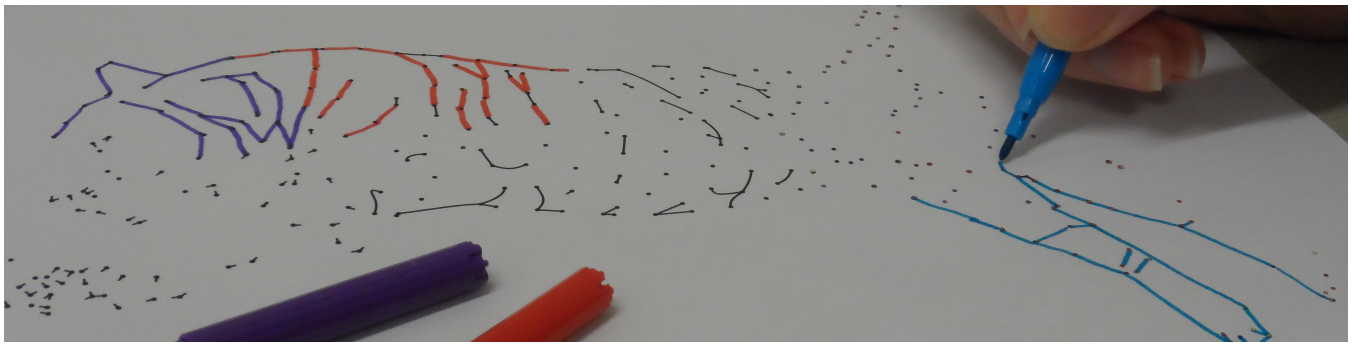
The Connect-The-Dots Family of Puzzles: Design and Automatic Generation

Maarten Löffler* Mira Kaiser† Tim van Kapel‡ Gerwin Klappe§ Marc van Kreveld¶ Frank Staals‖



Abstract

In this paper we introduce several innovative variants on the classic Connect-The-Dots puzzle. We study the underlying geometric principles and investigate methods for the automatic generation of high-quality puzzles from line drawings.

Specifically, we introduce three new variants of the classic Connect-The-Dots puzzle. These new variants use different rules for drawing connections, and have several advantages: no need for printed numbers in the puzzle (which look ugly in the final drawing), and perhaps more challenging "game play", making the puzzles suitable for different age groups. We study the rules of all four variants in the family, and design principles describing what makes a good puzzle. We identify general principles that apply across the different variants, as well as specific implementations of those principles in the different variants. We make these mathematically precise in the form of criteria a puzzle should satisfy.

Furthermore, we investigate methods for the automatic generation of puzzles from a plane graph that describes the input drawing. We show that the problem of generating a good puzzle –one satisfying the mentioned criteria– is computationally hard, and present several heuristic algorithms.

Using our implementation for generating puzzles, we evaluate the quality of the resulting puzzles with respect to two parameters: one for similarity to the original line drawing, and one for ambiguity; i.e. what is the visual accuracy needed to solve the puzzle.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric algorithms, languages, and systems F.2.2 [Analysis of Algorithms and Problem Complexity]: Nonnumerical Algorithms and Problems—Geometrical problems and computations;

Keywords: pencil-and-paper puzzles, geometry, algorithms

*m.loffler@uu.nl
†m.kaiser@students.uu.nl
‡timvankapel@gmail.com
§gerwin.klappe@gmail.com
¶m.j.vankreveld@uu.nl
‖f.staals@uu.nl

Links: DL PDF

1 Introduction

Point puzzles are puzzles where points are given and a drawing should be made according to certain rules. The best known type is called *Connect-The-Dots* or *Follow-The-Dots*, where points are labeled with a number in a range $1, \ldots, n$, and the rule is to connect points with consecutive numbers (see Figure 1). As a consequence, the puzzle can be solved by drawing one polygonal line with $1, 2, \ldots, n$ as the vertices. Many drawings cannot be drawn with a single polygonal line, so Connect-The-Dots puzzles often have parts pre-drawn. Furthermore, the labels clutter the final drawing. We present new point puzzle types and variations that do not suffer from these drawbacks.

Instead of connecting points based on their labels, our new point puzzles are purely based on a geometric rule. Examples of such a rule are: connect a point to the closest other point, connect to the point in the given direction, or connect any two points if they are at distance 1. For all these puzzles, we would still like that a user can find the solution without the use of a ruler or an other measurement device. Hence, ambiguity becomes an important aspect when designing such puzzles. Since this is difficult to deal with by hand, we formalize ambiguity criteria for point puzzles, and provide algorithms to generate a puzzle given the drawing that the solution should resemble.

