Connecting the Dots: The Ins and Outs of TSP Art

Robert Bosch
Oberlin College
Oberlin, OH 44074
(bobb@cs.oberlin.edu)

Abstract

Previous papers on TSP Art focused on creating TSPs that, when solved, yield tours that resemble user-supplied target images. Here we describe how to construct tours that wind through the cities in such a way that certain pairs of user-selected, city-free regions will end up on the same side of the tour, while others will end up on opposite sides.

1 Introduction

Producing a piece of TSP Art involves

1. converting a black and white user-supplied target image into an arrangment of dots,
2. thinking of the dots as the cities of a Traveling Salesman Problem, in which a salesman based in one of the cities must visit each of the other cities exactly once before returning home, while minimizing total distance traveled,
3. solving the TSP—in some cases with a slow-but-sure exact method that's guaranteed to produce an optimal solution, but in most cases with a much faster heuristic designed to have a good chance of finding a near-optimal solution—and finally,
4. drawing the salesman's tour.

Figure 1 illustrates the process. Note that the optimal tour is topologically equivalent to a circle; it is a closed curve, and none of its edges cross. This was no accident. Since the total distance traveled by the salesman is the sum of the lengths of the edges (line segments) that make up his tour, and since each edge length is computed using the Euclidean distance formula, all TSP Art TSPs are geometric TSPs. Optimal tours for geometric TSPs cannot have edge crossings [1].

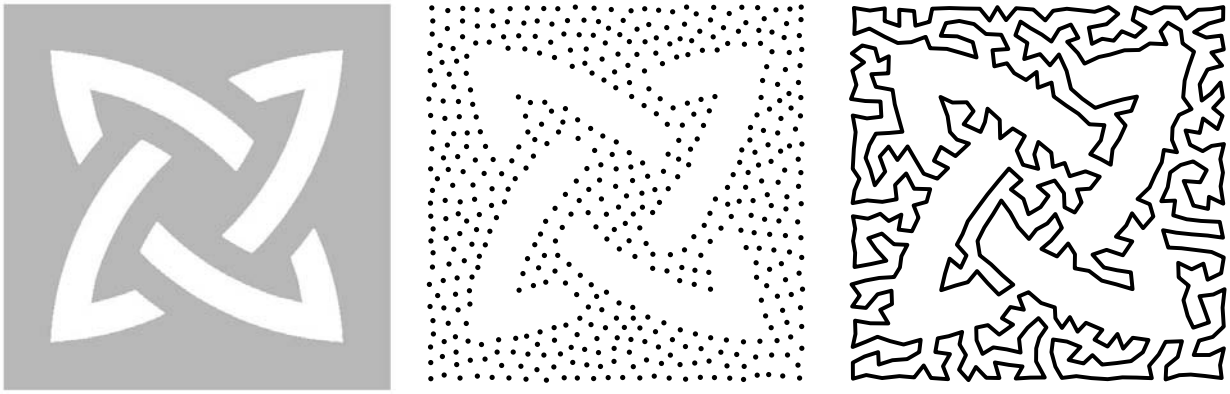

Figure 1: (a) the target image, (b) a 500-city TSP, (c) the optimal tour.

Previous papers on TSP Art [2,6] focused on arranging the dots (the cities). Here we focus on connecting the dots. We show that by adding side constraints to Dantzig, Fulkerson, and Johnson's integer programming formulation [4], we can construct tours that wind through the cities in such a way that certain pairs of user-selected, city-free regions will end up on the same side of the tour, while others will end up on opposite sides.

