Tuning a TSP Algorithm

Jon Bentley
Bell Labs Research, Retired

Outline
Review of Recursive Generation
The Traveling Salesperson Problem
A Sequence of TSP Algorithms
Principles of Algorithm Engineering
Recursive Generation

A technique for systematically generating all members of a class

Example: all subsets of the n integers 0, 1, 2, ..., n-1
  Representation? {1, 3, 4} or 01011 or ...

An Iterative Solution: binary counting
  00000 00001 00010 00011 00100 ... 11111

A Recursive Solution to Fill p

void allsubsets(int m)
{
  if (m == 0) {
    visit();
  } else {
    Sm-1 = 0;
    allsubsets(m-1);
    p[m-1] = 1;
    allsubsets(m-1);
  }
}
The TSP

The Problem

- Mr. Lincoln’s Eighth Circuit
- Scheduling vehicles, drills, plotters
- Automobile assembly lines

A Prototypical Problem

- NP-Hard
- Held-Karp dynamic programming
- Approximation algorithms
- Kernighan-Lin heuristics
A Personal History

Shamos’s 1978 Thesis

“In fact, the tour was obtained by applying the Christofides heuristic several times and selecting the best result. It is not known to be a shortest tour for the given set of points.”

A 1997 Exercise

Can simple code now solve a 16-city problem on faster machines?

A 2016 Talk

What has changed in two more decades?
This Talk

Alternative Titles
   A case study in ...
       ... implementing algorithms
       ... recursive enumeration
       ... algorithm engineering
       ... applying algorithms and data structures
   A master class in algorithms
   A sampler of performance engineering
   Two fun days of programming

Non-Titles
   State-of-the-art TSP algorithms
   Experimental analysis of algorithms
Representation Details

Count of Cities
#define MAXN 20
int n;

Permutation of Cities
int p[MAXN];

Distances Between Cities
Typedef double Dist;
Dist d(int i, int j)
Algorithm 1

The Idea
Recursively generate all n! permutations and choose the best

Implementation
void search1(int m)
{
    int i;
    if (m == 1)
        check1();
    else
        for (i = 0; i < m; i++) {
            swap(i, m-1);
            search1(m-1);
            swap(i, m-1);
        }
}
void check1()
{
    int i;
    Dist sum = dist1(p[0], p[n-1]);
    for (i = 1; i < n; i++)
        sum += dist1(p[i-1], p[i]);
    save(sum);
}

void save(Dist sum)
{
    int i;
    if (sum < minsum) {
        minsum = sum;
        for (i = 0; i < n; i++)
            minp[i] = p[i];
    }
}

void solve1()
{
    search1(n);
}
Run Time of Algorithm 1

Analysis

Permutations: n!
Distance calculations at each: n
Total distance calculations: n \times n!

Experiments: Intel Core i7-6500U @ 2.50GHz (Default machine)

<table>
<thead>
<tr>
<th>N</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>0.05 secs</td>
</tr>
<tr>
<td>9</td>
<td>0.34 secs</td>
</tr>
<tr>
<td>10</td>
<td>3.97 secs</td>
</tr>
<tr>
<td>11</td>
<td>45.47 secs</td>
</tr>
<tr>
<td>12</td>
<td>9 minutes</td>
</tr>
<tr>
<td>13</td>
<td>2 hours</td>
</tr>
</tbody>
</table>
Constant Factor Improvements

External to the Program
- Compiler optimizations
- Faster hardware

Internal Changes
- Modify the C code
Compiler Optimizations

An Experiment

<table>
<thead>
<tr>
<th>N</th>
<th>No Opt</th>
<th>-O3</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>0.34 secs</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>3.97 secs</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>45.47 secs</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>9 min</td>
<td>19.81 secs</td>
</tr>
<tr>
<td>13</td>
<td>2 hours</td>
<td>4.3 min</td>
</tr>
</tbody>
</table>

gcc -O3

Factor of ~25 on this machine
Factor of ~6 on a Raspberry Pi 3
Only full optimization shown from now on
Faster Hardware

