8th DIMACS Implementation Challenge:
The Traveling Salesman Problem

http://www.research.att.com/~dsj/chtsp/

David S. Johnson
AT&T Labs – Research
Florham Park, NJ 07932-0971

dsj@research.att.com
http://www.research.att.com/~dsj/

Co-Organized with
Lyle McGeoch, Fred Glover, Cesar Rego
DIMACS Implementation Challenges

1. Network Flows and Matching, 1990-91

2. Clique, Coloring, and Satisfiability, 1992-93

3. Parallel Computing on Trees and Graphs, 1993-94

4. Fragment Assembly and Genome Rearrangement, 1994-95

5. Priority Queues and Dictionaries, 1995-96


8. The Traveling Salesman Problem, 2000...
OUTLINE OF TALK

● Why a Challenge

● Who should Participate

● How to Participate

● Preliminary Results
  – Machine Speeds and Normalizations
  – Algorithmic Comparisons

● Future Directions
SCIENTIFIC GOALS

- Determine the current state of the art with respect to tradeoffs between running time and quality of solution for the TSP.

- Identify promising algorithmic ideas for the TSP worthy of further investigation.

- Gain insight into combinatorial optimization in general by seeing how various generic ideas are best adapted to the TSP context.

- Explore how best to conduct a distributed algorithmic comparison project of this sort, and how best to analyze and display the resulting data.

- Produce a DIMACS technical report summarizing what we learn, with all participants as co-authors.
OTHER AGENDAS

• Obtain source material for a summary chapter on experimental analysis of TSP algorithms to be written with Lyle McGeoch for an upcoming book on the TSP edited by Gutin and Punnen.

• Establish a long-lived mechanism for future researchers to evaluate their algorithms in comparison to works of the past.

• Stop the flow of uninformed papers on the TSP.
DESIRED PARTICIPANTS

• Current TSP researchers.

• Researchers who have published experimental results about TSP algorithms in the past, so that those results can be put in perspective.

• New TSP researchers interested in investigating new ideas and unanswered questions.

• Future TSP researchers who want to compare with previous results.
ARENAS FOR COMPETITION

(Currently Restricted to Symmetric TSP)

1. Heuristics
   - Tour Construction Heuristics
   - Simple Local Optimization
     (2-Opt, 3-Opt, and Variants)
   - Lin-Kernighan Variants
   - Chained Lin-Kernighan Variants
   - Other Metaheuristics
     (Simulated Annealing, Tabu Search, Neural Nets, Genetic Algorithms, etc.)

2. Fast Lower Bound Algorithms

3. Optimization Algorithms

4. Open to Suggestions...
HOW TO PARTICIPATE

1. Download Instances, Instance Generators, and Benchmark Codes from the website.

2. Compile Generators and Benchmark Codes (C code) using your standard compilers on your standard machine.

3. Run the Generators to generate the random instances in the testbed, comparing to downloaded samples to verify that Generators are performing correctly.

4. Run the Benchmark Greedy code on selected random instances (as specified on the website) to (roughly) benchmark your machine’s speed as a function of instance size. Do this for all the specified instance that will fit in your machine’s memory.

5. Run your own codes on the all the Benchmark Instances that they can handle. Allowed excuses for failure to run: Instance too big, Running time too long, Code can’t handle instances of this type (distance matrices, fractional coordinates, etc.)

6. Send results to dsj@research.att.com using formats specified at the website. (Tentative initial deadline: 30 September 2000.)

7. Extra Credit: Perform extra experiments as suggested by DSJ or other participants. Suggest extra experiments to be performed by DSJ or other participants.
TESTBED, Part I - 55 Random Instances

1. **Uniform Random Euclidean Instances**
   (Points uniform in the $10^6 \times 10^6$ square)
   Sizes increasing by factors of $\sqrt{10}$ from 1,000 to 10,000,000
   - Ten 1,000-city instances
   - Five 3,162-city instances
   - Three 10,000-city instances
   - Two 31,623-city instances
   - Two 100,000-city instances
   - One each of $10^{5.5}$-, $10^6$-, $10^{6.5}$-, and $10^7$-city instances.