We do this to gain some control over the interior (and exterior) of the tour. To see why this is desirable, take a closer look at the Figure 1 example. First, note that the target image is what knot theorists call a two-component link. Next, note that the TSP's 500 cities define four city-free regions, marked *A* through *D* in Figure 2(a). Regions *A* and *C* belong to one component of the link, and regions *B* and *D* belong to the other. Finally, note that the optimal tour ends up separating regions *A* and *B* from regions *C* and *D*; regions *C* and *D* are "inside" the tour, while regions *A* and *B* are on the "outside". Consequently, when we shade the tour's interior—in an effort to add color or an additional shade of gray to our TSP Art—we get an image, displayed in Figure 2(b), that not only fails to do justice to the target image but, in addition, lacks both symmetry and aesthetic appeal. The alternate tour displayed with shaded interior in Figure 2(c) is much better: it *looks* like a two-component link, it has 180° rotational symmetry (at least when viewed from a distance), and it is beautiful.

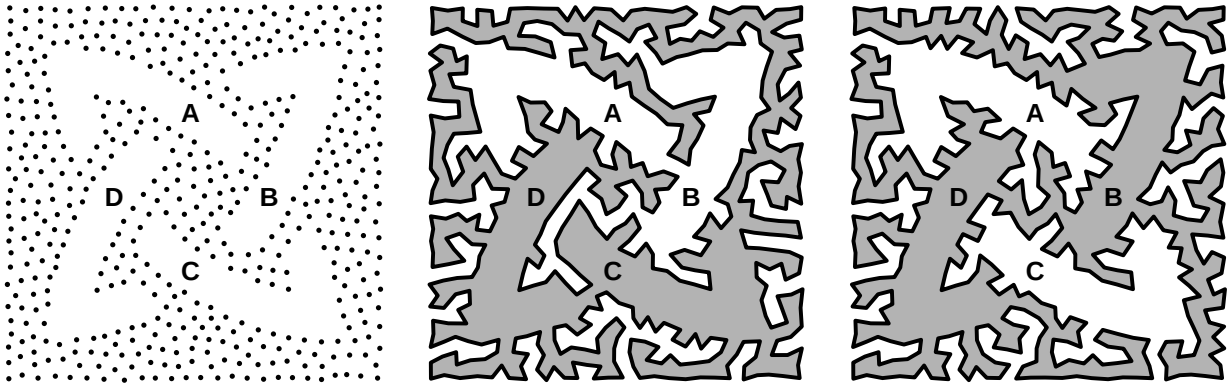

Figure 2: (a) the four city-free regions, (b) the shaded interior of the optimal tour, (c) the shaded interior of an alternate tour.

2 Integer Programming and the TSP

The idea behind Dantzig, Fulkerson, and Johnson's integer programming formulation is quite simple. When we build a tour, we face a binary (yes-no) decision for each pair of cities: Should we force cities i and j to be adjacent on the salesman's tour? If the answer is yes, we connect dots i and j with a straight line segment, and we charge ourselves a cost, $c_{i,j}$, the Euclidean length of the straight line segment connecting i and j. If the answer is no, we don't connect them and we don't charge ourselves anything. To model this mathematically, we introduce a binary decision variable

$$x_{i,j} = \begin{cases} 1 & \text{if cities } i \text{ and } j \text{ are adjacent} \\ & \text{on the salesman's tour, and} \\ 0 & \text{if they are not,} \end{cases}$$

for each pair of cities. To avoid double counting, we keep $i < j$. With these decision variables, the total cost, or tour length, ends up being

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} c_{i,j} x_{i,j}. \tag{1}$$

The DFJ formulation minimizes (1) subject to constraints that ensure that the values assigned to the $x_{i,j}$'s give rise to a legitimate tour. Two types of constraints are employed. First, for each city j, there is a degree constraint,

$$\sum_{i=1}^{j-1} x_{i,j} + \sum_{i=j+1}^{n} x_{j,i} = 2, \tag{2}$$

that ensures that city j is adjacent to precisely two other cities. It guarantees that the salesman will have a way in and a way out of each city j. In graph theoretic terms, each vertex of the $x_{i,j}$-induced subgraph will have degree two.