Two Machines

1997: Pentium Pro @ 200MHz
2016: Intel Core i7-6500U @ 2.50GHz

<table>
<thead>
<tr>
<th>N</th>
<th>1997</th>
<th>2016</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>20.9 secs</td>
<td>0.12 secs</td>
</tr>
<tr>
<td>11</td>
<td>4 min</td>
<td>1.62 secs</td>
</tr>
<tr>
<td>12</td>
<td>48 min</td>
<td>19.81 secs</td>
</tr>
<tr>
<td>13</td>
<td>10 hrs</td>
<td>4.3 min</td>
</tr>
</tbody>
</table>

Speedup of about 150
x12 due to clock speed; x12 due to wider data and deeper pipes
Constant Factor Improvements: Internal

Faster computation
- Change doubles to floats or (scaled) ints
- Measure and use fastest size (short, int, long)

Avoid recomputing math-intensive function

Algorithm 1
```c
Dist geomdist(int i, int j) {
    return (Dist) (sqrt(sqr(c[i].x-c[j].x) +
                    sqr(c[i].y-c[j].y)));
}
```

Algorithm 2
- Precompute all $n^2$ distances in a table
  ```c
  #define dist2(i, j) distarr[i][j]
  ```

Speedup of about 2.5 or 3
Algorithm 3

Idea
Reduce distance calculations by examining fewer permutations
Fix last city in the permutation

Code
void solve3()
{
    search2(n-1);
}

Analysis
Permutations: (n-1)!
Distance calculations at each: n
Total distance calculations: n x (n-1)! = n!
Algorithm 4

Don’t recompute sum; carry along a partial sum instead

```c
void solve4()
{
    search4(n-1, ZERO);
}

void search4(int m, Dist sum)
{
    int i;
    if (m == 1)
        check4(sum + dist2(p[0], p[1]));
    else
        for (i = 0; i < m; i++) {
            swap(i, m-1);
            search4(m-1, sum + dist2(p[m-1], p[m]));
            swap(i, m-1);
        }
}

void check4(Dist sum)
{
    sum += dist2(p[0], p[n-1]);
    save(sum);
}
```

Reduces \( n \times (n-1)! \) to \( \sim (1+e) \times (n-1)! \)
**Summary of Four Algorithms**

<table>
<thead>
<tr>
<th></th>
<th>Alg 1</th>
<th>Alg 2</th>
<th>Alg 3</th>
<th>Alg 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>n x n!</td>
<td>n x n!</td>
<td>n x (n-1)!</td>
<td>(1+e) x (n-1)!</td>
</tr>
<tr>
<td>10</td>
<td>0.83</td>
<td>0.25</td>
<td>0.02</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>10.28</td>
<td>3.22</td>
<td>0.28</td>
<td>0.25</td>
</tr>
<tr>
<td>12</td>
<td>120</td>
<td>33.50</td>
<td>3.47</td>
<td>2.59</td>
</tr>
<tr>
<td>13</td>
<td>1560</td>
<td>430</td>
<td>34.84</td>
<td>21.02</td>
</tr>
</tbody>
</table>

Seconds on an AMD E-450 at 1.65GHz

All run times are for initial members of one sequence uniform on the unit square
Perspective on Factorial Growth

Each factor of \( n \) allows us to increase the problem size by 1 in about the same wall-clock time.

Fast machines, great compilers and code tuning allow us to solve problems into the teens:

- 14 cities in 5 minutes
  - Lincoln's: 447.4
  - Opt: 399.1
- 16 cities in 16 hours
  - Thesis: 237.11
  - Opt: 236.08

But can we ever analyze, say, all permutations of a deck of cards?
Exponential Growth

Definition of “Growing Exponentially”

Popular usage: “growing real fast, looks like”
Mathematics: $c^n$ for some base $c$ and time period for $n$

Factorial Growth

By Stirling’s approximation,

\[
\ln n! = n \ln n - n + O(\ln n)
\]
\[
\lg n! \sim n \lg n - 1.386 n
\]

So $n! \sim 2^n (n \lg n - 1.386 n) \sim 2^n (n \lg n) \sim (n/e)^n$

Factorial grows faster than any exponential function

How Big is 52!? 

