An Exact Parallel Algorithm for Traveling Salesman Problem

V. Burkhovetskiy, B. Steinberg

Southern Federal University

October 21, 2017

Definitions

Hamiltonian Cycle

A Hamiltonian cycle is a graph cycle that visits each node exactly once.

Traveling Salesman Problem

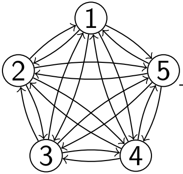
The traveling salesman problem is a problem of finding a minimum Hamiltonian cycle on a complete oriented graph with non-negative edge costs. It is NP-hard.

Balas' and Christofides' Algorithm

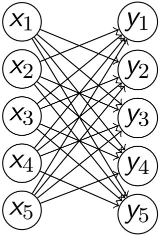
- Exact;
- Branch-and-bound;
- Each branch-and-bound tree node has up to $\frac{n}{2}$ branches (n – number of nodes in the graph);
- Branch-and-bound tree usually has small height;
- Works on any complete graph with non-negative edge costs.

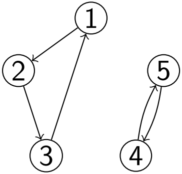
The Core Idea

- The Hungarian algorithm is used to solve the associated assignment problem;



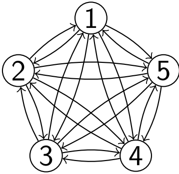
$$\begin{array}{c|c} & \begin{array}{ccccc} y_1 & y_2 & y_3 & y_4 & y_5 \end{array} \\ \begin{array}{c} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{array} & \begin{pmatrix} \infty & c_{12} & c_{13} & c_{14} & c_{15} \\ c_{21} & \infty & c_{23} & c_{24} & c_{25} \\ c_{31} & c_{32} & \infty & c_{34} & c_{35} \\ c_{41} & c_{42} & c_{43} & \infty & c_{45} \\ c_{51} & c_{52} & c_{53} & c_{54} & \infty \end{pmatrix} \end{array}$$



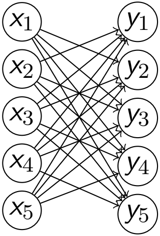


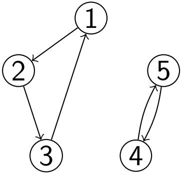
The Core Idea

- The Hungarian algorithm is used to solve the associated assignment problem;



$$
\begin{array}{c}
 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5
\end{array}
\begin{array}{c}
\begin{array}{ccccc} y_1 & y_2 & y_3 & y_4 & y_5 \end{array} \\
\begin{pmatrix}
\infty & c_{12} & c_{13} & c_{14} & c_{15} \\
c_{21} & \infty & c_{23} & c_{24} & c_{25} \\
c_{31} & c_{32} & \infty & c_{34} & c_{35} \\
c_{41} & c_{42} & c_{43} & \infty & c_{45} \\
c_{51} & c_{52} & c_{53} & c_{54} & \infty
\end{pmatrix}
\end{array}
$$

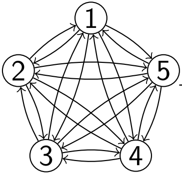




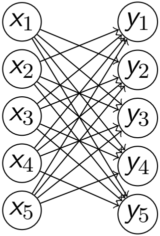
- Its solution forms either a Hamiltonian cycle (then the cycle is optimal on the current branch), or a union of disjoint simple subcycles on the initial graph, and said union covers all nodes of the graph;

The Core Idea

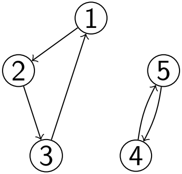
- The Hungarian algorithm is used to solve the associated assignment problem;



$$\longrightarrow \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{array} \!\!\begin{array}{c} \begin{array}{ccccc} y_1 & y_2 & y_3 & y_4 & y_5 \end{array} \\ \begin{pmatrix} \infty & c_{12} & c_{13} & c_{14} & c_{15} \\ c_{21} & \infty & c_{23} & c_{24} & c_{25} \\ c_{31} & c_{32} & \infty & c_{34} & c_{35} \\ c_{41} & c_{42} & c_{43} & \infty & c_{45} \\ c_{51} & c_{52} & c_{53} & c_{54} & \infty \end{pmatrix} \end{array} \longrightarrow$$



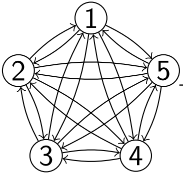
$\longrightarrow$



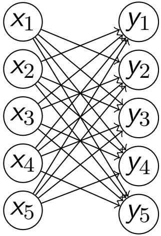
- Its solution forms either a Hamiltonian cycle (then the cycle is optimal on the current branch), or a union of disjoint simple subcycles on the initial graph, and said union covers all nodes of the graph;
- We merge the subcycles until we obtain a Hamiltonian cycle. The cycle is optimal on the current branch;

The Core Idea

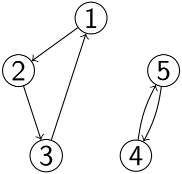
- The Hungarian algorithm is used to solve the associated assignment problem;



$$\longrightarrow \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{array} \begin{array}{c} \begin{array}{ccccc} y_1 & y_2 & y_3 & y_4 & y_5 \end{array} \\ \begin{pmatrix} \infty & c_{12} & c_{13} & c_{14} & c_{15} \\ c_{21} & \infty & c_{23} & c_{24} & c_{25} \\ c_{31} & c_{32} & \infty & c_{34} & c_{35} \\ c_{41} & c_{42} & c_{43} & \infty & c_{45} \\ c_{51} & c_{52} & c_{53} & c_{54} & \infty \end{pmatrix} \end{array} \longrightarrow$$



$\longrightarrow$



- Its solution forms either a Hamiltonian cycle (then the cycle is optimal on the current branch), or a union of disjoint simple subcycles on the initial graph, and said union covers all nodes of the graph;
- We merge the subcycles until we obtain a Hamiltonian cycle. The cycle is optimal on the current branch;
- Different branches consider different ways of merging the subcycles.