Unfortunately, the degree constraints are not enough; many more constraints are needed. To see why, consider the 50-city TSP, displayed in Figure 3(a), whose cities were positioned so that they'd form the image of a circle split into two regions—labeled *A* and *B*—by a vertical bisector. If we minimize (1) subject to the fifty degree constraints (that is, a copy of constraint (2) for each city $1 \le j \le 50$), we get the "solution" displayed in Figure 3(b). In this "solution" each city is adjacent to two others, but there are six sets of cities that form subtours:

$$\begin{aligned}
S_1 &= \{13,19,25\},\\
S_2 &= \{8,24,43,46,47\},\\
S_3 &= \{5,12,14,15,30,33,35,36,44\},\\
S_4 &= \{6,9,16,21,29,39,40,41,45\},\\
S_5 &= \{7,10,17,18,20,22,23,34,37,50\}, \text{ and}\\
S_6 &= \{1,2,3,4,11,26,27,28,31,32,38,42,48,49\}.
\end{aligned}$$

Fortunately, subtours are easily broken. To break the subtour for $S_1 = \{13,19,25\}$, the triangle-shaped subtour that connects cities 13, 19, and 25, we can add the subtour elimination constraint

$$x_{13,19} + x_{13,25} + x_{19,25} \le 2.$$

This constraint prohibits us from using more than two of the edges $\{13,19\}$, $\{13,25\}$, and $\{19,25\}$ in the tour. If we use all three, the salesman will end up going from 13 to 19 to 25 and then back to 13, or from 13 to 25 to 19 and then back to 13, in either case a cycle of three edges. Similarly, to break the subtour $S_2 = \{8,24,43,46,47\}$, we can add the constraint

$$x_{8,24} + x_{8,43} + x_{8,46} + x_{8,47} + x_{24,43} + x_{24,46} + x_{24,47} + x_{43,46} + x_{43,47} + x_{46,47} \le 4.$$

And if we want to make sure that absolutely no subtours will appear, we can impose

$$\sum_{\substack{1 \le i < j \le n \\ i,j \in S}} x_{i,j} \le |S| - 1 \tag{3}$$

for each $S \subseteq \{1,\ldots,n\}$ such that $3 \le |S| \le n-3$. The only problem is that there are too many of these subtour elimination constraints—a total of $\binom{n}{3} + \binom{n}{4} + \cdots + \binom{n}{n-3} = 2^n - (n^2+n+2)$ of them—for this to be viable. When $n = 50$, as in the Figure 3(a) TSP, there are over 10^{15} subtour elimination constraints!

The best way to overcome this "subtour problem" is to add subtour elimination constraints only as needed. And typically, only a small number are needed. When we solved the Figure 3(a) TSP, using ILOG's CPLEX [5] to solve the integer programs, we added a total of 17 subtour elimination constraints. We added them in four stages. The stages are displayed in Figures 3(b) through 3(e), and the optimal tour is displayed in Figure 3(f).