$52! \sim 8.0658 \times 10^{67} \sim 2^{225}$
52! in Everyday Terms

Set a timer to count down $52! = 8.0658 \times 10^{67}$ nanoseconds.
Stand on the equator, and take one step forward every million years.
When you've circled the earth once, take a drop of water from the Pacific Ocean, and keep going.
When the Pacific Ocean is empty, lay a sheet of paper down, refill the ocean and carry on.
When your stack of paper reaches the moon, check the timer. You’re about done.

Fact: the universe is about $26! = 4.03 \times 10^{26}$ nanoseconds old
Moral: we’re going to have to ignore some of the possible tours.
Puzzle Break

From Greg Conti

Find all permutations of 1..9 such that each initial substring of length m is divisible by m
For 1..3, 321 works but not 132

Two Main Approaches

Thinking

Computing

What structures? What language?

How to generate all n! permutations of a string?

Recursive search(left, right)
Start with left = “123456789”, right = “”; end when left == “”
search(“356”, “421978”) calls
search(“56”, “3421978”), search(“36”, “5421978”), search(“35”, “6421978”)
An Awk Program

function search(left, right, i) {
    if (left == "") {
        for (i = 1; i <= length(right); i++)
            if (substr(right, 1, i) % i)
                return
        print "   " right
    } else
    for (i = 1; i <= length(left); i++)
        search(substr(left, 1, i-1) substr(left, i+1), substr(left, i, 1) right)
}

BEGIN { search("123456789", ")

About 3 seconds to find 381654729

BEGIN { search("123456789", "") }
How To Make it Faster?

Constant Factors
  Don’t check for divisibility by 1; change language; ...

Pruning the Search – Lessons from 381654729?
  Even digits in even positions
  Odd digits in odd positions
  Digit 5 in position 5
  Old: 9 x 8 x 7 x 6 x 5 x 4 x 3 x 2 x 1 = 9! = 362880
  New: 4 x 4 x 3 x 3 x 1 x 2 x 2 x 1 x 1 = (4!)² = 576

Code
  digit = substr(left, i, 1)
  if (((length(left) % 2) == (digit % 2)) &&
      ((length(left) == 5) == (digit == 5)))
    search2(substr(left, 1, i-1) substr(left, i+1), digit right)
Lessons from the Break

Factorial grows very quickly
  We can never visit the entire search space
The key to speed is pruning the search
Some fancy algorithms can be implemented in very little code
Algorithm 5

Don’t keep doing what doesn’t work; sum never decreases

Code

```c
void search5(int m, Dist sum)
{
    int i;
    if (sum > minsum)
        return;
    if (m == 1) {
        ...
    }
}
```

Experiments on an Intel Core i7-3630QM @ 2.40GHz

<table>
<thead>
<tr>
<th></th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alg 4</td>
<td>0.59</td>
<td>5.14</td>
<td>54.44</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alg 5</td>
<td>0.01</td>
<td>0.12</td>
<td>0.41</td>
<td>0.99</td>
<td>3.92</td>
<td>39.00</td>
</tr>
</tbody>
</table>
Algorithm 6

A Better Lower Bound: Add MST of remaining points

Code

```c
void search6(int m, Dist sum, Mask mask)
{
    int i;
    if (sum + mstdist(mask | bit[p[m]]) > minsum)
        return;
    ...
    search6(m-1,
        sum + dist2(p[m-1], p[m]),
        mask & ~bit[p[m-1]]);
}
```
Prim-Dijkstra MST Code

```
Dist mstdist(Mask mask)
{
    int i, m, mini, newcity, n;
    Dist mindist, thisdist, totaldist;
    totaldist = ZERO;
    n = 0;
    for (i = 0; i < MAXN; i++)
        if (mask & bit[i])
            q[n++].city = i;
    newcity = q[n-1].city;
    for (i = 0; i < n-1; i++)
        q[i].nndist = INF;
    for (m = n-1; m > 0; m--)
        { mindist = INF;
            for (i = 0; i < m; i++)
                { thisdist = dist2(q[i].city, newcity);
                    if (thisdist < q[i].nndist) {
                        q[i].nndist = thisdist;
                        q[i].nnfrag = newcity;
                    }
                    if (q[i].nndist < mindist) {
                        mindist = q[i].nndist;
                        mini = i;
                    }
                }
            newcity = q[mini].city;
            totaldist += mindist;
            q[mini] = q[m-1];
        }
    return totaldist;
}
```
More Experiments