Connect-the-dots puzzles are not only fun to solve, they also help develop cognitive skills in young children by exposing them to sequencial numbers. While this particular feature disappears when we remove the labels, we expect that our new geometric variants will have similar educational benefits. Solving the puzzles requires estimation and comparison of distances, directions, and colors.

Related Work. Our basic problem is converting a line drawing (a planar straight-line graph) into a set of points so that the solution is a drawing that looks like the original line drawing. This is similar to the problem of line simplification. Typically, such problems are solved by algorithms such as Douglas-Peucker [1973] or Imai-Iri [1988]. Generally, a drawing consists of more than one curve. Instead, it forms a planar subdivision. Subdivision simplification is much harder; certain versions are even NP-hard [Estkowski and Mitchell 2001; Guibas et al. 1993]. A practical solution to simplify subdivisions is to consider maximal parts of a subdivision that are polygonal lines, and apply line simplification to each separately.

The puzzler performs reconstruction of a shape based on a set of points. This problem is well-known and has been studied as an algorithmic problem for many years [Amenta et al. 1998; Dey et al. 2000; O'Rourke et al. 1987], including the 3D variants of surface reconstruction from point clouds (e.g. [Hoppe et al. 1992]). The generation of points from a line drawing to make a puzzle is a form of sampling, and therefore the inverse of reconstruction from a sample. Our sampling version has different objectives than the more common sampling applications in graphics, where it may be used for anti-aliasing or converting a continuous signal into a discrete one.

Our work has relations to certain research in graph theory and graph drawing. One of the puzzle types is related to unit-distance graphs and matchstick graphs [Harborth 1994], another puzzle type is related to nearest neighbor graphs [Paterson and Yao 1992], and a third puzzle type is related to partially drawn links when displaying graphs [Burch et al. 2011].

Since we introduce three new point puzzle types, no earlier research exists on them, but there are several papers discussing the automated generation of mazes [Xu and Kaplan 2007] and other pen-and-paper puzzles [Yoon et al. 2008; Jin et al. 2013; Ortíz-García et al. 2007]. More generally, automated content generation for puzzle games and other games has received attention in digital games research [Browne 2011; Colton 2002; Hendrikx et al. 2013].

2 The Connect-The-Dots Family

This section describes a number of point puzzles, beginning with the well-known Connect-The-Dots puzzle and some variations. After this we introduce three new point puzzle types that do not require numbers. Rules to draw an edge (line segment) may depend on direction indicators, nearest neighbors, or unit distances.

2.1 Classic Connect-The-Dots

The standard Connect-The-Dots puzzle has already been described. To alleviate the problem that only a single polygonal line can be drawn to solve the puzzle, we allow interruptions by using two point symbols instead of just one. If the label of a dot is a normal number, then the edge to the next higher-numbered point should be drawn, but if the label has a "+" appended to it, then the next edge should not be drawn. In this way the puzzler draws several polygonal lines, see Figure 1.

Other variations that allow multiple curves can give each polygonal line in the drawing its own color or symbol, and use the sequence of labels $1, 2, \ldots$ for each color or symbol.

2.2 Connect-That-Dot

Instead of labeling each point, we can extend the point symbols with a small link that shows the direction in which an edge should be drawn. The puzzler extends this link until another point is reached, see Figure 2. Points can have more than one link attached, and then we draw more edges from the point. The puzzle type can be such that each edge to be drawn has links at both endpoints, or there is a link at exactly one of the endpoints. We call these types *two-sided* and *one-sided*.

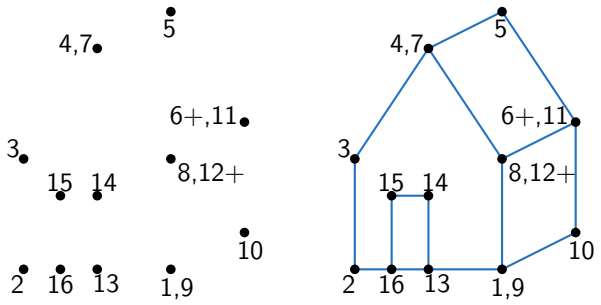


Figure 1: *A classical Connect-The-Dots puzzle and its solution.*

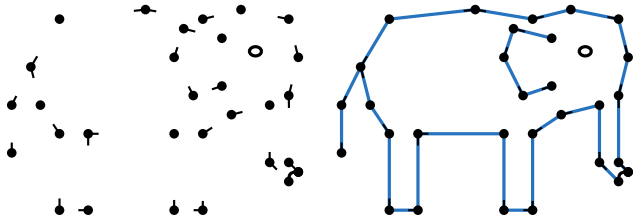


Figure 2: *A Connect-That-Dot puzzle with one-sided links and its solution.*

2.3 Connect-The-Closest-Dot

In Connect-The-Closest-Dot puzzles, every point must be connected to its nearest neighbor. So, for any edge to be drawn by a puzzler, at least one of its endpoints must have the other endpoint as a closest point. We will require that each point has a unique nearest neighbor.