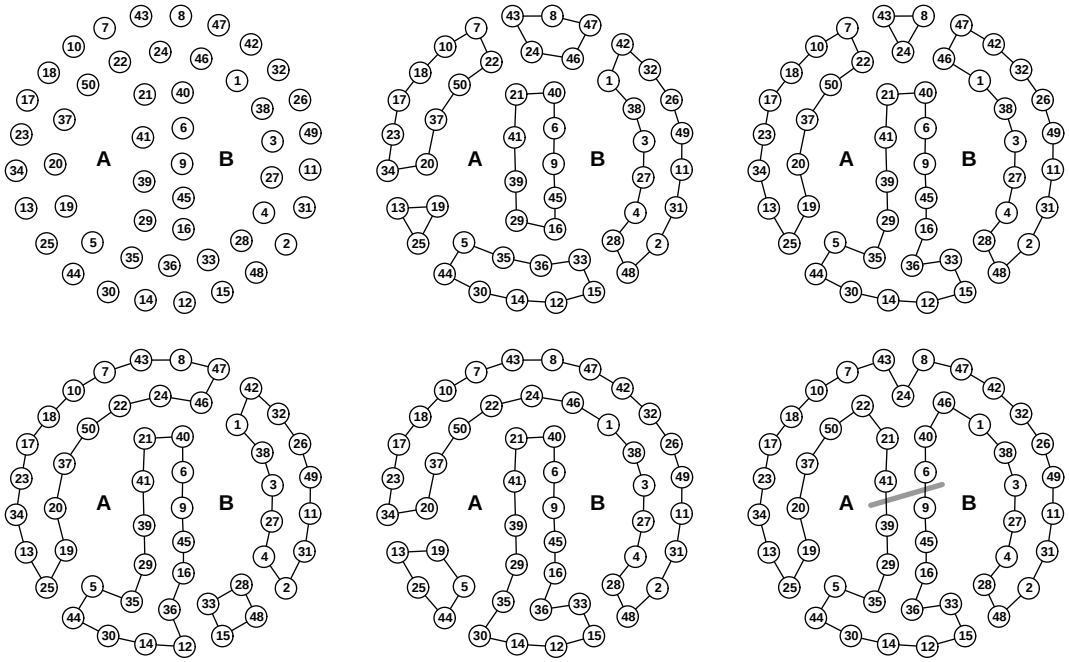

Figure 3: (a) a 50-city TSP, (b) six subtours, (c) four more subtours, (d) four more subtours, (e) three more subtours, (f) the optimal tour.

3 Controlling the Interior (and Exterior) of the Tour

A quick look at Figure 3(f) is all it takes for someone to determine whether or not A and B lie on the same side of the tour. But if the tour were more complicated, answering this question could be much more difficult.

There are two ways to determine whether or not regions A and B lie on the same side of the tour. Both involve drawing. One can be termed the "path method" and the other, the "crossing number" method.

In the path method, we try to draw a path from A to B that doesn't cross any edges of the tour. By doing this, we are trying to find a path through a maze which may or may not have a solution. For our small example, this is trivial. Larger more complicated tours require more work.

The crossing number method is less obvious, but it is much easier to execute. It has two steps. First, we draw a straight line segment $\ell_{A,B}$ that has one endpoint in A and the other in B. When drawing $\ell_{A,B}$, we must make sure that it doesn't pass through any cities. Second, we count the number of times that $\ell_{A,B}$ crosses edges of the tour. We call this number the crossing number for $\ell_{A,B}$, and we denote it $\nu(\ell_{A,B})$. If $\nu(\ell_{A,B})$ is odd, then A and B lie on opposite sides of the tour; if $\nu(\ell_{A,B})$ is even, then A and B lie on the same side.

In Figure 3(f), $\ell_{A,B}$ (drawn in light gray) crosses only edge $\{6,9\}$ and edge $\{39,41\}$, so $\nu(\ell_{A,B}) = 2$. Since the crossing number is even, A and B lie on the same side of the tour.

For any two regions A and B, there are infinitely many lines that can serve as $\ell_{A,B}$. The value of $\nu(\ell_{A,B})$ may not be the same for each of these lines. But the parity of $\nu(\ell_{A,B})$ will not change. It will be same for all lines that have one endpoint in A and the other in B.

We can use the crossing number method to create simple linear constraints that can be used as side constraints in the DFJ formulation of the TSP. Suppose we want regions A and B to lie on opposite sides of the tour. We begin by drawing a line segment $\ell_{A,B}$ with endpoints in A and B. We then construct the set of all edges $\{i, j\}$ that cross $\ell_{A,B}$. We denote this set $\chi(\ell_{A,B})$. Finally, we introduce an nonnegative integer variable

$y_{\ell_{A,B}}$ and add the constraint

$$\sum_{\{i,j\}\in\chi(\ell_{A,B})} x_{i,j} = 2y_{\ell_{A,B}} + 1 \tag{4}$$

to the DFJ formulation. Equation (4) forces $\nu(\ell_{A,B})$ to be odd; the lefthand side equals $\nu(\ell_{A,B})$, and since $y_{\ell_{A,B}}$ is a nonnegative integer, the righthand side will definitely be odd.

