)| + b_2 < B$, recursively explore G_2
- Output B.

Example: Vertex Cover

Given $G = (V, E)$, find a minimum sized vertex cover in G.

- Initialize $B = n - 1$.
- Pick a vertex u. Branch on u: either choose u or discard it.
- Let b_1 be a lower bound on $G_1 = G - u$.
- If $1 + b_1 < B$, recursively explore G_1
- Let b_2 be a lower bound on $G_2 = G - u - N(u)$ where $N(u)$ is the set of neighbours of u.
- If $|N(u)| + b_2 < B$, recursively explore G_2
- Output B.

How do we compute a lower bound?
One possibility: solve an LP relaxation that we saw in previous lecture.

Local Search

Local Search: a simple and broadly applicable heuristic method

- Start with some arbitrary solution s
- Let $N(s)$ be solutions in the "neighbourhood" of s obtained from s via "local" moves/changes
- If there is a solution $s' \in N(s)$ that is better than s, move to s' and continue search with s'
- Else, stop search and output s.

Local Search

Main ingredients in local search:

- Initial solution
- Definition of neighbourhood of a solution
- Efficient algorithm to find a good solution in the neighbourhood

Example: TSP

TSP: Given a complete graph $G = (V, E)$ with c_{ij} denoting cost of edge (i, j), compute a Hamiltonian cycle/tour of minimum edge cost.

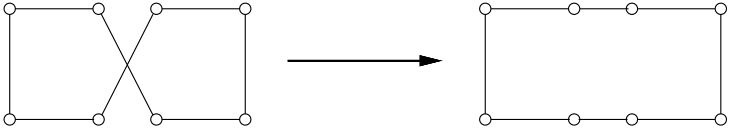
Example: TSP

TSP: Given a complete graph $G = (V, E)$ with c_{ij} denoting cost of edge (i, j), compute a Hamiltonian cycle/tour of minimum edge cost.

2-change local search:

- Start with an arbitrary tour s_0
- For a solution s define s' to be a neighbour if s' can be obtained from s by replacing two edges in s with two other edges.
- For a solution s at most $O(n^2)$ neighbours and one can try all of them to find an improvement.

TSP: 2-change example



TSP: 2-change example

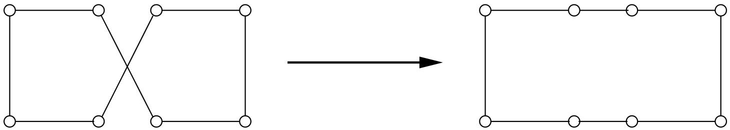
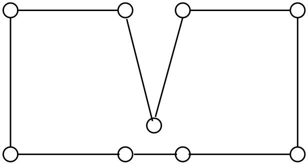
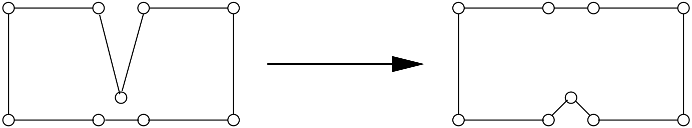


Figure below shows a bad local optimum for 2-change heuristic



TSP: 3-change example

3-change local search: swap 3 edges out.



Neighbourhood of s has now increased to a size of $\Omega(n^3)$
Can define k-change heuristic where k edges are swapped out.
Increases neighbourhood size and makes each local improvement step less efficient.

Local Search Variants

Local search terminates with a local optimum which may be far from a global optimum. Many variants to improve plain local search.

- Randomization and restarts. Initial solution may strongly influence the quality of the final solution. Try many random initial solutions.
- Simulated annealing is a general method where one allows the algorithm to move to worse solutions with some probability. At the beginning this is done more agressively and then slowly the algorithm converges to plain local search. Controlled by a parameter called "temperature".
- Tabu search. Store already visited solutions and do not visit them again (they are "taboo").

Heuristics

Several other heuristics used in practice.

- Heuristics for solving integer linear programs such as cutting planes, branch-and-cut etc are quite effective.
- Heuristics to solve SAT (SAT-solvers) have gained prominence in recent years
- Genetic algorithms
- ...

Heuristics design is somewhat adhoc and depends heavily on the problem and the instances that are of interest. Rigorous analysis is sometimes possible.

Part II

Finals and Closing Thoughts

Topics for Finals

- Recursion: reduce problem to smaller instance(s) of itself
  - Divide and Conquer: divide into multiple pieces, solve recursively and build solution to the original instance
  - Dynamic Programming: recursion with memoization, runs in polynomial time if number of sub-problems is polynomial.
- Graph Algorithms: DFS/BFS, directed graphs, DAGs, strong components, MST, shortest paths
- Greedy Algorithms: always think of a proper proof!
- Network Flows and Applications
- Polynomial time Reductions, P, NP, co-NP, NP-Completeness
- Light touch: Linear Programming, Integer Programming, Approximation Algorithm

Topics I wish I had time for

- Data structures - hashing, splay trees etc.
- Randomization in algorithms
- Basic lower bounds
- More on heuristics and applications
- Experimental evaluation

Theoretical Computer Science

- Algorithms: find efficient ways to solve particular problems
- Computational Complexity: understand nature of computation — classification of problems into classes (P, NP, coNP) and their relationships, limits of computation.
- Logic, Languages and Formal Methods

Form the foundations for computer "science"

The Computational Lens

The Algorithm: Idiom of Modern Science by Bernard Chazelle
http://www.cs.princeton.edu/chazelle/pubs/algorithm.html

Computation has gained ground as *fundamental* artifact in mathematics and science.

- nature of proofs, P vs NP, complexity, ...
- quantum computation and information
- computational biology and the biological processes, ...

Standard question in math and sciences: Is there a *solution/algorithm*?

The Computational Lens

The Algorithm: Idiom of Modern Science by Bernard Chazelle
http://www.cs.princeton.edu/chazelle/pubs/algorithm.html

Computation has gained ground as *fundamental* artifact in mathematics and science.

- nature of proofs, P vs NP, complexity, ...
- quantum computation and information
- computational biology and the biological processes, ...

Standard question in math and sciences: Is there a *solution/algorithm*?
New: Is there an *efficient* solution/algorithm?

Questions?

Questions?

Final Exam: Thursday, Dec 17th, 1.30 - 4.30pm in 1404 Siebel.

Thanks!