Computation: Potentials and Limitations

Edward Tsang

Computation: Potentials and Limits

- Computers are fast
- How fast can we solve a problem?
- It depends on your algorithm
- How to measure the speed of algorithms?
- What problem-algorithms are "intractable"
- What can we do about it?
  - Approximations
  - Heuristics

17 October 2011

Dealing with data

- Suppose you believe the following:
  - Whenever the short-term moving average crosses with the long-term moving average from below, it signals a chance to buy
  - How to turn that into a concrete trading rule?

17 October 2011

FTSE 2009.08.18-2010.10.22

17 October 2011

Moving Average Rules to Find

- Let
  - m-MA be the m-days moving average
  - n-MA be the n-days moving average
  - m < n
- Possible rules to find:
  - If the m-MA ≤ n-MA, on day d, but m-MA > n-MA on day d+1, then buy
  - If the m-MA ≥ n-MA, on day d, but m-MA < n-MA on day d+1, then sell

17/10/2011

Learning Moving Average Rules

- To find Moving Averages (MAs)
  - You need to compare m-days and n-days MA
  - Where m < n
- Not all m and n work with each other
- To find a good rule, you have to try different m and n values, one at a time
- You can examine how good a particular (m, n) is by testing it with past data

17/10/2011

Computation consideration

- Suppose you decide that m is in [1, 20] and n is in [21, 70]
- You have 20 × 50 = 1,000 combinations to evaluate
- Suppose each combination takes 1 second to evaluate
- So evaluation takes 1,000 seconds
  - or 17 minutes, which is acceptable

Finding more robust rules

- Suppose you want to find separate m and n for buying and selling
- Now you need 1,000 seconds to find a buying rule, and another 1,000 seconds for selling
  - You need 1,000,000 seconds to find combinations
  - That is 115 days
- You could speed it up with multiple computers
  - 115 computers will take 1 day approximately

Realistic rules are more complex

- Simple rules would have been found by others
- Prices will be changed to reflect rules found
- Can you beat the market?
- Yes, by finding more complex rules
- For example, rules that relate stocks with index

Example: relating stock with index

- Let
  - $k\text{-}MA_s$ be the k-days moving average for stock s
  - $k\text{-}MA_I$ be the k-days moving average for index I
- Buy if crossing is found in both the stock and the index graph within D days:
  - $m\text{-}MA_s \le n\text{-}MA_s$ on day d, but $m\text{-}MA_s > n\text{-}MA_s$ on day d+1
  - $m\text{-}MA_I \le n\text{-}MA_I$ on day d', but $m\text{-}MA_I > n\text{-}MA_I$ on day d'+1
  - $| d - d' | \le D$

Time needed to find the complex rule

- Let D be a value between 0 to 9
- To find buying rules, 1,000 pairs of m and n's
- Total evaluations required 10,000
- Same number to find selling rules
- This time, it may take 2 seconds per evaluation
- Time required: 2×10^8 seconds to complete
  - That is 63 years!

A Closer Look at Complexity

Sudoku Puzzles

- The task is to put one digit into one square
- Each digit should appear once in a row, column or sub-square
- Solvable by constraint solvers within 2 seconds

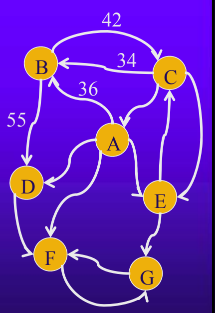
5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Finding the Shortest Path

- Given:
  - Junctions
  - Connections
  - Distance per connection (could be miles/minutes)
- Find the shortest path from A to B

Path-finding, Graph Representation

- A graph is:
  (Nodes, Arcs)
- Each Arc is a pair of nodes
- Add a distance on each arc
- Assume, for simplicity:
  - No multiple-paths between two nodes
  - All nodes are reachable