Since the nearest-neighbor graph of n points has $n - 1$ or fewer edges, only very simple drawings are possible. To allow for more complex ones, we assign each point a color. The nearest-neighbor rule is now applied only for points of the same color. See Figure 3. When two or more differently colored points should be at the same location, a pie-chart like point can be used. We connect such a point to the nearest point of each color that it represents.

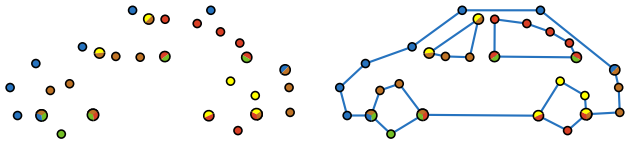


Figure 3: *A Connect-The-Closest-Dot puzzle and its solution.*

2.4 Connect-The-Unit-Dots

In Connect-The-Unit-Dots puzzles we connect two points if and only if they lie at a unit distance. The exact value of the unit is not important because a point set that is good as a puzzle for one distance can be scaled to make it good for another distance. From the puzzler's perspective, it may make sense to choose the distance

equal to the diameter of a small coin, so that the coin can be slid over the points to decide which ones should be connected.

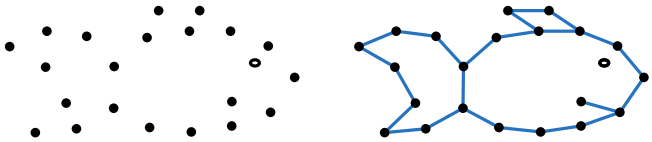


Figure 4: *A Connect-The-Unit-Dots puzzle and its solution.*

A variation of the unit-distance rule is to use two or more distances, that is, connect two points if their distance is either one centimeter, or three centimeter. This can be done with points of one color, or with different colors where each color is associated with one distance.

When certain parts of a line drawing cannot be represented well by points at unit distances, we pre-draw those parts. This possibility exists for all other point puzzle types as well. Obviously, a good puzzle should have only few pieces pre-drawn.

3 Mathematical Model

We would like to automate the process of generating a puzzle from a line drawing, assumed to be given as an embedded planar straight-line graph. Typically, such a graph has vertices of various degrees, where vertices of degree 1 and 3 and higher are *structural vertices* and vertices of degree 2 determine shape. The structural vertices partition the edges of the graph into a number of edge sequences that are polygonal lines containing the degree 2 vertices. In such a partition, a vertex of degree d, with $d \in \{1, 3, 4, 5, \ldots\}$, is the endpoint of d polygonal lines. We call this partition the *stroke decomposition*, and the resulting polygonal lines are called *strokes*.

In all puzzle types, the main task is to determine where to place points so that the solution of the puzzle is similar to the original drawing, but the original drawing is not immediately apparent in the puzzle. Depending on the puzzle type we must also decide what colors to use for which points, or on which side to place a directional link.

3.1 General guidelines

We now discuss general guidelines for good point puzzles. Specific point puzzle types have additional guidelines that we will discuss later. For each puzzle, we require that:

- The solution to the point puzzle should be similar to the original drawing.
- The puzzle should not have any two points too close together.
- The puzzle solution should not have intersections that the original drawing did not have.
- The puzzle should not be ambiguous: visual inspection should make clear what point pairs are to be connected.
- The puzzle should obscure the original drawing.

Next, we formalize the guidelines above. The general idea is to specify the first four guidelines as constraints, defining whether or not a puzzle is valid, and to express the last guideline as an optimization function.

To make sure that the solved puzzle and the original drawing are similar, we require that all structural vertices of the drawing are structural vertices in the solution as well. This means that they should either lie on a pre-drawn piece of the puzzle, or are represented by points. That is, they cannot be missing in the puzzle. Furthermore, for every stroke in the drawing there must be a stroke in the solution that is sufficiently close, using a small distance value ε. In particular, for each point on a stroke in the drawing there must be a point on the corresponding stroke in the solution that is within distance ε.

