Solving the Traveling Salesman Problem using Branch and Bound

Robert Clark and Danny Iland
What is the Traveling Salesman Problem?

- For a group of N cities, find the shortest loop that visits all N cities exactly once.

i.e.
The shortest cycle is: 0, 4, 2, 3, 1, 0 with a cost of: 2387
Why the Traveling Salesman Problem?

- NP-hard
- Brute Force takes $O(n!) \text{ to complete}$
- Algorithms to solve take one of two forms:
  - Exact algorithms
    - Branch and Bound $O(2^n)$
    - Linear Programming
  - Heuristic algorithms
    - Nearest neighbor $O(\log n)$
    - Pairwise exchange
    - Randomized improvement (markov chains, genetic algorithms)
- Many real world applications:
  - UPS saves 3 million gallons of gasoline per year.
Branch and Bound and the TSP

Special properties of the Traveling Salesman Problem that make it suitable for Branch and Bound:

- As you build your solution, cost increases.
- Partial solutions have valid costs (lower bound costs)

Inclusion-exclusion:
- Each state can produce two discrete sets of children states: Those with a specific edge and those without.
  - Left branch (with a specific edge) removes 2*N possible edges from the pool.
  - Right branch removes 1 edge from the pool
Solving Sequentially

Put the starting state on the left queue
While ( the left queue isn't empty or the right queue isn't empty )
  if( leftQueue isn't empty ) node = leftQueue.front
  else node = rightQueue.front
  if( node is Terminal state )
    if( node.cost < bestCost )
      bestCost = node.cost
      bestRoute = node.route
  else
    Determine the best edge to split on
    if( node.leftChild.lowerBoundCost < bestCost )
      leftQueue.add(node.leftChild)
    if( node.rightChild.lowerBoundCost < bestCost )
      rightQueue.add(node.rightChild)

Print best found
Solving in Parallel

- Do the sequential algorithm, pushing onto a shared stack instead of the right queue, until the shared stack contains one node for each processor.
- Then, each process gets one of those nodes and performs the standard sequential algorithm, with one difference:
  - If a processor runs out of nodes, it sets a sharedBoolean flag to true.
  - When a processor generates a right child, it checks if the flag is set, and if so pushes its right child onto the shared stack instead of its own right stack, and resets the flag to false.
  - When the flag becomes false, the waiting process puts the first element in sharedStack on its leftStack.
Performance: Running Time

Running Time vs. Processors

$T(N,K)$ (sec)

Processors, $K$

$N = 28$

$N = 24$
Performance: Speedup

![Graph showing Speedup vs. Processors]

- N = 26
- N = 28
- N = 24

The graph illustrates the speedup achieved with increasing number of processors (K). The speedup is measured in terms of time taken (seconds) and is normalized by the number of processors (N). As the number of processors increases, the speedup also increases, indicating improved performance.
Performance: Efficiency

Efficiency vs. Processors

- $N = 26$
- $N = 28$
- $N = 24$
Performance: EDSF

EDSF vs. Processors

EDSF (N,K) (/1000)

-250 -175 -100 -25 50 125 175 250 300 375 425 500

Processors, K

N = 26
N = 28
What we learned

- Node evaluation order matters. A lot.
- Just because you have a parallelizeable problem doesn't mean you have a parallel algorithm
- Debugging problems using large data sets is very difficult.
- Finding a good problem size for useful performance metrics can be difficult.
Future Work

- Solve the Traveling Salesman Problem with other parallel algorithms:
  - Any Colony Optimization
  - Genetic Algorithms

- Changing the way we do parallel Branch and Bound:
  - Use heuristics on set of unexplored nodes to decide where to go next
  - Implement reporting to help understand what causes performance differences
    - Where in the tree the best answer was has large impact.
  - Problem specific optimizations