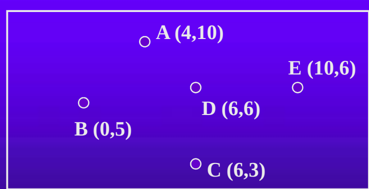
Path-finding – Dijkstra's Algorithm

- Find shortest paths from A to every other node
- For each node
  - Remember the current shortest distance from A
  - Current parent that is links to A
- Starting from A, compute one node at a time
  - Pick the remaining node x that is closest to A
  - Update distance/parent of every neighbours of x if needed
- Complexity: $O(n^2)$

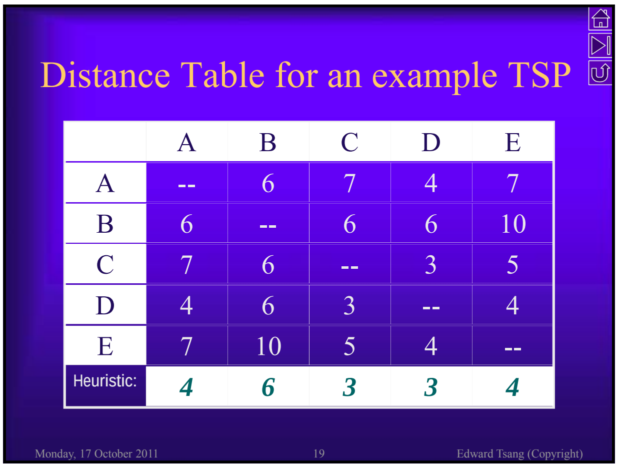
Dijkstra's Algorithm Pseudo Code

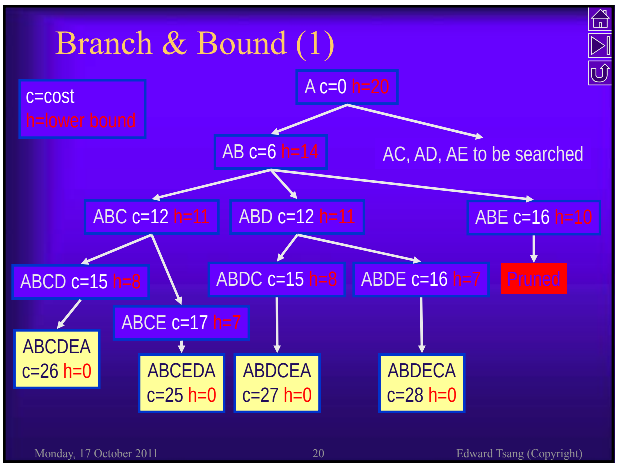
```
For each node v in graph
    parent[v] ← undefined; dist[v] ← ∞
Dist[source] ← 0
Q ← {all nodes in graph}
While Q is not empty Do
    Remove x from Q s.t. dist[x] is minimum
    For each of x's neighbour y
        alt ← dist[x] + distance[x,y]
        If alt < dist[y]
            dist[y] ← alt; parent[y] ← x
```

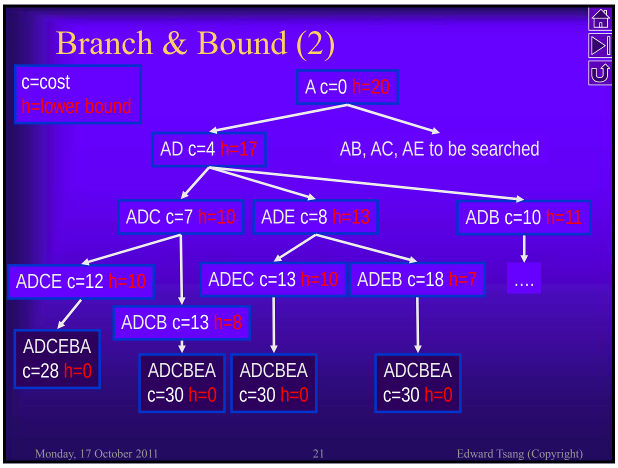

Travelling Salesman Problem (TSP)