Note that if we want regions A and B to lie on the same side of the tour, we can use a modified version of equation (4),

$$\sum_{\{i,j\}\in\chi(\ell_{A,B})} x_{i,j} = 2y_{\ell_{A,B}}. \tag{5}$$

In addition, note that we can use equation (4) to force a region A to be in the interior of the tour and equation (5) to force a region A to be in the exterior of the tour. In each case, we let region B be the exterior.

With a single constraint of type (4), we were able to find a tour (Figure 4(a)) that has A on one side and B on the the other. We needed to add a total of nine subtour elimination constraints in three stages. And with one constraint of type (5) and one constraint of type (4), we were able to find a tour (Figure 4(b)) that has both A and and B in the interior. We needed only one stage and three subtour elimination constraints.

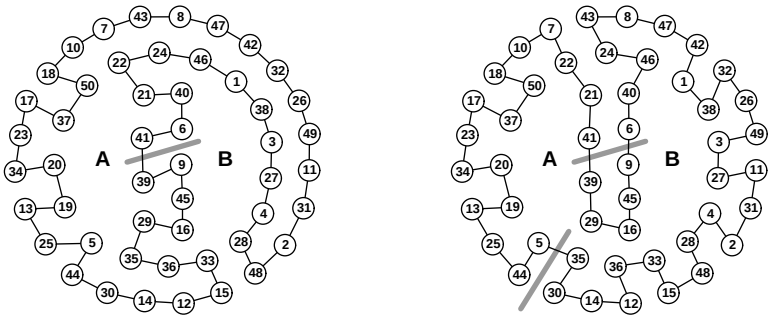

Figure 4: (a) regions A and B lie on opposite sides of the tour, (b) regions A and B lie in the tour's interior.

4 Examples

To date, we have focused our efforts on using the DFJ formulation with crossing number side constraints to produce TSP Art renditions of various knots and two-component links (Figures 2(c), 5(b), 5(c), 6(b), 6(c), and 7(a-d)).

Figure 5(a) shows a 500-city TSP that resembles a trefoil knot. Figures 5(b) and 5(c) display tours that keep the trefoil knot in the interior and exterior, respectively. Figure 5(d) gives, for comparison, the optimal tour found by Concorde [3], a state-of-the-art TSP Solver which, though a truly wonderful package, doesn't allow users to solve TSPs that have side constraints.

Figure 6 shows a 1200-city TSP that resembles a two-component link in which a ring (the unknot) weaves its way in and out of a trefoil knot. Figures 6(b) and 6(c) display tours that keep the ring in the exterior and interior, respectively. Figure 6(d) gives, for comparison, the optimal (but flawed) tour found by Concorde.

Figure 7 is a gallery of additional images.

5 Final Comments

The method described here produces beautiful images, but it requires a great deal of computer time (hours of CPU time for each of the Figure 6(b,c) and Figure 7 images), even when we give up on the goal of solving