Algorithms 5 and 6

<table>
<thead>
<tr>
<th>N</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>23</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alg 5</td>
<td>0.95</td>
<td>4.03</td>
<td>40.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alg 6</td>
<td>0.00</td>
<td>0.02</td>
<td>0.03</td>
<td>0.05</td>
<td>0.06</td>
<td>0.22</td>
<td>0.20</td>
<td>2.39</td>
<td></td>
</tr>
</tbody>
</table>

Algorithm 6

<table>
<thead>
<tr>
<th>N</th>
<th>23</th>
<th>24</th>
<th>25</th>
<th>26</th>
<th>27</th>
<th>28</th>
<th>29</th>
<th>30</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alg 6</td>
<td>2.39</td>
<td>0.75</td>
<td>1.81</td>
<td>5.56</td>
<td>8.97</td>
<td>13.36</td>
<td>7.67</td>
<td>13.73</td>
</tr>
</tbody>
</table>
Algorithms 7 and 8

Cache MST distances rather than recomputing them

Algorithm 7: Store all (used) distances in a table of size $2^n$

$$nmask = mask \mid bit[p[m]];$$
$$\text{if } (mstdistarr[nmask] < 0.0)$$
$$\quad mstdistarr[nmask] = mstdist(nmask);$$
$$\text{if } (sum + mstdistarr[nmask] > \text{minsum})$$
$$\quad \text{return;}$$

Algorithm 8: Store them in a hash table

$$\text{if } (sum + mstdistlookup(mask \mid bit[p[m]]) > \text{minsum})$$
$$\quad \text{return;}$$
Hash Table Implementation

Dist mstdistlookup(Mask mask)
{
    Tptr p;
    int h;
    h = mask % MAXBIN;
    for (p = bin[h]; p != NULL; p = p->next)
        if (p->arg == mask)
            return p->val;
    p = (Tptr) malloc(sizeof(Tnode));
    p->arg = mask;
    p->val = mstdist(mask);
    p->next = bin[h];
    bin[h] = p;
    return p->val;
}
## Experiments

### Algorithms 6 and 8

<table>
<thead>
<tr>
<th>N</th>
<th>30</th>
<th>31</th>
<th>32</th>
<th>33</th>
<th>34</th>
<th>35</th>
<th>36</th>
<th>37</th>
<th>38</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alg 6</td>
<td>13.44</td>
<td>12.69</td>
<td>40.39</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alg 8</td>
<td>0.50</td>
<td>0.45</td>
<td>1.41</td>
<td>1.88</td>
<td>5.33</td>
<td>2.31</td>
<td>2.44</td>
<td>48.52</td>
<td>7.59</td>
</tr>
</tbody>
</table>
Algorithm 9

Sort edges to visit nearest city first, then others in order

```
for (i = 0; i < m; i++)
    unvis[i] = i;
for (top = m; top > 0; top--)
    {  
        mindist = INF;
        for (j = 0; j < top; j++)
            { 
                thisdist = dist2(p[unvis[j]], p[m]);
                if (thisdist < mindist)
                    { 
                        mindist = thisdist;
                        minj = j;
                    }
            }
        swap(unvis[minj], m-1);
        search9(m-1,
            sum + dist2(p[m-1], p[m]),
            mask & ~bit[p[m-1]]);
        swap(unvis[minj], m-1);
        unvis[minj] = unvis[top-1];
    }
```
Experiments

Algorithms 8 and 9

<table>
<thead>
<tr>
<th></th>
<th>38</th>
<th>39</th>
<th>40</th>
<th>41</th>
<th>42</th>
<th>43</th>
<th>44</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alg 8</td>
<td>8.27</td>
<td>11.08</td>
<td>25.84</td>
<td>93.61</td>
<td>55.20</td>
<td>41.95</td>
<td></td>
</tr>
<tr>
<td>Alg 9</td>
<td>4.41</td>
<td>0.86</td>
<td>3.44</td>
<td>10.80</td>
<td>12.69</td>
<td>10.39</td>
<td>21.19</td>
</tr>
</tbody>
</table>

In 1997, stopped at n=30 and 305.66 seconds
A 45-City Tour

42 seconds on a 2.50GHz Core i7
How Far Can Algorithm 9 Go?