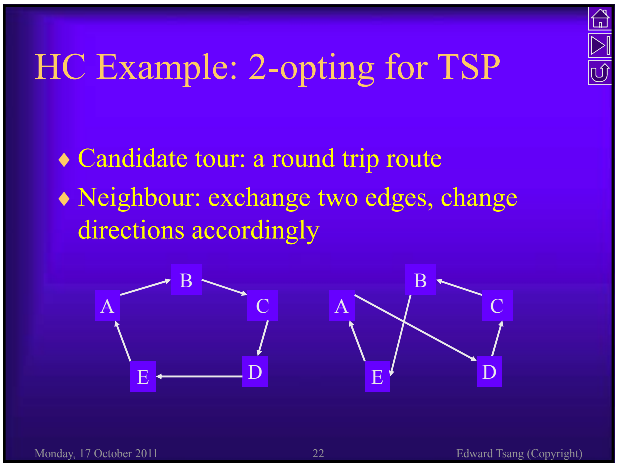


- Goal: to find *shortest route* through all cities
- Optimization involved: minimization

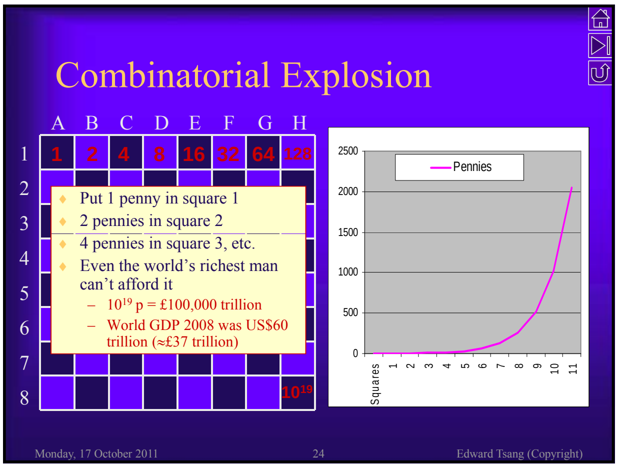

Distance Table for an example TSP
	A	B	C	D	E
A	--	6	7	4	7
B	6	--	6	6	10
C	7	6	--	3	5
D	4	6	3	--	4
E	7	10	5	4	--
Heuristic:	4	6	3	3	4
Monday, 17 October 2011
19
Edward Tsang (Copyright)


Branch & Bound (1)
c=cost
h=lower bound
A c=0 h=20
AB c=6 h=14
AC, AD, AE to be searched
ABC c=12 h=11
ABD c=12 h=11
ABE c=16 h=10
ABCD c=15 h=8
ABDC c=15 h=8
ABDE c=16 h=7
Pruned
ABCE c=17 h=7
ABCDEA c=26 h=0
ABCEDA c=25 h=0
ABDCEA c=27 h=0
ABDECA c=28 h=0
Monday, 17 October 2011
20
Edward Tsang (Copyright)


Branch & Bound (2)
c=cost
h=lower bound
A c=0 h=20
AD c=4 h=17
AB, AC, AE to be searched
ADC c=7 h=10
ADE c=8 h=13
ADB c=10 h=11
ADCE c=12 h=10
ADEC c=13 h=10
ADEB c=18 h=7
....
ADCB c=13 h=8
ADCEBA c=28 h=0
ADCBEA c=30 h=0
ADCBEA c=30 h=0
ADCBEA c=30 h=0
Monday, 17 October 2011
21
Edward Tsang (Copyright)

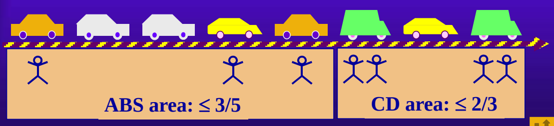

HC Example: 2-opting for TSP
♦ Candidate tour: a round trip route
♦ Neighbour: exchange two edges, change directions accordingly
A B C D E
A B C D E
Monday, 17 October 2011
22
Edward Tsang (Copyright)