To decide whether two points of a puzzle are too close together we use a simple distance threshold value, denoted by δ.

We will assume that the original drawing does not have intersections; it is a planar drawing of a graph. Therefore, we do not allow the puzzle solution to have any intersections either.

To decide whether the puzzle is ambiguous we must differentiate on puzzle type; we cannot specify ambiguity in general. In the next section, we define for each type what it means for a puzzle to be ambiguous. In all cases, we express the maximum allowed ambiguity in terms of a parameter γ. We refer to this as the *tolerance* of the puzzle.

Finally, we need to make sure that the original drawing is not directly visible in the puzzle. Small initial experiments indicated that there are two main aspects to this: (i) the length and number of pre-drawn pieces in the puzzle, and (ii) the number of points in the puzzle. We express (i) as another criterion, so we can optimize for the number of points in the puzzle. Hence, we require that the length and/or the number of the pre-drawn pieces is at most λ.

3.2 Ambiguity

Connect-The-Dots. The classic Connect-The-Dots puzzle completely specifies which pairs of points are to be connected. The puzzler does not need to estimate distances or directions. We note that ambiguity may arise if the labels of the points are not well placed; this relates to the cartographic map labeling problem (e.g. [Christensen et al. 1995]). Map labeling has been well-studied for many years and we will not discuss it further in this paper.

Connect-That-Dot. In a one-sided Connect-That-Dot puzzle two points should be connected if one has a small link attached that is directed to the other point. Every link is the beginning of an edge to be drawn. Ambiguity may arise if there are several points in roughly the same direction. It is natural to draw the edge to the closest point that is in the right direction. Therefore, if two points p, q should be connected using a link at p pointing towards q, then there is no ambiguity if a certain region based on angular cones at p and q is empty of other points, see Figure 5 (left and middle). In the two-sided case it is natural to have a symmetric region, and we require that any edge is non-ambiguous regardless of the point and link we take as the start (Figure 5 (right)). The angles used to construct the region for points p and q form the tolerance of this puzzle.

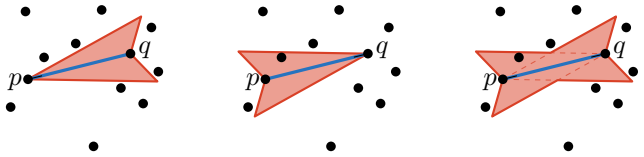


Figure 5: *(left) A valid one-sided puzzle in which we will connect p to q using a link at p. The puzzle is invalid if q has a link pointing to p (middle and right).*

Connect-The-Closest-Dot. To avoid ambiguity in a Connect-The-Closest-Dot puzzle we require that for every point p with color c, the difference in distance between the nearest point to p with

color c, and the second-nearest point to p with color c is at least γ, the tolerance. Hence, if we have to connect p to q, then p and q have to be the only points with color c in the disk of radius $\|pq\|+\gamma$ centered at p (see Figure 6).

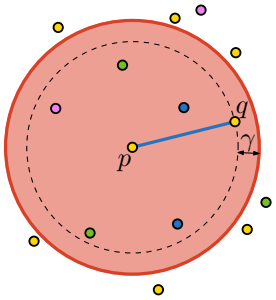


Figure 6: *If there is an edge from p to q then p and q are the only yellow points in the red disk of radius* $\|pq\| + \gamma$.

Connect-The-Unit-Dots For Connect-The-Unit-Dots puzzles we use a similar criterion as for the Connect-The-Closest-Dot puzzles. Ambiguity may arise when two points lie at nearly unit distance, so we require that no pair of points lie at a distance in the union of open intervals $(1-\gamma, 1) \cup (1, 1+\gamma)$, for some $0 < \gamma < 1$. See Figure 7.

