Traveling Salesperson Problem

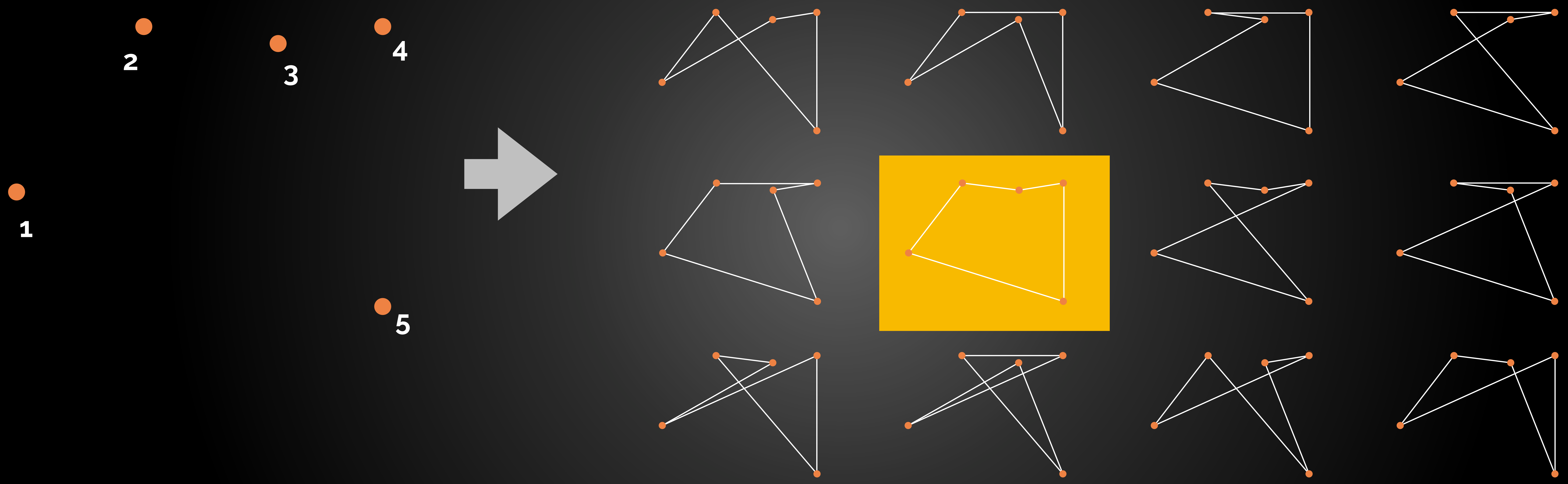
Java — Tips and Tricks

Dr. Jérémie Lumbroso

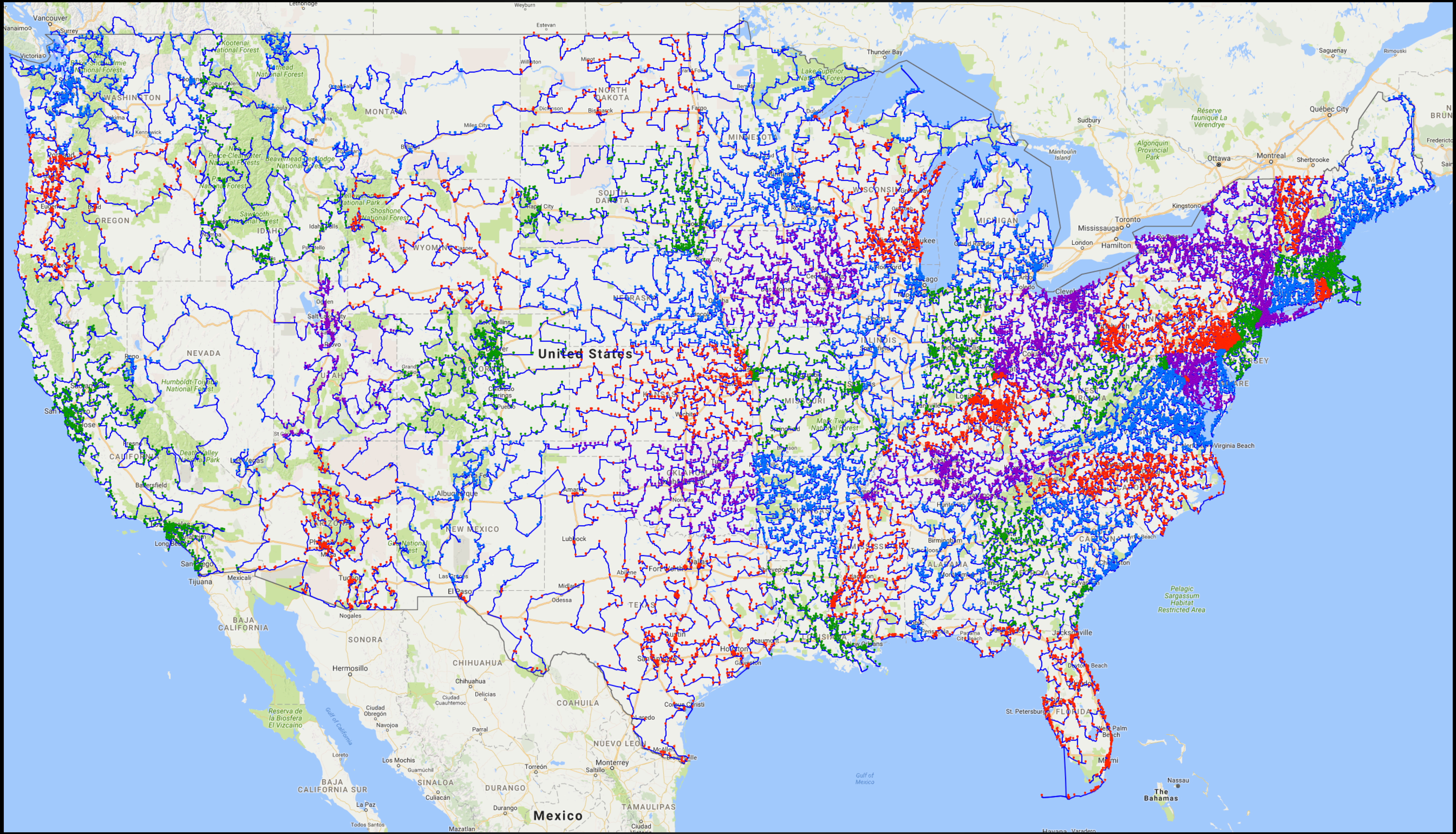
Traveling Salesperson Problem

set of *N* cities

circuit (or "tour") with shortest outline



- Traveling Salesperson needs to drive to *N* cities, using least amount of gas/mileage
- **How many possibilities?** $N!$ orderings / (2 directions * N starting points) = $\mathbf{1/2^{*}(N-1)!}$
- For N=5, $1/2^{*}(N-1)!$ = 12; more generally, $1/2\ (N-1)!$ ~ $.5\ N^N$ which is exponential