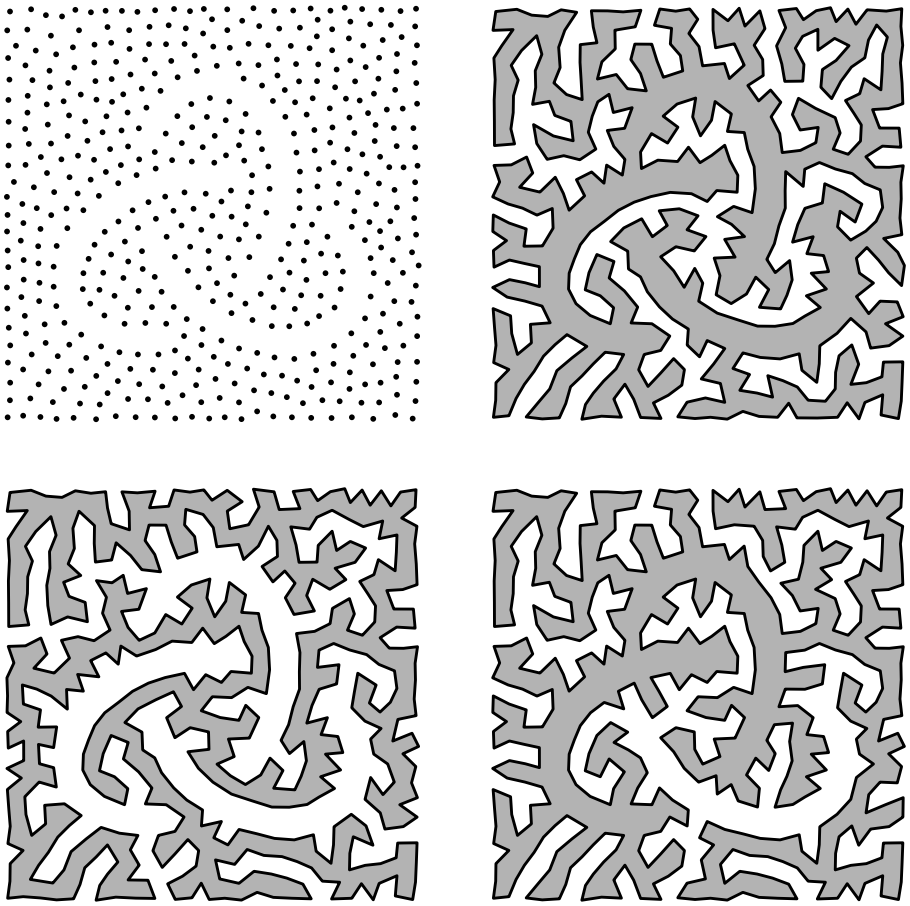

Figure 5: (a) 500-city trefoil-knot TSP, (b) a tour that keeps the trefoil knot in its interior, (c) a tour that keeps the trefoil knot in its exterior, (d) the optimal (but flawed) tour found by Concorde.

each stage to optimality and instead stop work on a stage whenever we find a tour within, say, one percent of optimality. Perhaps a better approach would be to design (or modify existing) TSP heuristics that could handle the crossing number side constraints.

References

[1] David L. Applegate, Robert E. Bixby, Vasek Chvátal, and William J. Cook. *The Traveling Salesman Problem: A Computational Study.* Princeton University Press, 2006.

[2] Robert Bosch and Adrianne Herman. Continuous line drawing via the Traveling Salesman Problem. *Operations Reseach Letters*, 32(4):302-303, 2004.

[3] Concorde TSP Solver. `www.tsp.gatech.edu/concorde/index.html`.

[4] G. Dantzig, R. Fulkerson, and S. Johnson. Solution of a large-scale traveling-salesman problem. *Operations Reseach*, 2:393-410, 1954.

[5] ILOG CPLEX. High-performance software for mathematical programming and optimization. `www.ilog.com/products/cplex/`.

[6] Craig S. Kaplan and Robert Bosch. TSP Art. In *Rennaissance Banff: Bridges 2005: mathematical connections in art, music, and science*, pages 301-308. 2005.