2. **Uniform Random Euclidean Instances**
   (Points clustered in the $10^6 \times 10^6$ square)
   - Ten 1,000-city instances
   - Five 3,162-city instances
   - Three 10,000-city instances
   - Two 31,623-city instances
   - Two 100,000-city instances

3. **Random Distance Matrices**
   (Distances chosen uniformly from $[0, 10^6]$)
   - Four 1,000-city instances
   - Two 3,162-city instances
   - One 10,000-city instance
TESTBED, Part II - 34 TSPLIB Instances

dsj1000   d2103
pr1002    u2152
si1032    u2319
u1060     pr2392
vm1084    pcb3038
pcb1173   fl3795
d1291     fnl4461
rl1304    rl5915
rl1323    rl5934
nrw1379   pla7397
fl1400    rl11849
u1432     usa13509
fl1577    brd14051
d1655     d15112
vm1748    d18512
u1817     pla33810
rl1889    pla85900
1,000-City Clustered Random Euclidean Instance
3,162-City Clustered Random Euclidean Instance
10,000-City Clustered Random Euclidean Instance
100,000-City Uniform Random Euclidean Instance
(From Johnson, Bentley, McGeoch, & Rothberg, 1993)
## The Test Battery

- time greedy E1k.0 1000
- time greedy E3k.0 316
- time greedy E10k.0 100
- time greedy E31k.0 32
- time greedy E100k.0 10
- time greedy E316k.0 3
- time greedy E1M.0 1
- time greedy E3M.0 1
- time greedy E10M.0 1

## User Seconds

<table>
<thead>
<tr>
<th>Instance</th>
<th>500 Mhz Alpha</th>
<th>400 Mhz MIPS R12000</th>
<th>300 Mhz MIPS R12000</th>
<th>500 Mhz Pentium III</th>
<th>440 Mhz Sparc Ultra 10</th>
<th>196 Mhz MIPS R10000</th>
<th>135 Mhz IBM Power2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000 x 1000</td>
<td>12</td>
<td>14</td>
<td>18</td>
<td>22</td>
<td>23</td>
<td>29</td>
<td>51</td>
</tr>
<tr>
<td>316 x 3162</td>
<td>14</td>
<td>15</td>
<td>20</td>
<td>24</td>
<td>27</td>
<td>31</td>
<td>55</td>
</tr>
<tr>
<td>100 x 10,000</td>
<td>16</td>
<td>17</td>
<td>23</td>
<td>28</td>
<td>34</td>
<td>37</td>
<td>74</td>
</tr>
<tr>
<td>32 x 31,623</td>
<td>23</td>
<td>22</td>
<td>29</td>
<td>40</td>
<td>52</td>
<td>71</td>
<td>128</td>
</tr>
<tr>
<td>10 x 100,000</td>
<td>36</td>
<td>41</td>
<td>59</td>
<td>47</td>
<td>73</td>
<td>160</td>
<td>143</td>
</tr>
<tr>
<td>3 x 316,228</td>
<td>55</td>
<td>73</td>
<td>92</td>
<td>52</td>
<td>89</td>
<td>380</td>
<td>184</td>
</tr>
<tr>
<td>1 x 1,000,000</td>
<td>88</td>
<td>124</td>
<td>156</td>
<td>68</td>
<td>120</td>
<td>480</td>
<td>259</td>
</tr>
<tr>
<td>1 x 3,162,278</td>
<td>330</td>
<td>711</td>
<td>790</td>
<td>235</td>
<td>430</td>
<td>1690</td>
<td>911</td>
</tr>
<tr>
<td>1 x 10,000,000</td>
<td>1330</td>
<td>3670</td>
<td>3600</td>
<td>--</td>
<td>--</td>
<td>6100</td>
<td>--</td>
</tr>
</tbody>
</table>
Normalization: 196 Mhz MIPS to 500 Mhz Alpha

Correction Factor = \frac{\text{Benchmark Greedy time for Alpha}}{\text{Benchmark Greedy time for MIPS}}
Errors in Running Time Normalization: Greedy Algorithm

![Diagram showing overestimate and underestimate](image-url)
Errors in Running Time Normalization: Lin-Kernighan

- Overestimate
- Underestimate

NUMBER OF CITIES

MIPS NORMALIZED TIME/ALPHA ACTUAL TIME
Greedy versus Clarke-Wright (Alpha vs MIPS)

NUMBER OF CITIES

RATIO OF NORMALIZED RUNNING TIMES

CW Better

GR Better
Greedy versus Clarke-Wright (Same Machine)

RATIO OF NORMALIZED RUNNING TIMES

NUMBER OF CITIES
Greedy versus Clarke-Wright

Number of cities versus percent difference in tour lengths. The figure shows the comparison between Greedy (GR) and Clarke-Wright (CW) algorithms in terms of tour length difference. The y-axis represents the percent difference in tour lengths, while the x-axis represents the number of cities. The plot indicates that CW is better for a higher number of cities, whereas GR is better for a smaller number of cities.
Chained LK: Johnson-McGeoch vs Applegate-Cook-Rohe

