DM8XX Presentation
Subject 8: Erlang

Bjørn Madsen

IMADA, SDU

May 25, 2009

Solving TSP with branch and bound

Solving TSP with branch and bound

- Generate an initial solution, via a fast algorithm (eg. greedy)

Solving TSP with branch and bound

- Generate an initial solution, via a fast algorithm (eg. greedy)
- Generate an initial set of partial tours, calculate their lower bounds

Solving TSP with branch and bound

- Generate an initial solution, via a fast algorithm (eg. greedy)
- Generate an initial set of partial tours, calculate their lower bounds
- Discard any partial tour with a lower bound higher than the current best solution's length

Solving TSP with branch and bound

- Generate an initial solution, via a fast algorithm (eg. greedy)
- Generate an initial set of partial tours, calculate their lower bounds
- Discard any partial tour with a lower bound higher than the current best solution's length
- In turn add a city to each "promising" partial tour, until you have a full tour or the lower bound goes above the best length

Solving TSP with branch and bound

- Generate an initial solution, via a fast algorithm (eg. greedy)
- Generate an initial set of partial tours, calculate their lower bounds
- Discard any partial tour with a lower bound higher than the current best solution's length
- In turn add a city to each "promising" partial tour, until you have a full tour or the lower bound goes above the best length
- Replace the best solution each time a better one is found

Solving TSP with branch and bound

- Generate an initial solution, via a fast algorithm (eg. greedy)
- Generate an initial set of partial tours, calculate their lower bounds
- Discard any partial tour with a lower bound higher than the current best solution's length
- In turn add a city to each "promising" partial tour, until you have a full tour or the lower bound goes above the best length
- Replace the best solution each time a better one is found
- When all "paths" are either discarded or done, we have an optimal solution

Implementation of a distributed TSP solver

Implementation of a distributed TSP solver

- One server, several nodes

Implementation of a distributed TSP solver

- One server, several nodes
- Server contains a priority queue of partial tours as well as the current best tour

Implementation of a distributed TSP solver

- One server, several nodes
- Server contains a priority queue of partial tours as well as the current best tour
- Nodes request a partial tour when they need work

Implementation of a distributed TSP solver

- One server, several nodes
- Server contains a priority queue of partial tours as well as the current best tour
- Nodes request a partial tour when they need work
- When they have a complete tour, they offer it to the server and request more work

Implementation of a distributed TSP solver

- One server, several nodes
- Server contains a priority queue of partial tours as well as the current best tour
- Nodes request a partial tour when they need work
- When they have a complete tour, they offer it to the server and request more work
- When the queue is empty and no nodes are currently working, we have the best solution

Java version

Java version

- Using Remote Method Invocation (RMI), so nodes can call methods on objects residing on the server

Java version

- Using Remote Method Invocation (RMI), so nodes can call methods on objects residing on the server
- Needs to have an RMI registry running somewhere

Java version

- Using Remote Method Invocation (RMI), so nodes can call methods on objects residing on the server
- Needs to have an RMI registry running somewhere
- Server binds to this registry, nodes find server through registry

Java version

- Using Remote Method Invocation (RMI), so nodes can call methods on objects residing on the server
- Needs to have an RMI registry running somewhere
- Server binds to this registry, nodes find server through registry
- Everything needs to use thread-safe version or run in synchronized blocks

Erlang version

Erlang version

- Using Erlang message passing

Erlang version

- Using Erlang message passing
- Nodes need to have an Erlang system running on them

Erlang version

- Using Erlang message passing
- Nodes need to have an Erlang system running on them
- The master can spawn nodes on remote or local Erlang systems

Erlang version

- Using Erlang message passing
- Nodes need to have an Erlang system running on them
- The master can spawn nodes on remote or local Erlang systems
- With minor rewriting, nodes can join or leave mid-computation

Comparison

Single, multi-core computer

Comparison

Single, multi-core computer

- Quad-core computer

Comparison

Single, multi-core computer

- Quad-core computer
- Solving a TSP instance with 30 cities

Comparison

Single, multi-core computer

- Quad-core computer
- Solving a TSP instance with 30 cities
- Java time: around 9 seconds

Comparison

Single, multi-core computer

- Quad-core computer
- Solving a TSP instance with 30 cities
- Java time: around 9 seconds
- Erlang time:

Comparison

Single, multi-core computer

- Quad-core computer
- Solving a TSP instance with 30 cities
- Java time: around 9 seconds
- Erlang time:
- Over four minutes...

Comparison

Distributed

Comparison

Distributed

- Eight computers on IMADA

Comparison

Distributed

- Eight computers on IMADA
- Solving a TSP instance with 35 cities

Comparison

Distributed

- Eight computers on IMADA
- Solving a TSP instance with 35 cities
- Java time: 16 minutes

Comparison

Distributed

- Eight computers on IMADA
- Solving a TSP instance with 35 cities
- Java time: 16 minutes
- Erlang time:

Comparison

Distributed

- Eight computers on IMADA
- Solving a TSP instance with 35 cities
- Java time: 16 minutes
- Erlang time:
- I stopped the nodes after 2.5 hours...

Comparison

Distributed

- Eight computers on IMADA
- Solving a TSP instance with 35 cities
- Java time: 16 minutes
- Erlang time:
- I stopped the nodes after 2.5 hours...
- The Erlang version even used two processes on each node, utilizing both cores of the CPUs

Conclusion

Conclusion

- Even Ericsson themselves say it: Don't use Erlang for speed, use it for the easiness of distributing jobs

Conclusion

- Even Ericsson themselves say it: Don't use Erlang for speed, use it for the easiness of distributing jobs
- I tried to implement the Erlang version very close to the Java version for comparison purposes. It can probably be implemented a lot more efficient if done "the Erlang way"

Conclusion

- Even Ericsson themselves say it: Don't use Erlang for speed, use it for the easiness of distributing jobs
- I tried to implement the Erlang version very close to the Java version for comparison purposes. It can probably be implemented a lot more efficient if done "the Erlang way"
- Solving the same problem was faster on my own quad-core machine than distributed on eight dual-core machines. Maybe there was too much network traffic overhead