<table>
<thead>
<tr>
<th>N</th>
<th>Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>3.59</td>
</tr>
<tr>
<td>41</td>
<td>11.33</td>
</tr>
<tr>
<td>42</td>
<td>12.17</td>
</tr>
<tr>
<td>43</td>
<td>11.62</td>
</tr>
<tr>
<td>44</td>
<td>23.49</td>
</tr>
<tr>
<td>45</td>
<td>41.89</td>
</tr>
<tr>
<td>46</td>
<td>183.70</td>
</tr>
<tr>
<td>47</td>
<td>1036.28</td>
</tr>
<tr>
<td>48</td>
<td>1199.11</td>
</tr>
<tr>
<td>49</td>
<td>7815.83</td>
</tr>
<tr>
<td>50</td>
<td>2172.03</td>
</tr>
<tr>
<td>51</td>
<td>6537.12</td>
</tr>
<tr>
<td>52</td>
<td>11254.17</td>
</tr>
</tbody>
</table>

My Thanksgiving 2016 Cyclefest

When to think and when to run programs?

= 3hrs 7 min
Complete Code: 160 Lines of C

void solve9()
{
    int i, j;
    Mask mask;
    for (i = 0; i < n; i++)
        distaar[i] = 1e35;
    for (i = 0; i < MAXBIN; i++)
        q[i] = MAXBIN;
    mask = 0; // mask is a bit vector
    minsum = 1e35;
    search9(n-1, 0.0, mask);
}

void main()
{
    int i;
    FILE *fp;
    fp = fopen("rand60.txt", "r");
    while (fscanf(fp, "%f %f", &c[i].x, &c[i].y) != EOF)
        n = 25;
    solve9();
}

/* tspfastonly.c -- Cut tsp.c down to only final algorithm 9 */

#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>

#define MASKTYPE long
#define MAXN 60
#define MAXBIN 9999991
typedef double Dist;
int n = 0;
typedef struct point {
    float x;
    float y;
} Point;
Point c[MAXN];
int p[MAXN], minp[MAXN];
Dist minsum;
Dist distarr[MAXN][MAXN];

float sqr(float x) { return x*x; }

Dist geomdist(int i, int j)
{
    return (Dist) (sqrt(sqr(c[i].x - c[j].x) +
                    sqr(c[i].y - c[j].y)));
}

#define dist2(i, j) (distarr[i][j])

// Search algo
void swap(int i, int j)
{
    int t = p[i]; p[i] = p[j]; p[j] = t;
}

void check4(Dist sum)
{
    Dist totaldist;
    int i;
    if (sum < minsum) {
        minsum = sum;
        for (i = 0; i < n; i++)
            minp[i] = p[i];
    }
}

typedef MASKTYPE Mask;

Mask Bit[MAXN];

struct link {
    int city;
    int nnfrag;
    Dist nndist;
} q[MAXN];

Dist mstdist(Mask mask)
{
    int i, m, mini, newcity, n;
    Dist mindist, thisdist, totaldist;
    n = 0;
    for (i = 0; i < MAXN; i++)
        if (mask & bit[i])
            q[n].city = i;
            mindist = thisdist;
            newcity = q[n].city;
            for (i = 0; i < n-1; i++)
                if (mindist < dist2(p[i], p[i+1]))
                    mindist = dist2(p[i], p[i+1]);
            totaldist += mindist;
            n++;
    return totaldist;
}

Dist mstdistlookup(Mask mask)
{
    Tptr p;
    Hash int h;
    h = mask % MAXBIN;
    for (p = bin[h]; p != NULL; p = p->next)
        if (p->arg == mask)
            return p->val;
    p = (Tptr) malloc(sizeof(Tnode));
    p->arg = mask;
    p->val = mstdist(mask);
    p->next = bin[h];
    bin[h] = p;
    return p->val;
}