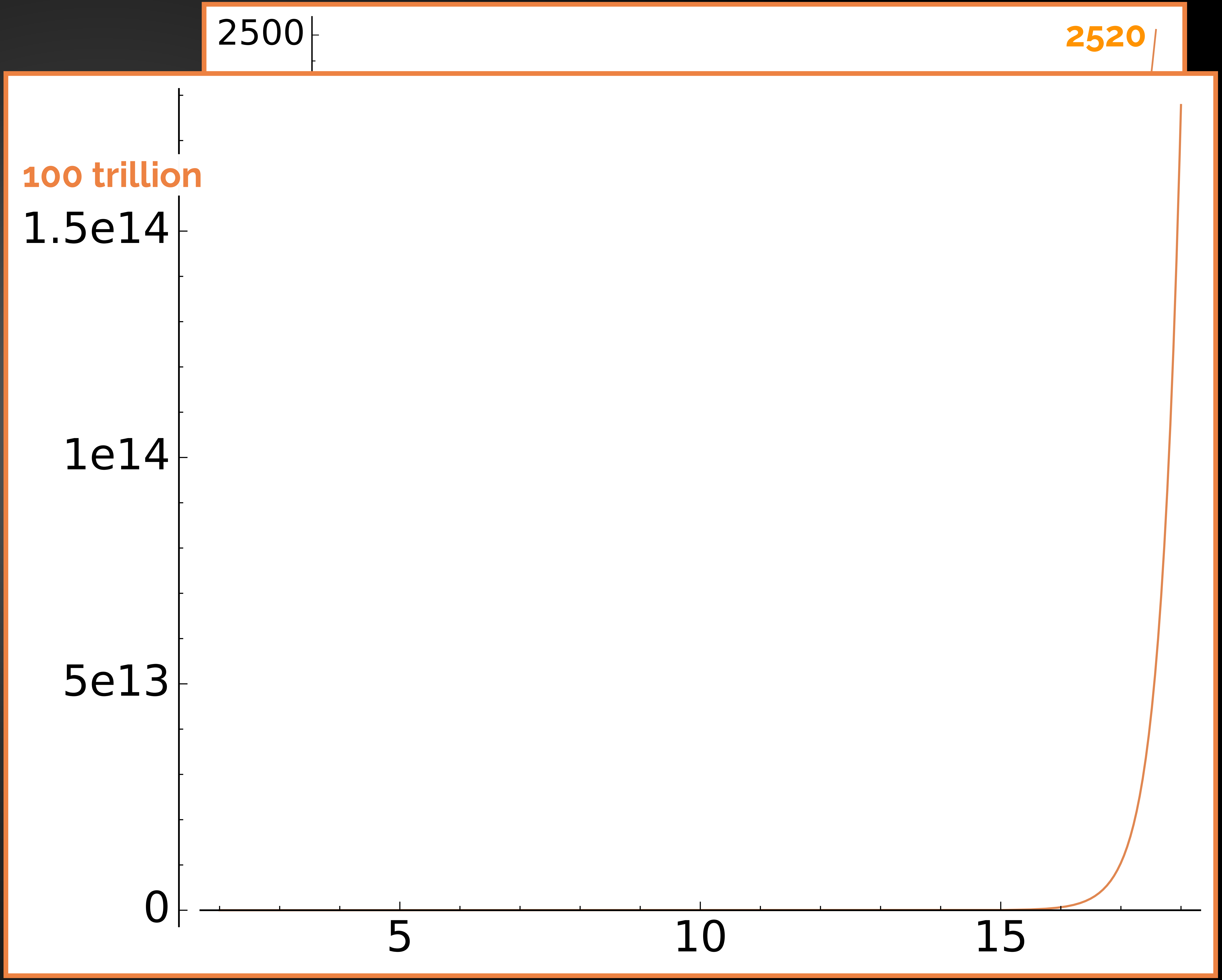


© 2018 William Cook, http://www.math.uwaterloo.ca/tsp/

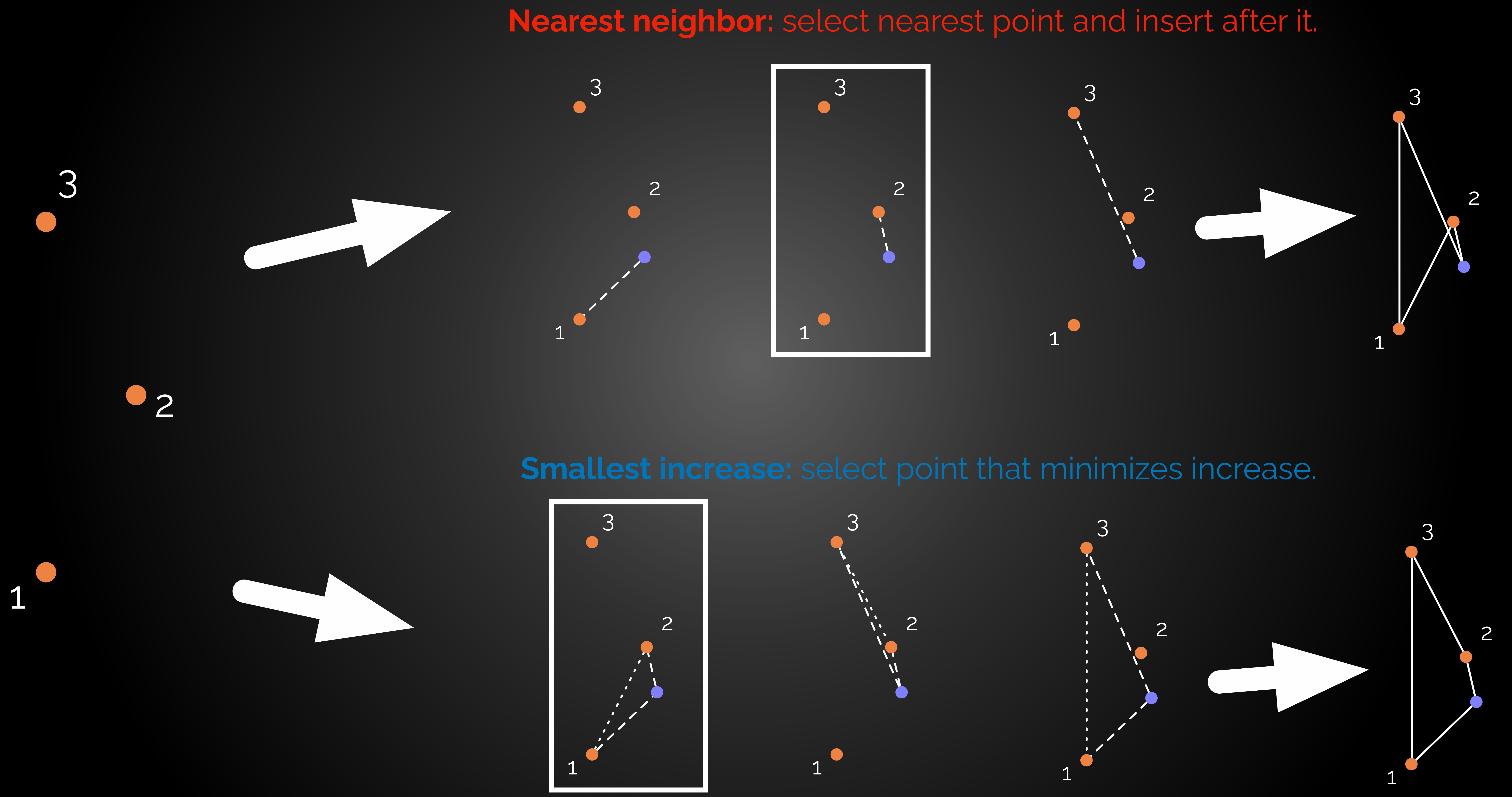
Shortest-possible tour to 49,603 sites from the National Register of Historic Places

Combinatorial Optimization Problems

- Only way to find **optimum** for TSP is to **look at all** possibilities until finding best one(s)
- Possibilities grow exponentially!!! Performance of naive approach is **factorial**, *N*!
- In practice, **heuristics** can exploit specificities of a dataset or problem to perform accurately and efficiently
- But TSP belongs to broader class of **universally** difficult problems (**NP-hard**)—details in upcoming lectures



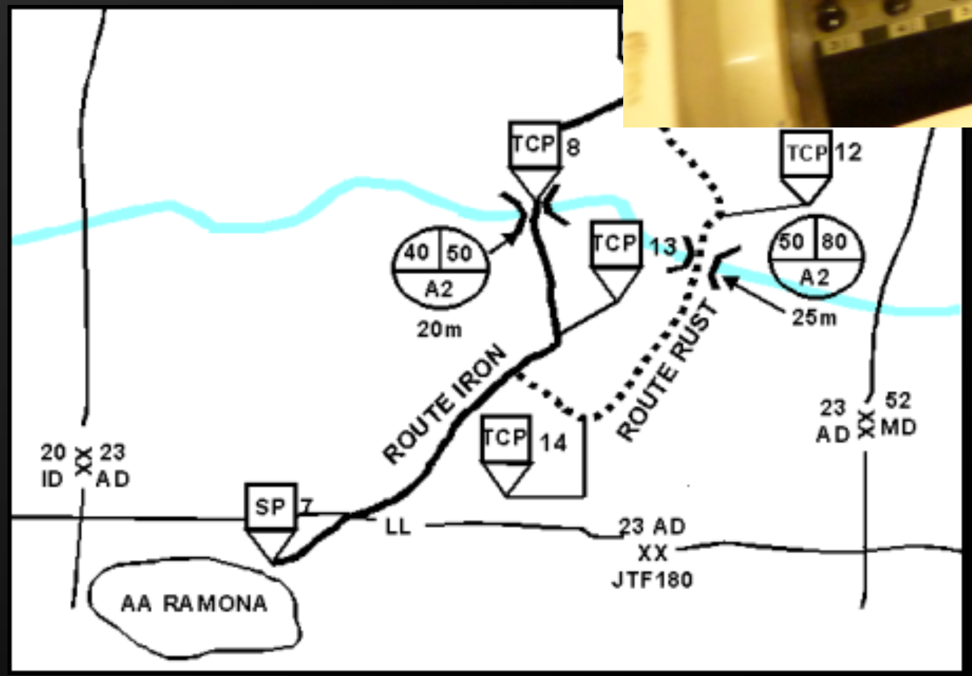
Two Heuristics



Measure increase = *(Length of both dashed lines) - (Length of dotted line)*

Some Applications

- School bus routing, since 1972
- (Delivery) vehicle routing in city, since 1974
- Order picking problem in warehouses, since 1983
- Drilling Printed Circuit Boards (PCBs), since 1991
- Military mission planning, since 1996, and in UAVs, since 1998
- Many other applications, in genomics, in medicine, *etc.*