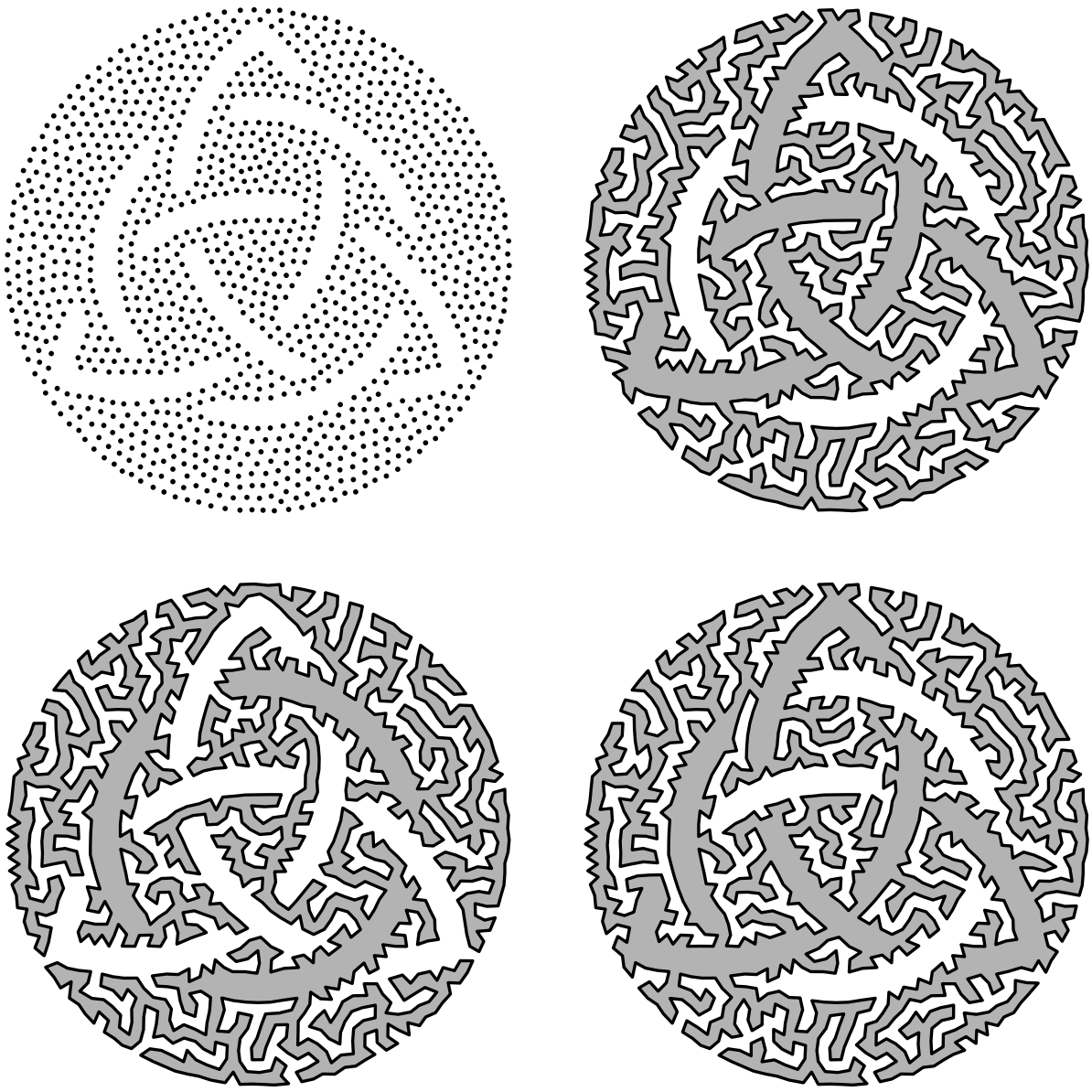

Figure 6: (a) 1200-city ringed-trefoil TSP, (b) a tour that keeps the ring outside and the trefoil inside, (c) a tour that keeps the ring inside and the trefoil outside, (d) the optimal (but flawed) tour found by Concorde.