PERCENT DIFFERENCE IN TOUR LENGTHS

A-C-R Better

J-M Better

NUMBER OF CITIES

1000 5000 10000 50000 100000

-2.0 -1.5 -1.0 -0.5 0.0 0.5
<table>
<thead>
<tr>
<th>Excess over HK Bound</th>
<th>Excess over Optimal</th>
<th>Normalized Running Time</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>17.25</td>
<td>16.48</td>
<td>0.28</td>
<td>greedy</td>
</tr>
<tr>
<td>4.96</td>
<td>4.27</td>
<td>87.41</td>
<td>2opt</td>
</tr>
<tr>
<td>3.32</td>
<td>2.64</td>
<td>1.23</td>
<td>acrlk</td>
</tr>
<tr>
<td>3.25</td>
<td>2.57</td>
<td>1.31</td>
<td>acrlk10</td>
</tr>
<tr>
<td>3.03</td>
<td>2.35</td>
<td>87.53</td>
<td>3opt</td>
</tr>
<tr>
<td>2.42</td>
<td>1.74</td>
<td>1.84</td>
<td>acrlk100</td>
</tr>
<tr>
<td>2.18</td>
<td>1.51</td>
<td>101.76</td>
<td>lk</td>
</tr>
<tr>
<td>1.37</td>
<td>0.71</td>
<td>6.69</td>
<td>acrlk1000</td>
</tr>
<tr>
<td>1.30</td>
<td>0.63</td>
<td>155.98</td>
<td>ilk.1N</td>
</tr>
<tr>
<td>1.04</td>
<td>0.38</td>
<td>276.02</td>
<td>ilk.3N</td>
</tr>
<tr>
<td>0.91</td>
<td>0.25</td>
<td>63.83</td>
<td>acrlkN</td>
</tr>
<tr>
<td>0.86</td>
<td>0.19</td>
<td>609.67</td>
<td>ilkN</td>
</tr>
<tr>
<td>0.00</td>
<td>-0.66</td>
<td>118.86</td>
<td>heldkarp</td>
</tr>
<tr>
<td>-0.12</td>
<td>-0.78</td>
<td>106.93</td>
<td>rhk1</td>
</tr>
<tr>
<td>-0.30</td>
<td>-0.95</td>
<td>26.15</td>
<td>rhk2</td>
</tr>
<tr>
<td>-0.35</td>
<td>-1.01</td>
<td>13.32</td>
<td>rhk3</td>
</tr>
</tbody>
</table>
### 3,162-City Random Distance Matrix

<table>
<thead>
<tr>
<th>Excess over HK Bound</th>
<th>Excess over Optimal</th>
<th>Normalized Running Time</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>219.05</td>
<td>219.04</td>
<td>16.70</td>
<td>greedy</td>
</tr>
<tr>
<td>89.17</td>
<td>89.17</td>
<td>19.48</td>
<td>2opt</td>
</tr>
<tr>
<td>46.73</td>
<td>46.72</td>
<td>18.90</td>
<td>3opt</td>
</tr>
<tr>
<td>7.43</td>
<td>7.43</td>
<td>3.58</td>
<td>acrlk</td>
</tr>
<tr>
<td>6.08</td>
<td>6.08</td>
<td>4.77</td>
<td>acrclk10</td>
</tr>
<tr>
<td>4.66</td>
<td>4.66</td>
<td>12.56</td>
<td>acrclk100</td>
</tr>
<tr>
<td>4.18</td>
<td>4.18</td>
<td>20.19</td>
<td>lk</td>
</tr>
<tr>
<td>3.75</td>
<td>3.75</td>
<td>62.64</td>
<td>acrclk1000</td>
</tr>
<tr>
<td>2.72</td>
<td>2.72</td>
<td>31.51</td>
<td>ilk.1N</td>
</tr>
<tr>
<td>2.50</td>
<td>2.50</td>
<td>57.55</td>
<td>ilk.3N</td>
</tr>
<tr>
<td>2.31</td>
<td>2.31</td>
<td>121.50</td>
<td>ilkN</td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>612.49</td>
<td>concorde</td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>40.36</td>
<td>heldkarp</td>
</tr>
</tbody>
</table>
CONCLUSIONS

Yet to be derived...

Your Help Needed!