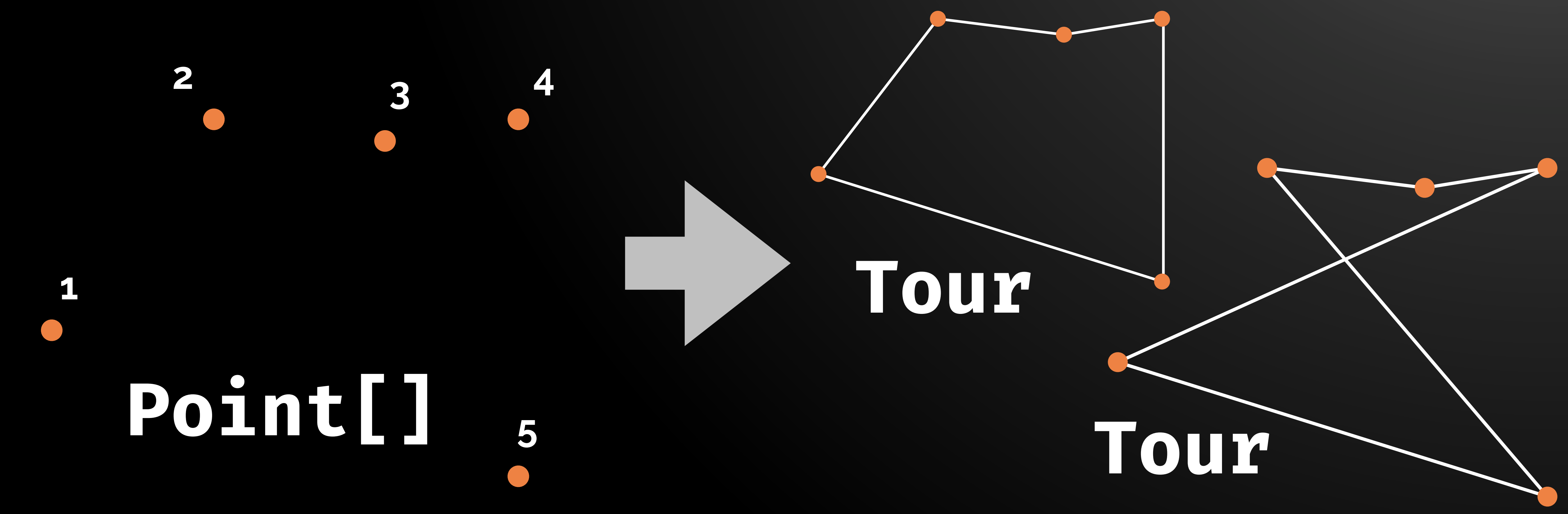


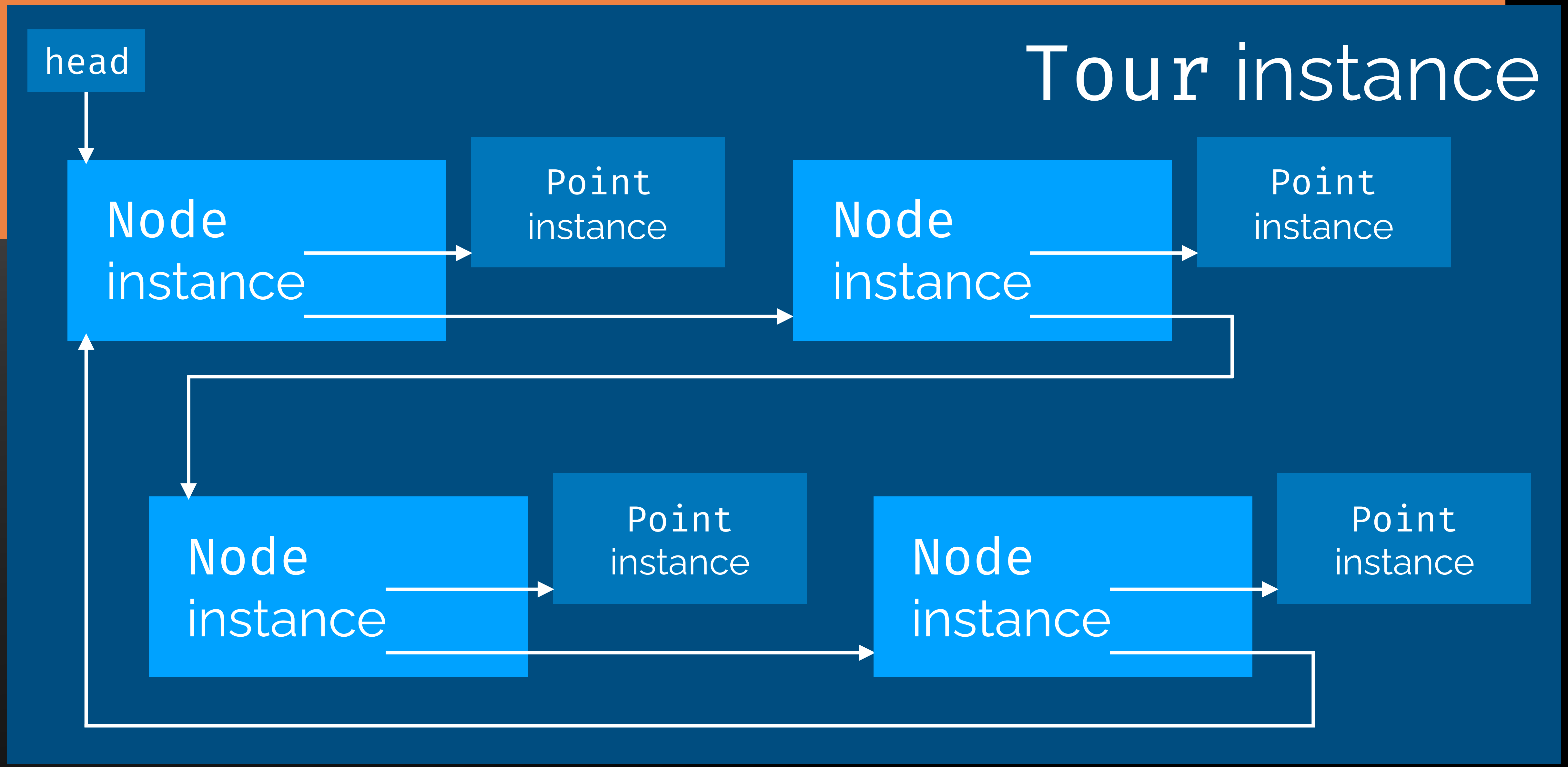
https://bit.ly/TSPApplicationsPDF

Assignment Specifics

Your Job: Implement the Tour API

```
public class Tour {
  public         Tour()                                       // creates an empty tour
  public         Tour(Point a, Point b, Point c, Point d)     // creates the 4-point tour
                                                              //    a→b→c→d→a (for debugging)
  public    int size()                                        // returns the number of points in this tour
  public double length()                                      // returns the length of this tour
  public String toString()                                    // returns string representation of this tour
  public   void draw()                                        // draws this tour to standard drawing
  public   void insertNearest(Point p)                        // inserts p using nearest neighbor heuristic
  public   void insertSmallest(Point p)                       // inserts p using smallest increase heuristic

  // tests this class
  public static void main(String[] args)
}
```

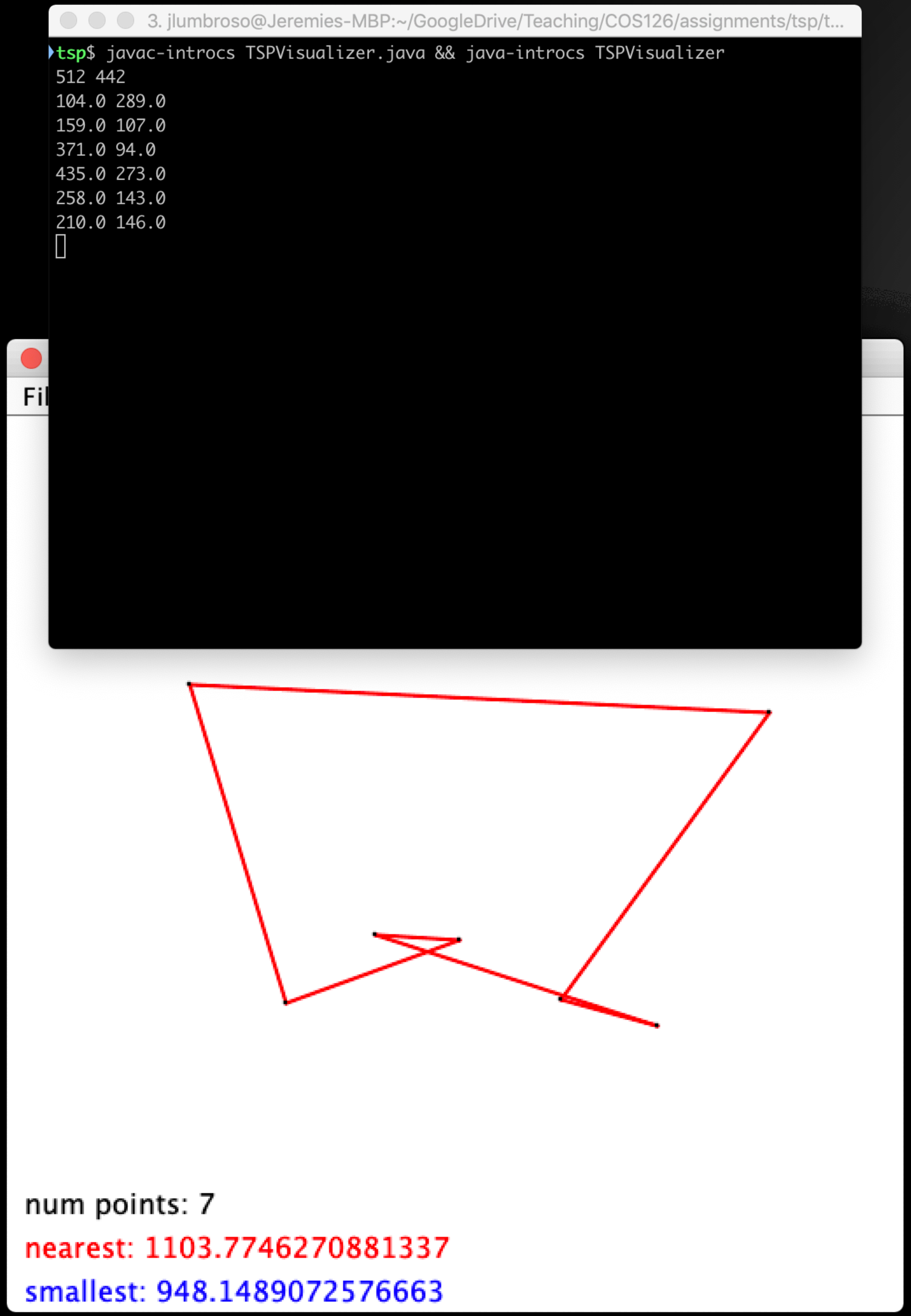




Assignment Inputs and Goals

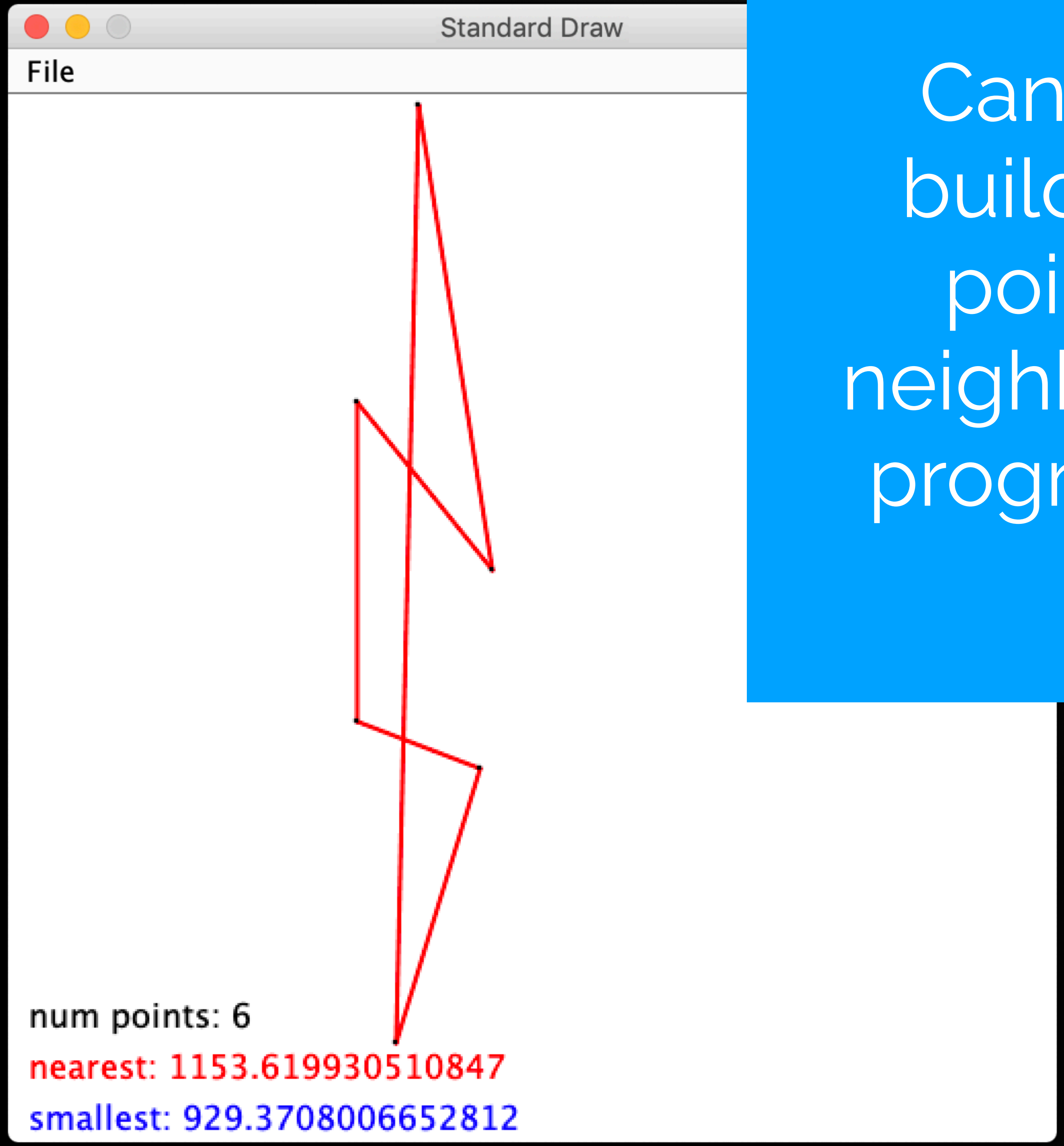
- You have to implement a class `Tour.java`
- You are provided with `Point.java`, the `Node` class, **several test clients** and sample datasets, to check whether your implementation is correct
- The assignment introduces you to **linked lists**
  - *Can you use a data type that is provided to you?* see use of `Point`
  - *Can you use a private node type?* see `Node` definition and use
  - *Can you traverse a list?* see `Tour.size()`, `Tour.length()`
  - *What about when there are different base cases?* `Tour.toString()`
  - *Can you modify a circular list?* `Tour.insertNearest()` and other