void search9(int m, Dist sum, Mask mask)
{
    int i;
    if (sum + mstdistlookup(mask | bit[p[m]]) > minsum)
        return;
    if (m == 1) {
        check4(sum + dist2(p[0], p[1]));
    } else {
        int j, minj, unvis[MAXN], top;
        Dist mindist, thisdist, totaldist;
        for (i = 0; i < m; i++)
            unvis[i] = i;
        for (top = m-1; top > 0; top--)
            mindist = thisdist;
            thisdist = 1e35; // worst case
            for (i = 0; i < top; i++)
                if (thisdist < dist2(p[i], p[i+1]))
                    thisdist = dist2(p[i], p[i+1]);
            if (thisdist < mindist)
                mindist = thisdist;
                minj = i;
        if (minj < m-1)
            swap(minj, m-1);
        for (i = unvis[top-1]; i < n; i++)
            if (thisdist < dist2(p[i], q[minj].city, newcity))
                i = unvis[top-1];
            totaldist += dist2(p[i], q[minj].city, newcity);
            minj = i;
        if (mindist < dist2(p[m-1], p[m], mask & bit[p[m-1]]))
            minj = i;
        if (minj < m-1)
            swap(minj, m-1);
        totaldist += dist2(p[m-1], p[m], mask & bit[p[m-1]]);
        minsum = totaldist;
    }
}

void solve9()
{
    int i, j;
    Mask mask;
    for (i = 0; i < n; i++)
        distaar[i] = 1e35;
    for (i = 0; i < MAXBIN; i++)
        q[i] = MAXBIN;
    mask = 0; // mask is a bit vector
    minsum = 1e35;
    search9(n-1, 0.0, mask);
}

void main()
{
    int i;
    FILE *fp;
    fp = fopen("rand60.txt", "r");
    while (fscanf(fp, "%f %f", &c[i].x, &c[i].y) != EOF)
        n = 25;
    solve9();
    for (i = 0; i < n; i++)
        printf("%3d\t%10.8f\t%10.8f \n", minp[i], c[minp[i]].x, c[minp[i]].y);
}

hash_sort_intersection

## Each Algorithm in under a Minute

1. Simple  
2. Store precomputed distances  
3. Fix starting city  
4. Accumulate distance  
5. Prune search  
6. Add MST of remaining cities  
7. Store MST distances  
8. Store MST distances in hash table  
9. Visit cities sorted by distance

<table>
<thead>
<tr>
<th>Step</th>
<th>N</th>
<th>Time (secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>13</td>
<td>35</td>
</tr>
<tr>
<td>4</td>
<td>13</td>
<td>21</td>
</tr>
<tr>
<td>5</td>
<td>17</td>
<td>40</td>
</tr>
<tr>
<td>6</td>
<td>32</td>
<td>40</td>
</tr>
<tr>
<td>7</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>8</td>
<td>40</td>
<td>26</td>
</tr>
<tr>
<td>9</td>
<td>45</td>
<td>42</td>
</tr>
</tbody>
</table>
Possible Additional Improvements

Constant Factor Speedups
- Faster machines
- Code tuning as before
- Better hashing: larger table, remove `malloc`

Better Pruning
- Better starting tour
- Better bounds: MST Length + Nearest Neighbor to each end
- Earlier pruning tests

Better Sorting
- Tune insertion sort; better algorithms
- Precompute all sorts
  - Sort once for each city
  - Select subsequence using mask

Bentley: TSP
Components

Incremental Software Development
   Total of 580 lines of C

Algorithmic Techniques
   Recursive permutation generation
   Store precomputed results: distances, MST lengths
   Partial sums
   Early cutoffs

Algorithms and Data Structures
   Vectors, strings
   Sets: Arrays and bit vectors
   Minimum Spanning Trees (MSTs)
   Hash tables
   Insertion sort
Code Tuning Rules

Space-for-Time 2: Store precomputed results
   Interpoint distances in a matrix; table of MST lengths

Space-for-Time 4: Lazy evaluation
   Compute all $n^2$ distances but only compute MSTs as needed

Logic Rule 2: Short-circuiting monotone functions
   Prune search when length exceeds best so far

Logic Rule 3: Reordering tests
   Sort cities to visit closest first

Expression Rule 5: Exploit word parallelism
   Bit mask represents a set of cities

From *Writing Efficient Programs*, 1982
Performance Tools Behind the Scenes

Driver to make experiments easy
   Variety of input data: real, uniform, annular, random symmetric matrices, …
   Count critical operations: distances, MSTs, …

Software profilers

Cost models

Spreadsheet as a “lab notebook”
   Graphs of performance
   Curve fitting