Our Modifications

- ✓ Graphs are represented as weighed adjacency lists as opposed to edge lists used by Balas and Christofides;

Our Modifications

- ✓ Graphs are represented as weighed adjacency lists as opposed to edge lists used by Balas and Christofides;
- ✓ The branch-and-bound search tree is traversed in a different order;

Our Modifications

- ✓ Graphs are represented as weighed adjacency lists as opposed to edge lists used by Balas and Christofides;
- ✓ The branch-and-bound search tree is traversed in a different order;
- ✓ We parallelized the algorithm with OpenMP;

Our Modifications

- ✓ Graphs are represented as weighed adjacency lists as opposed to edge lists used by Balas and Christofides;
- ✓ The branch-and-bound search tree is traversed in a different order;
- ✓ We parallelized the algorithm with OpenMP;

- ✗ Balas' and Christofides' bounding procedures and the second branching method are excluded.

Implementation Details and Test Environment

- Programming language: C++;
- Compiler: GCC v. 6.3.0, compiler option: -Ofast;
- GNU OpenMP v. 6.3.0;
- OS: Debian 9 (Linux);
- Processor: Intel® Core™ i5-6600 CPU @ 3.30GHz
  - 4 cores;
  - no hyperthreading;
  - L3: 6 MB (shared);
  - L2: 256 kB (split);
  - L1: 32 kB instruction cache, 32 kB data cache (split);
- RAM: DDR4, 32 GB, clock speed: 2133 MHz;
- Download link:
  http://ops.rsu.ru/download/progs/BalasChristofides_v1_0.zip

Results

(On graphs with uniform random integer edge costs in range from 0 to 1 000 000)

Number of nodes	Average Time, sec		Speedup
	Sequential	Parallel (4 Cores)	
1000	4.3838	3.7466	1.17
1500	12.5135	11.9873	1.04
2000	60.5096	32.5627	1.86
2500	67.0658	50.7420	1.32
3000	130.0924	75.8543	1.72

Results

(On graphs with uniform random integer edge costs in range from 0 to 1 000 000)

Number of nodes	Average Time, sec		Speedup
	Sequential	Parallel (4 Cores)	
1000	4.3838	3.7466	1.17
1500	12.5135	11.9873	1.04
2000	60.5096	32.5627	1.86
2500	67.0658	50.7420	1.32
3000	130.0924	75.8543	1.72

- During a sequential tree traversal the current best value of the cost function can (and probably will) improve several times, which helps to cut unnecessary branches closer to the root;

Results

(On graphs with uniform random integer edge costs in range from 0 to 1 000 000)

Number of nodes	Average Time, sec		Speedup
	Sequential	Parallel (4 Cores)	
1000	4.3838	3.7466	1.17
1500	12.5135	11.9873	1.04
2000	60.5096	32.5627	1.86
2500	67.0658	50.7420	1.32
3000	130.0924	75.8543	1.72

- During a sequential tree traversal the current best value of the cost function can (and probably will) improve several times, which helps to cut unnecessary branches closer to the root;
- Threads of a parallel program could explore the unneeded branches further than one thread of a sequential program would, because the current best value could not have been improved enough to eliminate them.

Different Tree Traversal Order (TTO)

Number of nodes	Average Time, sec (Parallel Algorithm, 4 Cores)	
	Balas' and Christofides' TTO[1]	Our TTO
1000	7.1217	3.7466
1500	29.0514	11.9873
2000	61.9874	32.5627
2500	137.3588	50.7420
3000	219.6173	75.8543

[1] Their TTO was used in our algorithm

Comparison to Other Algorithms

Number of nodes	Average Time, sec		
	Concorde	Fischetti-T.	Our Parallel
1000	3954.1[1]	4.6[1]	3.7

[1] Results from http://www.graphalgorithms.it/erice2008/Talks/ATSP_Lecture_Erice_Toth.pdf

Comparison to Other Algorithms

Number of nodes	Average Time, sec		
	Concorde	Fischetti-T.	Our Parallel
1000	3954.1[1]	4.6[1]	3.7

Fischetti-T. and Concorde were run on matrices with smaller edge cost range (from 0 to 1 000), which is not a good choice for such problem sizes.

Number of nodes	Average Time, sec (Parallel Algorithm, 4 Cores)	
	0...1 000 000	0...1 000
1000	3.7466	1.0018
1500	11.9873	1.0836
2000	32.5627	1.1875
2500	50.7420	2.1314
3000	75.8543	1.8445

[1] Results from http://www.graphalgorithms.it/erice2008/Talks/ATSP_Lecture_Erice_Toth.pdf

Further Improvements

- Decrease memory consumption;

Further Improvements

- Decrease memory consumption;
- Increase the parallel speedup;

Further Improvements

- Decrease memory consumption;
- Increase the parallel speedup;
- Experiment on sparse graphs.

Thank you!
Any questions?