TSPVisualizer (1)



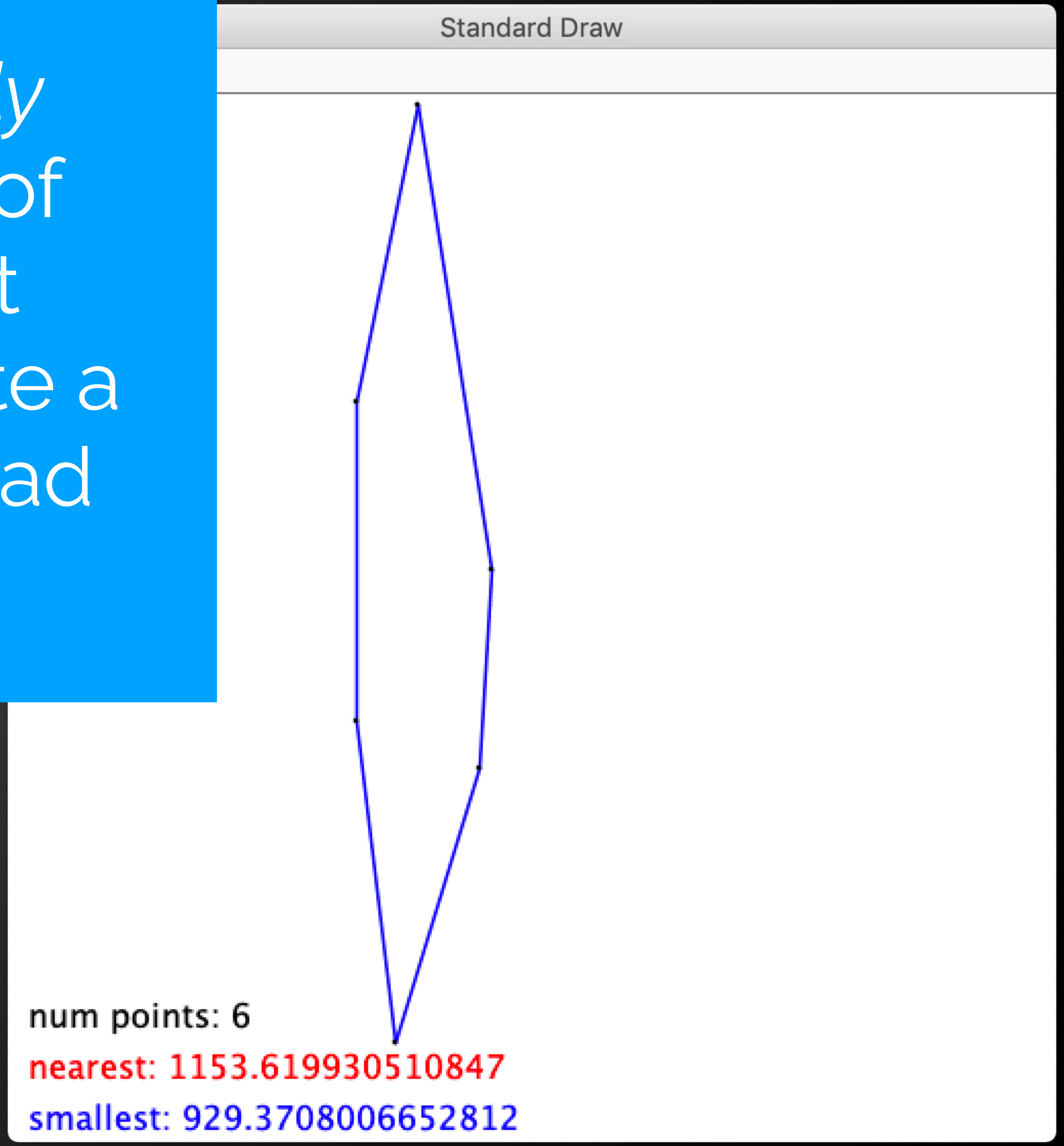
- Test client provided in the project files, which uses **your** Tour implementation, calling the following to color the outlines, before `Tour.draw()`:
  - `StdDraw.setPenColor(StdDraw.RED);`
- Can take a starting set of points; and outputs points in its diagram to the console
- Initially **nearest neighbor** heuristic and **smallest increase** heuristic appear similar
- The nearest neighbor heuristic does not always do what we intuitively want it to: It depends on the order in which points have been added, not proximity

TSPVisualizer (2)



Challenge for the Bored 1

Can you *systematically* build bad sequences of points for our nearest neighbor heuristic? Write a program to generate bad sequences?



Tips and Tricks

The Point API

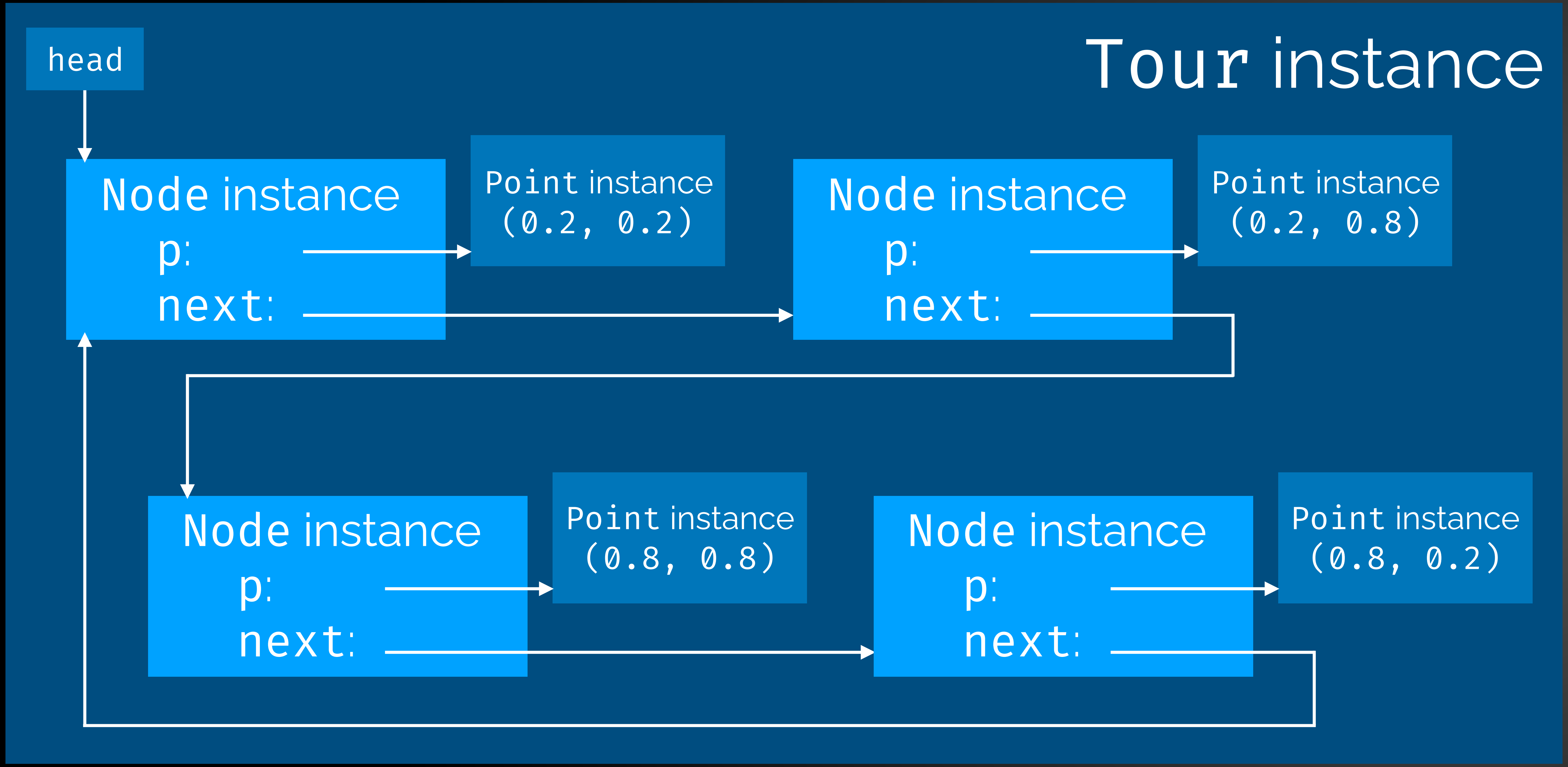
```
public class Point {
    public        Point(double x, double y)  // creates the point (x, y)
    public double distanceTo(Point that)     // returns the Euclidean distance between the two points
    public   void draw()                     // draws this point to standard drawing
    public   void drawTo(Point that)         // draws the line segment between the two points
    public String toString()                 // returns a string representation of this point
}
```