List reversing ➔ 2-Opting
♦ List representation:
– A list could represent cities in sequence
♦ 2-Opting can be seen as sub-list reversing
– Easy to implement
1 3 4 8 6 5 2 7
1 3 4 5 6 8 2 7
Breaking points
Edward Tsang (Copyright)
23
Monday, 17 October 2011


Combinatorial Explosion
A B C D E F G H
1 2 3 4 5 6 7 8
1 2 4 8 16 32 64 128
♦ Put 1 penny in square 1
♦ 2 pennies in square 2
♦ 4 pennies in square 3, etc.
♦ Even the world's richest man can't afford it
– 10^19 p = £100,000 trillion
– World GDP 2008 was US$60 trillion (≈£37 trillion)
10^19
Pennies
2500
2000
1500
1000
500
0
Squares
1 2 3 4 5 6 7 8 9 10 11
Monday, 17 October 2011
24
Edward Tsang (Copyright)

Car Sequencing Problem

Options					
ABS	×	√	√	×	
CD	×	×	√	√	
…					Total:
Production:	30	30	20	40	120



Combinatorial Explosion in Car Sequencing

- Schedule 30 cars:
  - Search space: 30 factorial ≅ $\mathbf{10^{32}}$ leaf nodes
- Generously allow:
  - Explore one in every $\mathbf{10^{10}}$ leaf nodes!
  - Examine $\mathbf{10^{10}}$ nodes per second!
- Problem takes over **32 thousand years** to solve!!!
  - $\mathbf{10^{32} \div 10^{10} \div 10^{10} \div 60 \div 60 \div 24 \div 365 \cong 31{,}710}$
- How to contain ***combinatorial explosion***?

Computational complexity basics

- Let *n* measures the size of a problem
- Can we express how fast an algorithm is in term of *n*?
  - On average and at worst?
  - Expressed as $O(n^2)$, $O(\log n)$, $O(10^n)$
- Also applied to amount of memory required
- A problem that cannot be solved fast enough to be useful is called *intractable*

NP Completeness in laymen terms

- A concept in computer science
  - It is about complexity in computation
- A problem is NP-complete if finding solutions take exponential time
- Example: try all combinations of a password
  - Assume 6 characters from a to z, A to Z, 0 to 9
  - There are 62^6 (roughly 10^{10}) combinations
  - Trying 2 passwords per second takes 2 milleniums!

A little bit more technical on NP

- NP is a concept in computational complexity
  - NP: non-polynomial time complete
- Let *n* measures the size of a problem
- A problem is NP-complete if:
  - Any solution can be verified in polynomial time
    - E.g. n^2, where *n* measures the size of the problem
  - But it takes exponential time to find solutions
    - E.g. 2^n

Parallel processing for NP problems

- Suppose we need to make n decisions
  - Each decision has m choices, where m is constant
- There are m^n combinations to explore
- Suppose we use 10 processors
  - Assume linear speed-up, no overhead
- Problem will be solved in 1/10th of time
  - Getting an answer in 1 hour is better than 10 hours
  - But exploring m^n combinations may take 10^{30} years

What to do with NP problems?

- Just because a problem is NP-complete doesn't mean that it is intractable
  - Sudoku: constraint propagation
  - Linear programming: exploiting problem features
- However, most NP-complete problems are intractable in nature
  - Find approximations
  - Heuristics may help

Searching

Artificial Intelligence

≈ Knowledge representation + Search

Search Space = the set of all possible solutions under the given representation

Complete Search
Systematically explore every candidate solution in the search space

Incomplete Search
Use heuristics to search in promising areas for solutions

Stochastic Search

- Incomplete search
  - i.e. even if solutions exist, they may not be found
- Evolutionary computation
  - To evolve solutions thru maintaining a population
- Hill Climbing
  - To heuristically improve on the current solution
- Many more
  - Tabu search, guided local search, neural network, …

Conclusion

- Computers are fast
- Some problems can be solved very quickly
- However, many problems are intractable
- Computer scientists study complexity of algorithms
- This helps us decide what technique to use
  - Great if optimal solutions can be found
  - Apply approximation methods otherwise