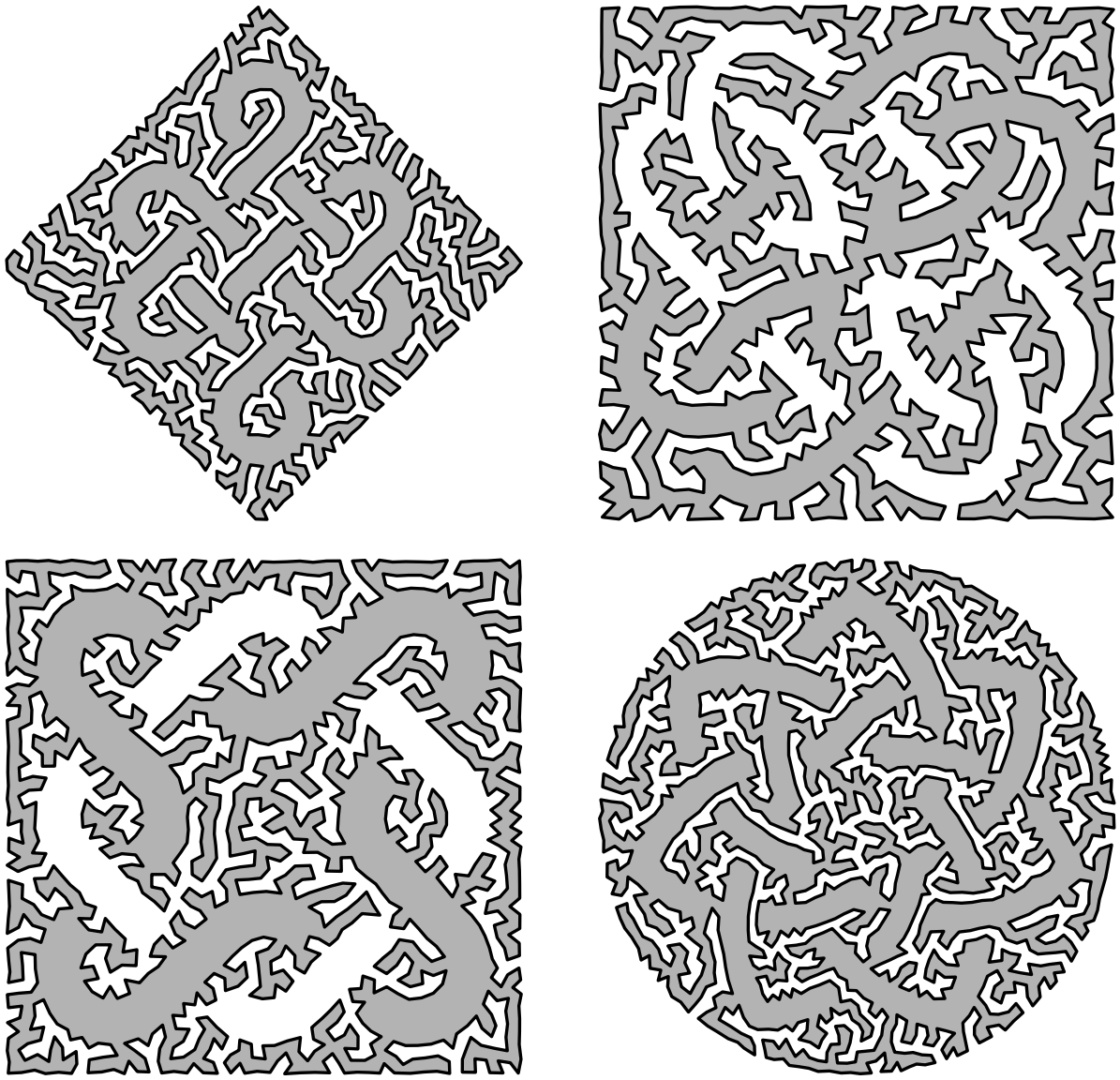

Figure 7: A gallery of TSP Art produced using the DFJ formulation with crossing number side constraints: (a) 1000 cities, (b) 1000 cities, (c) 1200 cities, (d) 1500 cities.