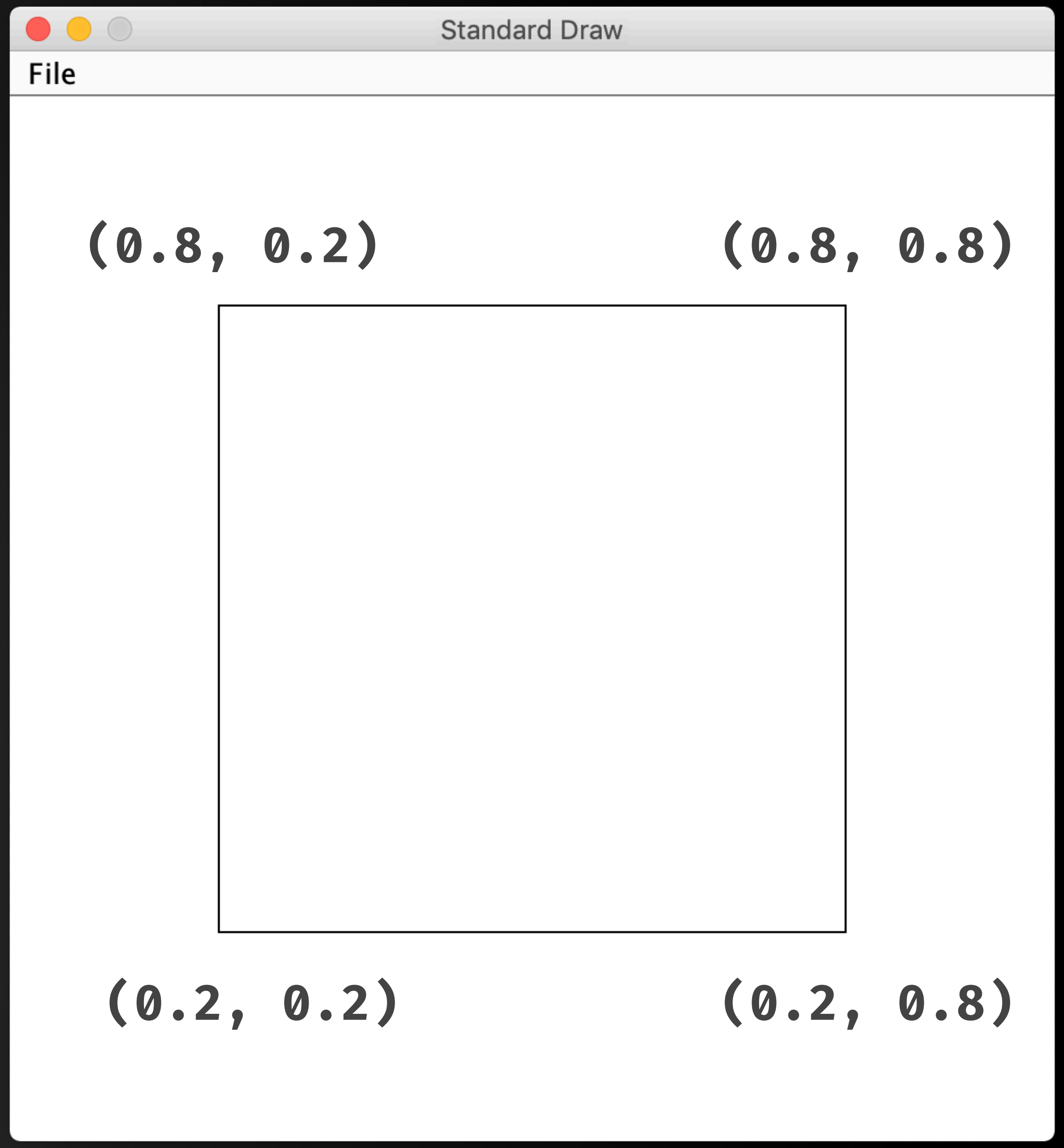
- No way to access the x or y coordinate of a `Point` class instance
- In `Tour.length()`, to measure perimeter of tour:
  - Use `Point.distanceTo()`
- In `Tour.toString()`, to list coordinates of all points:
  - Use `Point.toString()`
- In `Tour.draw()`, to draw the outline of the tour:
  - Use `Point.drawTo()`

Challenge for the Bored 2

I can think of two ways to extract the coordinates anyway, a **math-based** and **text-based** method. Can you figure them out?

Circular Linked List

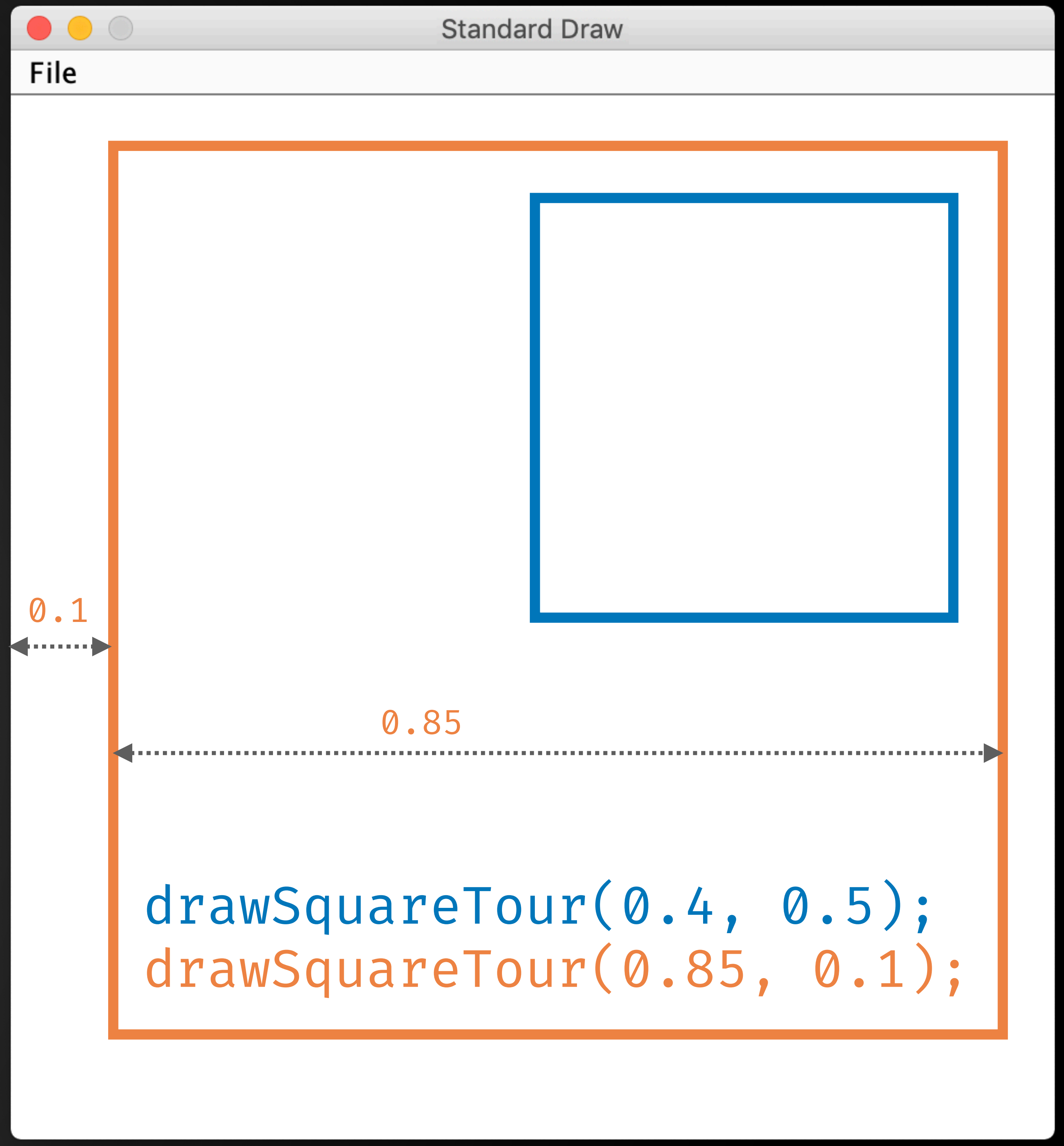





```java
public static void main(String[] args) {
    // Or: Tour square = createSquareTour(0.6, 0.2);
    Tour square = new Tour(new Point(0.2, 0.2),
                           new Point(0.2, 0.8),
                           new Point(0.8, 0.8),
                           new Point(0.8, 0.2));
    square.draw();
}
```

Make Helper Functions for Testing

```java
// Create a square tour of side alpha, shifted by beta
private static Tour createSquareTour(double alpha, double beta) {
    return new Tour(
            new Point(beta + 0.0, beta + 0.0),
            new Point(beta + 0.0, beta + 1.0 * alpha),
            new Point(beta + 1.0 * alpha, beta + 1.0 * alpha),
            new Point(beta + 1.0 * alpha, beta + 0.0)
    );
}

//
private static boolean testOne(double alpha) {
    Tour test = createSquareTour(alpha);
    boolean sizeTest = (test.size() == 4);
    boolean lengthTest = (Math.abs(test.length() - 4.0 * alpha) <= 0.001);
    return sizeTest && lengthTest;
}

// ... possibly called this way in main() ...
int NUM_TEST_REPETITIONS = 1000;
for (int i = 0; i < NUM_TEST_REPETITIONS; i++) {
    double alpha = StdRandom.uniform(0.5, 100.0);
    if (!testOne(alpha))
        StdOut.println("testOne failed, alpha = " + alpha);
}
```



Any method that makes it easier to write more tests is a good helper method!