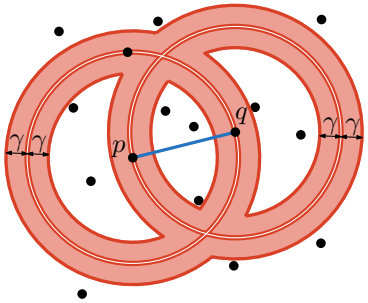


Figure 7: *There may be no pairs of points with a distance in* $(1-\gamma, 1) \cup (1, 1+\gamma)$*, hence the red region should be empty.*

4 Automatic Generation

In this section we investigate how to automatically generate a point puzzle, starting with a drawing that the solution should look like. More formally, we are given a straight line plane graph D representing the drawing, and parameters ε, δ, γ, and λ. We now wish to generate a puzzle resembling D that satisfies the criteria from the previous section. However, unfortunately even testing if there exists a puzzle satisfying (just) requirements one and two is NP-hard:

Theorem 1. *Given a plane graph D, it is NP-hard to determine if there exists a plane straight-edge graph G such that the minimum distance between any two vertices in G is at least δ, and the Hausdorff distance between (the embeddings of) D and G is at most ε.*

It follows from Theorem 1 that there are no efficient algorithms to generate Connect-The-Dots, Connect-That-Dot, or Connect-The-Closest-Dot puzzles. Similarly, we can show that computing Connect-The-Unit-Dots puzzles is NP-hard as well. Both proofs can be found in the Technical Supplement.

We now present several heuristic algorithms to generate puzzles. We focus on the case $\lambda = 0$, that is, we do not allow the algorithms to pre-draw anything. For all but the Connect-The-Unit-Dots puzzles this means that we can compute puzzles by finding a minimum size set of points that satisfies (most of) the criteria. To do this we use variations of the line simplification algorithm by Imai and Iri [1988]. For the Connect-The-Unit-Dots puzzles we cannot avoid pre-drawing part of the input drawing, thus we use a different approach that we describe in more detail later. Before we continue, we briefly review the line simplification algorithm by Imai and Iri.

The Imai-Iri algorithm simplifies a polygonal line $L = (v_1, \ldots, v_n)$ by replacing parts of L by single line segments, called *shortcuts*, between two vertices of L. A shortcut $\overline{v_i v_j}$ is valid if $i < j$ and the part of L between v_i and v_j does not deviate more than ε from the line segment between v_i and v_j. Let H be the graph with $v_1, \ldots, v_n$ as the vertices and an edge (v_i, v_j) for every valid shortcut $\overline{v_i v_j}$. A shortest path in H from v_1 to v_n gives a minimum vertex simplification of H under the restriction that the deviation is at most ε. The algorithm runs in $O(n^2)$ time, if implemented well [Chan and Chin 1996].

To generate the points of the puzzle, we use an Imai-Iri style algorithm on each stroke. Depending on the type of puzzle that we wish to generate, we modify what it means for a shortcut to be valid. For example, the basic algorithm does not guarantee that the simplification of L does not have self-intersections. Once we have chosen points for each stroke, we combine them and assign labels, colors, or links to generate the puzzle.

4.1 Classic Connect-The-Dots

Recall that for the puzzle to be valid we need that (i) for each point on a stroke S in the drawing, there must be a point on S in the solution that is within distance ε, (ii) the solution does not have any intersections, and (iii) the distances between all pairs of points are at least δ. Thus, a shortcut $\overline{v_i v_j}$, with $i < j$, on stroke $v_1, .., v_n$ is valid if (and only if) it satisfies the following conditions.

For requirement (i) we use the standard requirement that all vertices $v_{i+1}, \ldots, v_{j-1}$ lie within distance ε from $\overline{v_i v_j}$.

For requirement (ii) we consider the bounded region(s) created by $\overline{v_i v_j}$ together with the part of the stroke $v_i, v_{i+1}, \ldots, v_j$. We reject a shortcut if any bounded region contains any vertex from the input (other than $v_i, v_{i+1}, \ldots, v_j$) inside it. This adaptation works well in practice, but it may happen that we reject a shortcut that is needed in a minimum vertex simplification. Hence, the shortest path in the graph is not guaranteed to be optimal for the overall problem.

For requirement (iii) we could also reject every shortcut that is too short. This is not sufficient, however, because it need not be consecutive points that are too close. Furthermore, enforcing this condition could make the problem unsolvable. Therefore, we will not explicitly enforce this requirement.

Often, there are several minimum-link simplifications that the (adapted) Imai-Iri algorithm could return. We choose to return the one that has largest Euclidean length, since this has a positive effect on the shape fidelity. The adaptation to be made to the Imai-Iri algorithm is simple: instead of assigning the weight 1 to every shortcut, we assign the pair of weights (1, −[Euclidean length]), and compute a lexicographically shortest path. The same idea can be used to avoid shortcuts with a length below δ: we let their secondary weight be large, causing the shortest path algorithm to avoid them.

Once we have computed all simplifications, we can create the puzzle. Vertices of degree 3 and higher need more than one label. To minimize the number of times this happens, we can partition the graph edges into a minimum number of paths. The standard algorithm for finding an Eulerian path in a graph can easily be extended

to achieve this.