Helper Functions for Insertion

- Modularity is often very desirable: Part of the point of functions
- Helper functions can be useful in many situations
  - To avoid duplicating the same logic in several places:

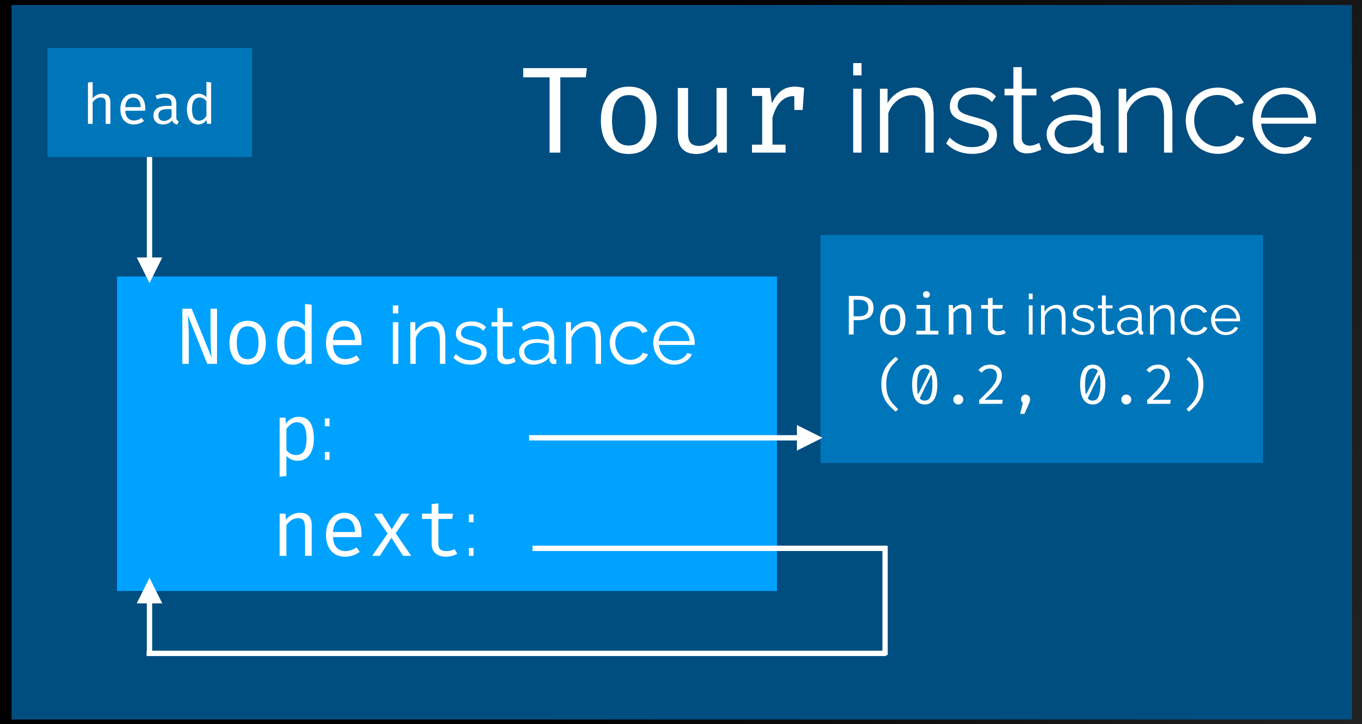
```
// Insert a new node containing point newPoint right after
// the node that is referenced by the parameter cursor

private void insertPointAfter(Node cursor, Point newPoint)
```

  - To make the calling code clearer, by abstracting a complicated sequence of operations to a function

```
// Compute the increase in tour length that would result from
// inserting point newPoint after the node at cursor

private double computeIncrease(Node cursor, Point newPoint)
```

0 nodes

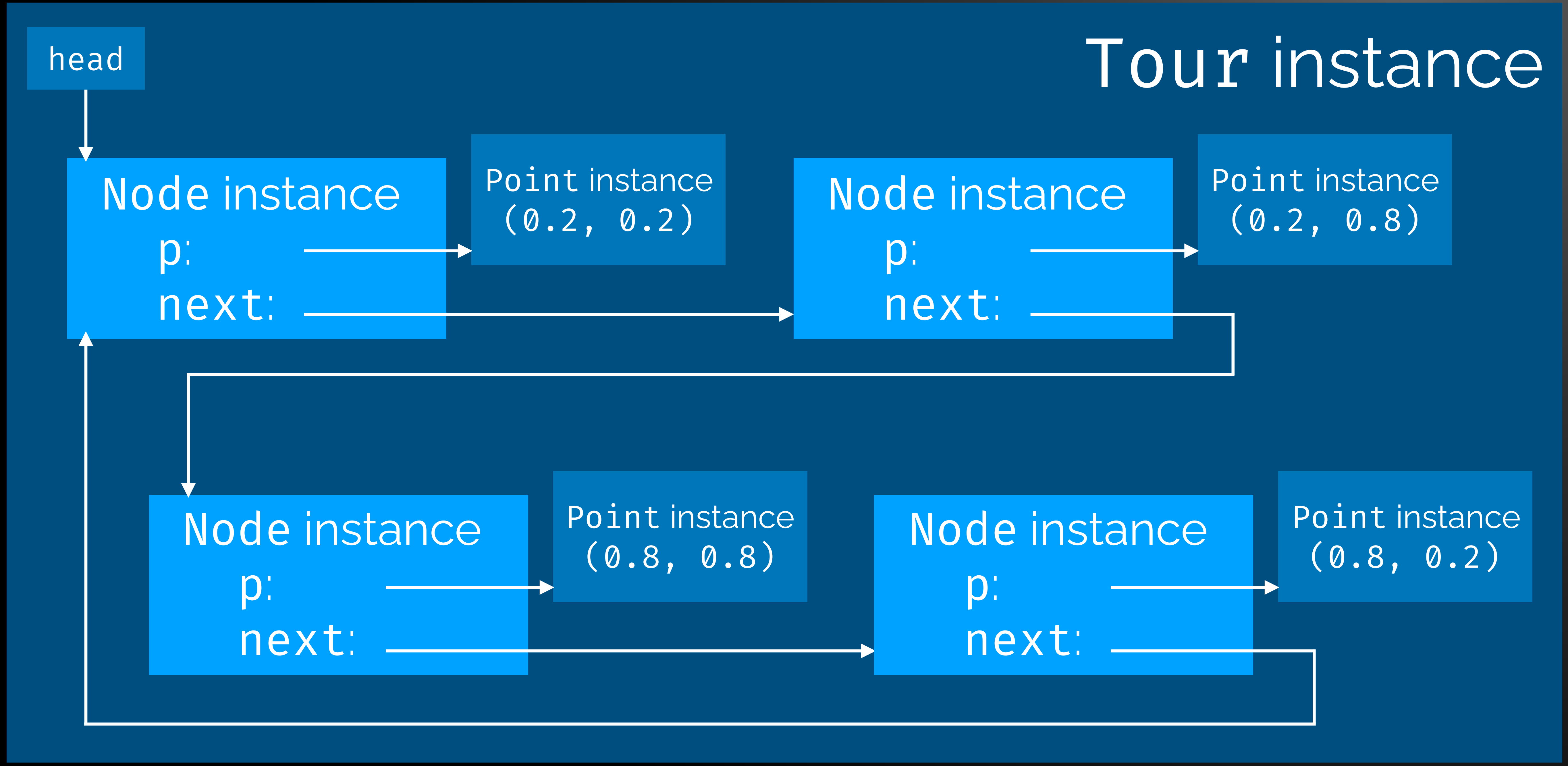
1 node



Edge-cases/Base cases?

- Correctly identifying [smallest possible number of] edge-case(s) for list operations, helps code complexity
- Using the **`do { ... } while ( ... )`** construct allows you to writer shorter code
- Circular list vs. normal lists saves you a few edge cases...

many nodes




```java
public int traverseCircularList() {
    // <... some initialization...>

    if (head == null) return ...;

    Node x = head;
    do {
        // <... do something with element x ...>
        x = x.next;
    } while (x != first);

    // <... some more work...>

    return ...;
}
```

Pseudo-Code for TSP Approximation

```
tour ← []
for i = 1 to N:

    p ← pointsToInsert[i]
    bestValueSoFar ← <default value>
    bestCandidateSoFar ← null

    for each point x on tour:
        if computeValue(x, p) < bestValueSoFar:
            bestValueSoFar ← computeValue(x, p)
            bestCandidateSoFar ← x

    insertPointAfter(bestCandidateSoFar, p)
```

Real-World Example: Additional Constraints

ORION: The algorithm proving that left isn't right

October 2016

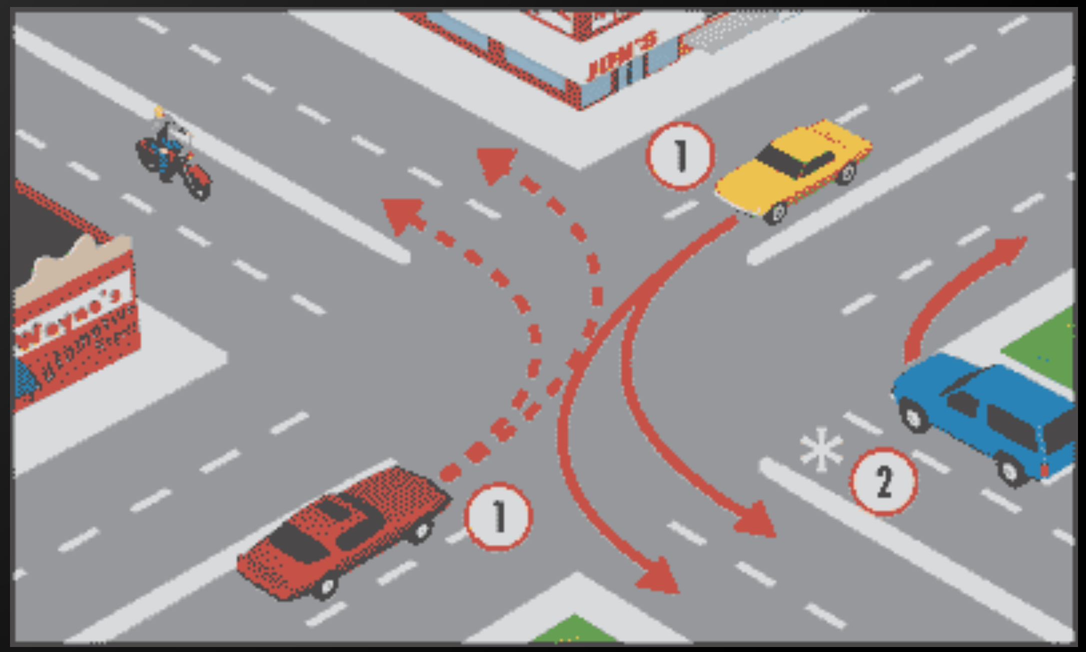
[...] Left turns mean idling, which increases the time a route takes. Left turns mean going against traffic, which increases exposure to oncoming cars. **Right turns are faster.** Right turns save fuel.

Because most UPS managers have been UPS drivers, they have driven the routes and **plotted on maps how to drive them with as many right-hand loops as possible**. They knew right turns were the way to go, but that knowledge was in their heads.

"Before computers, engineering was about measurement and process," says Jack Levis, senior director of process management at UPS. "UPS has always believed in data, not intuition."

Eventually, UPS's technology caught up with experience. The result is ORION (or **On-Road Integrated Optimization and Navigation**). By optimizing delivery routes in regard to distance, fuel and time, ORION seeks to solve the Traveling Salesman Problem, which has stumped scientists for more than 200 years. [...]

- UPS routinely computes TSP tours
- Eliminating 1 mile, per driver, per day over one year can save up to **$50 million**
- Typical optimization: Prefer right-turns over left-turns (essentially because they require less idling)



The Lin-Kernighan Heuristic

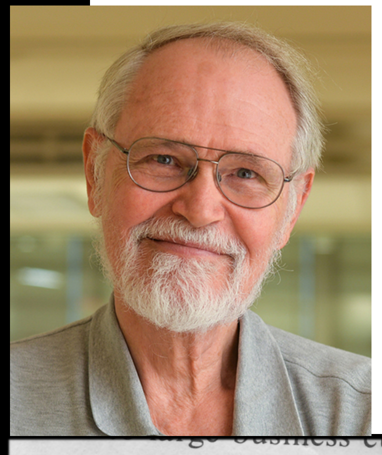
An Effective Heuristic Algorithm for the Traveling-Salesman Problem

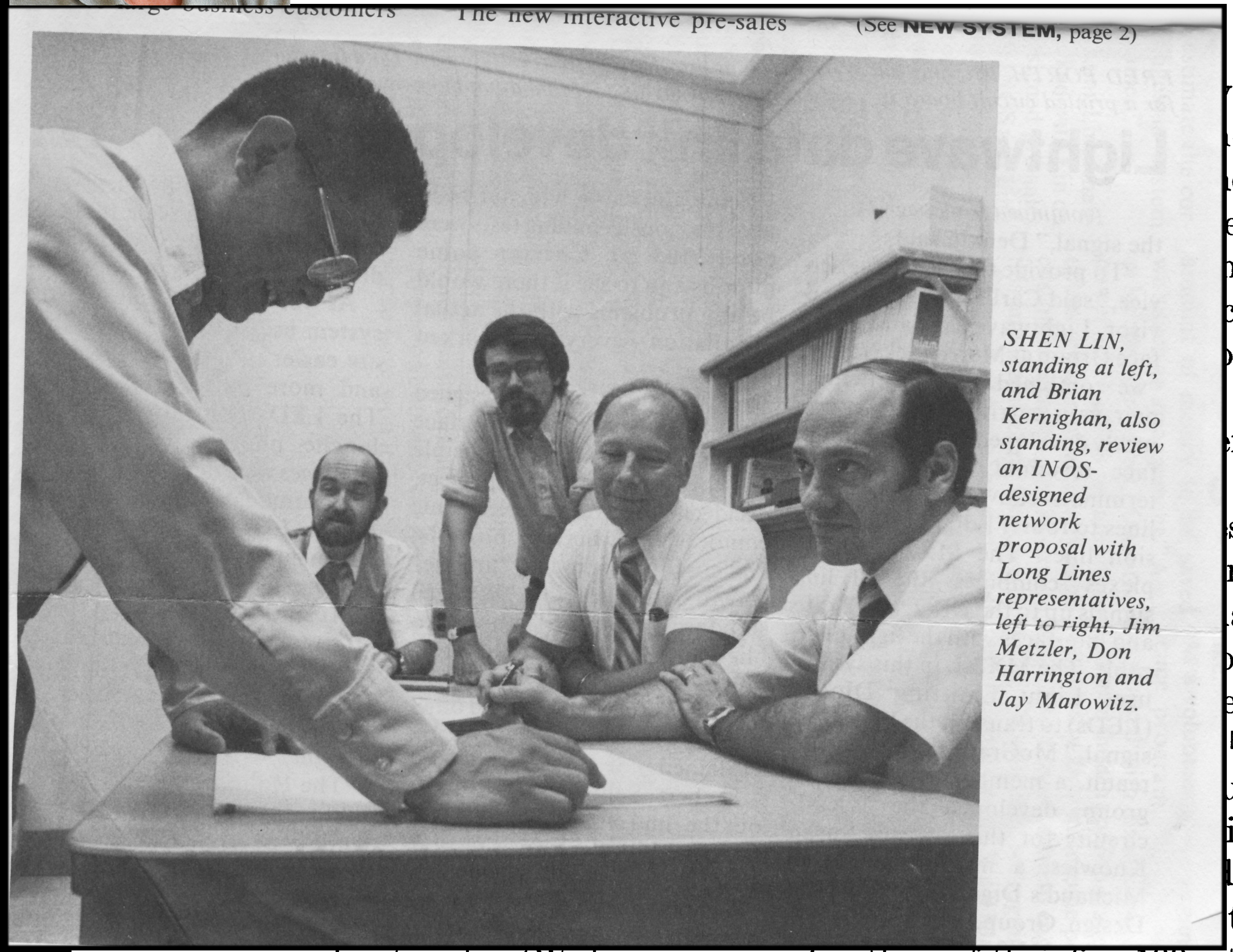
S. Lin and B. W. Kernighan

Bell Telephone Laboratories, Incorporated, Murray Hill, N.J.

(Received October 15, 1971)

This paper discusses a highly effective heuristic procedure for generating optimum and near-optimum solutions for the symmetric traveling-salesman problem. The procedure is based on a general approach to heuristics that is believed to have wide applicability in combinatorial optimization problems. The procedure produces optimum solutions for all problems tested, 'classical' problems appearing in the literature, as well as randomly generated test problems, up to 110 cities. Run times grow approximately as n^2; in absolute terms, a typical 100-city problem requires less than 25 seconds for one case (GE635), and about three minutes to obtain the optimum with above 95 per cent confidence.





ven an n by n symmetric
-length tour that visits
er notions such as time,
esent any such measure.
mited success.[1] Exact
c methods produce good
but provide no guarantee
heuristics, effectiveness
re has been little work

s a method that solves
reasonable time. How-
ass, the procedure must
branch and bound—and
ey report on is 64 cities.
[9] who use several fast,
utions, and then apply
interaction") to try for
l to large problems (200
the results are generally
suboptimal. (We have improved on three of their five 100-city problems.) Further-

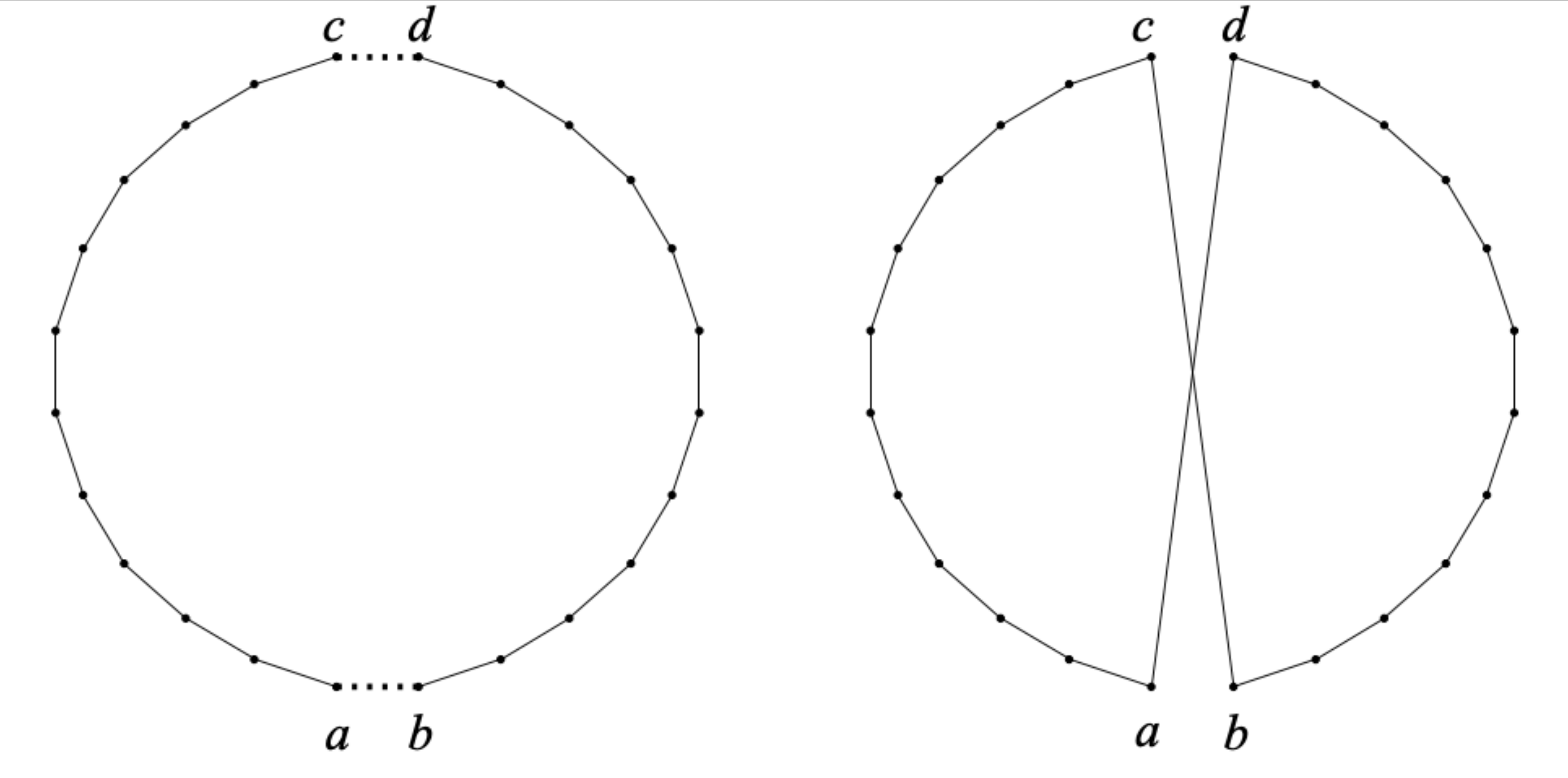


Figure 1. A 2-Opt move: original tour on the left and resulting tour on the right.

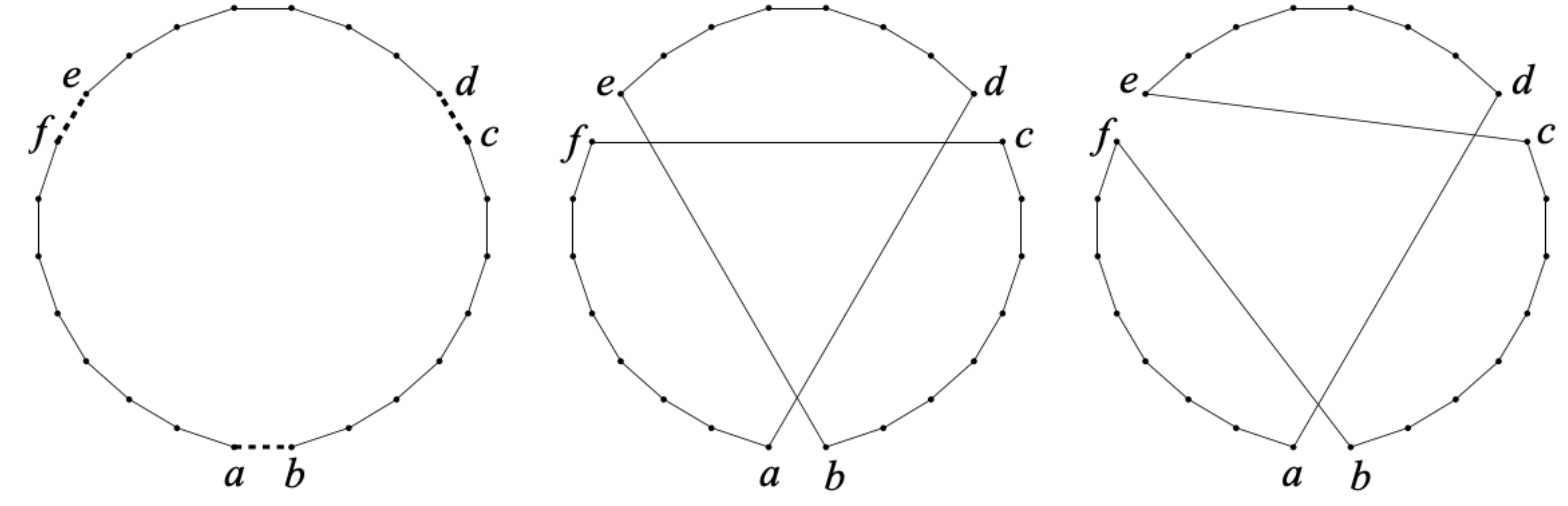


Figure 2. Two possible 3-Opt moves: original tour on the left and resulting tours on the right.

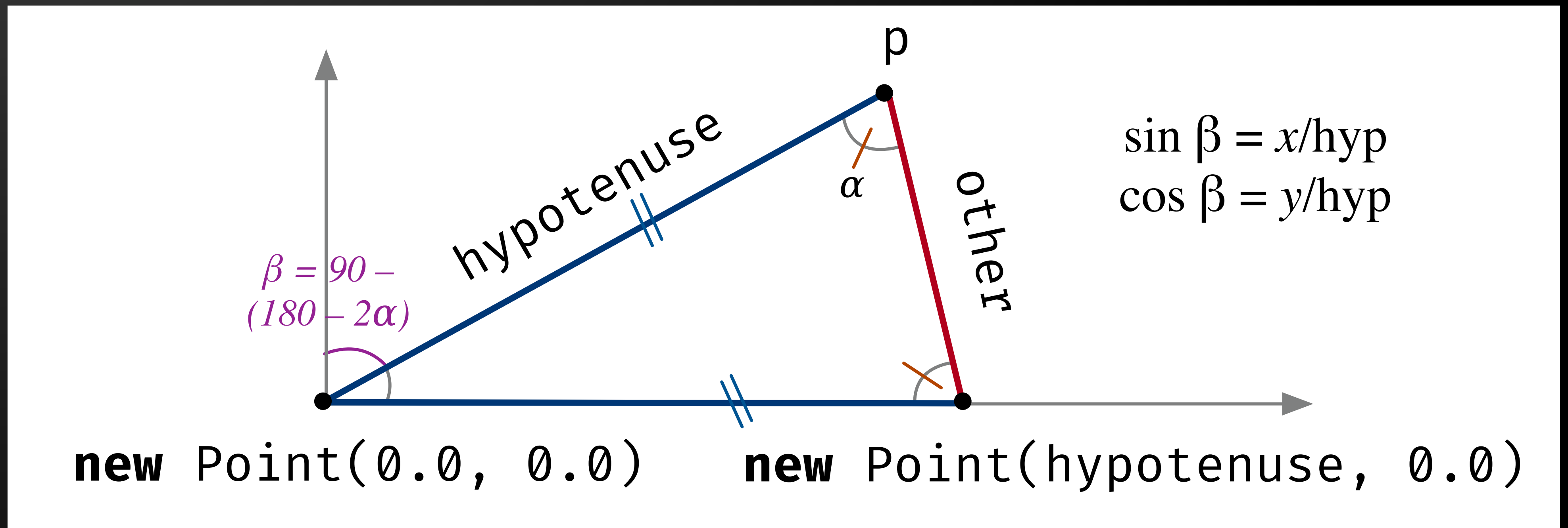
http://bit.ly/TSPHistoryPDF

Challenge for the Bored

How to circumvent an API to get the information you want/need?

```java
private static double[] extractPointByText(Point p) {
    String s = p.toString();
    String x = "", y = "";
    int cursor = 1;

    // Extract first number
    while (s.charAt(cursor) != ',') {
        x += s.charAt(cursor);
        cursor++;
    }

    // Skip whitespace
    while (s.charAt(cursor) == ' ' ||
           s.charAt(cursor) == ',')
        cursor++;

    // Extract second number
    while (s.charAt(cursor) != ')') {
        y += s.charAt(cursor);
        cursor++;
    }

    return new double[] { Double.parseDouble(x),
                          Double.parseDouble(y) };
}
```

```java
private static double[] extractPointByMath(Point p) {
    double hypotenuse = p.distanceTo(new Point(0, 0));
    double other = p.distanceTo(new Point(hypotenuse, 0));

    double angle = Math.toDegrees(
        Math.acos((other / 2.0) / hypotenuse));
    double otherAngle = 90.0 - (180.0 - 2 * angle);

    double x = Math.sin(Math.toRadians(otherAngle)) * hypotenuse;
    double y = Math.cos(Math.toRadians(otherAngle)) * hypotenuse;

    return new double[] { x, y };
}
```



Analysis

My timings: Timing of a single random instance of size N with both heuristics

N	lengthNearest	timeNearest	lengthSmallest	timeSmallest
500	18934	0.00	11168	0.00
1000	26775	0.01	15929	0.01
2000	37855	0.01	22281	0.01
4000	52117	0.04	31029	0.05
8000	74289	0.21	43780	0.27
16000	105392	1.27	62208	1.41
32000	149731	6.30	87921	6.44
64000	210791	43.36	123992	32.81
128000	297889	248.15	175256	230.00

First experiment that last longer than 60 seconds

We assume the performance is polynomial:

$$f(N) = aN^b$$

Thus we can use the doubling method:

$$\frac{f(2N)}{f(N)} = \frac{a(2N)^b}{aN^b}$$

With which we solve:

$$b = \log_2\left(\frac{f(2N)}{f(N)}\right)$$

$$a = \frac{f(2N)}{(2N)^b}$$

Creating and preparing a dataset

```java
// Create set of N random points (borrowed from TSPTimer.java)
private static Point[] randomPointSet(int N) {
    double lo = 0.0, hi = 600.0;
    Point[] testSet = new Point[N];
    for (int i = 0; i < N; i++) {
        double x = StdRandom.uniform(lo, hi);
        double y = StdRandom.uniform(lo, hi);
        testSet[i] = new Point(x, y);
    }
    return testSet;
}

// Time both heuristics with a random instance of N points
private static String timeSingleBoth(int N) {
    Point[] testSet = randomPointSet(N);

    // <... do computations and measure with Stopwatch ...>

    return (N + "," +
            lengthNearest + "," +
            elapsedNearest + "," +
            lengthSmallest + "," +
            elapsedSmallest);
}
```

2. jlumbroso@Jeremies-MBP:~/GoogleDrive/Teaching/COS126/assignments/tsp (...

```
tsp$ javac-introcs Tour.java && java-introcs Tour
N,lengthNearest,timeNearest,lengthSmallest,timeSmallest
500,18934.05221355573,0.003,11167.986279062763,0.003
1000,26774.506922171782,0.005,15929.489748561908,0.008
2000,37854.70037288836,0.008,22280.73070191083,0.012
4000,52116.85594778351,0.037,31028.759032544785,0.047
8000,74289.35199621355,0.211,43780.067587295074,0.272
16000,105391.61893569703,1.273,62207.724440124504,1.406
32000,149730.9366426918,6.304,87920.95460680297,6.437
64000,210791.2207307945,43.358,123991.9159385805,32.81
128000,297889.0396289771,248.149,175256.2756630417,230.003
```

My TSP Analysis

File Edit View Insert Format Data Tools Add-ons Help

100% $ % .0 .00 123 Arial

fx =round(log(C3/C2,2),2)

	A	B	C	D	E
1	N	lengthNearest	timeNearest	dblNearest	lengthSm
2	500	18934.05221	0.003		11167
3	1000	26774.50692	0.005	0.74	15929
4	2000	37854.70037	0.008	0.68	2228
5	4000	52116.85595	0.037	2.21	31028
6	8000	74289.352	0.211	2.51	43780
7	16000	105391.6189	1.273	2.59	62207
8	32000	149730.9366	6.304	2.31	87920
9	64000	210791.2207	43.358	2.78	12399
10	128000	297889.0396	248.149	2.52	17525
11					
12					
13					
14					
15					
16					

Sheet1

Better Estimates Through Averaging

My TSP Analysis

File Edit View Insert Format Data Tools Add-ons Help

fx =round(log(C3/C2,2),2)

	A	B	C	D	E
1	N	lengthNearest	timeNearest	dblNearest	lengthSm
2	500	18934.05221	0.003		11167
3	1000	26774.50692	0.005	0.74	15929
4	2000	37854.70037	0.008	0.68	2228
5	4000	52116.85595	0.037	2.21	31028
6	8000	74289.352	0.211	2.51	43780
7	16000	105391.6189	1.273	2.59	62207
8	32000	149730.9366	6.304	2.31	87920
9	64000	210791.2207	43.358	2.78	12399
10	128000	297889.0396	248.149	2.52	17525
11					
12					
13					
14					
15					
16					

Sheet1

For more accurate readings, must average timing across *K* different executions (with *K* different random sets of points)

	A	B	C	D	E	F
1	N	K	nearestAvgTime	dblNearest	smallestAvgTime	dblSmallest
2	500	10	9.00E-04		0.0017	
3	1000	10	0.0014	0.64	0.0032	0.91
4	2000	10	0.0073	2.38	0.0112	1.81
5	4000	10	0.0346	2.24	0.0475	2.08
6	8000	10	0.1942	2.49	0.2442	2.36
7	16000	10	1.174	2.6	1.2777	2.39
8	[illegible]	[illegible]	[illegible]	2.39	6.5816	2.36
9	[illegible]	[illegible]	[illegible]	2.38	31.6238	2.26

Question for the Curious

Compute the ratio of the length of the tour created with the nearest heuristic, and with smallest increase heuristic?

Have fun!

I am sticking around to
answer questions