Testing requirement (ii) for all shortcuts efficiently can be done using basic computational geometry methods, leading to a worst-case running time of $O(n^2 \log n)$ [de Berg et al. 1998]. While this seems slow, the practical running time is much better. For example, if shortcuts do not skip more than 100 vertices, the running time is only $O(n \log n)$ in the worst case. Furthermore, we do not need interactivity, we will not have very large values of n, and spatial indexing structures will also improve the running time in practice.

4.2 Connect-That-Dot

For (one-sided) Connect-That-Dot puzzles we have the same general requirements as for the classic Connect-The-Dots puzzles. Additionally, we require that if we place a link at p pointing to q, the cone-shaped region starting at p should be empty (Figure 5). So, when deciding if the shortcut $\overline{v_i v_j}$ is valid, we can test the two cone-shaped regions for containment of any vertex of the input drawing, which is a superset of the points to be selected for the puzzle. If we find out that both regions are non-empty, then we may not be able to assign a link to either endpoint. So we deem the shortcut invalid and discard it.

This adaptation ensures that the Imai-Iri algorithm will always return a set of points for a stroke for which links can be assigned. Algorithmically we can implement this adaptation to run in $O(n^2 \log^2 n)$ time overall. We compute all shortcuts leaving a particular vertex v_i in $O(n \log^2 n)$ time by using a dynamic data structure that can check cone-region emptiness on the right subset of the points in $O(\log^2 n)$ time and with $O(\log^2 n)$ update time, as explained briefly next.

Let v_i be fixed. We sort all vertices of the input except for v_i by angular order around v_i. We use this angular order to store these points in a balanced binary search tree. Each internal node represents an angular interval and a subset of the points. This subset is stored in an associated structure that supports half-plane emptiness queries, such as a dynamic convex hull [Brodal and Jacob 2002]. The tree with associated structures has size $O(n \log n)$ and allows queries to decide if the cone region is empty of points in $O(\log^2 n)$ time.

When computing the valid shortcuts from v_i, we process the points $v_{i+1}, \ldots$ incrementally. Suppose we are at vertex v_j. We first test the ϵ-condition for the shortcut $\overline{v_i v_j}$ in the usual way and the cone-emptiness condition by a query, and then we remove the point v_j from the data structure before proceeding to test v_{j+1}.

4.3 Connect-The-Closest-Dot

Connect-The-Closest-Dot puzzles are more complex to compute, although we can still use a variant on the Imai-Iri algorithm. Observe that within a single color, the drawing produced by the puzzler is a nearest-neighbor graph, in which each connected component is a tree that has one vertex such that distances increase along tree edges away from this vertex.

To compute puzzles we take the following sequence of steps. Firstly, we determine the stroke decomposition of the input drawing. Then, for each stroke, we densify the edges so that there are many vertices on the stroke. Thirdly, we compute the valid shortcuts as in the Imai-Iri algorithm, yielding a graph G. Fourthly, we turn G into a different graph H as follows. Every edge of G becomes a vertex in H. Two vertices w_s and w_t are connected by an edge in H if and only if w_s represents the directed edge (v_i, v_j) in G and w_t represents the directed edge (v_j, v_k) in G, and (v_j, v_k) is longer than (v_i, v_j) by at least the tolerance γ. A path in H represents a distance-increasing set of points on the stroke; when we say "distance-increasing" we implicitly assume that the increase uses at least the tolerance in every step.

To ensure that simplifications of the stroke up to v_k will be reconstructed correctly and unambiguously, we will require that all vertices of the stroke up to and including v_i lie outside the circle centered at v_k and with radius the distance to v_j plus the tolerance γ. If some point of $v_1, \ldots, v_i$ lies inside, then we discard the directed edge (w_s, w_t) from H.

We try different ways to turn a stroke into a group of points for a puzzle. Firstly, we can try to represent the stroke with a single color if a path in H exists from the one end of the stroke to the other end, or vice versa. We can also use just a single color if there is a point in the middle from which we have distance-increasing paths to both ends of the stroke. We also consider representing the stroke by two colors. In this case we compute distance-increasing paths from both ends of the stroke, and see if there is a vertex that we can reach from both ends. If a stroke is long, we may start computing a distance-increasing path until distances get too long or we can no longer extend it. Then we treat the remaining part of the stroke recursively with one or more colors. In all cases, a stroke yields one or more groups of points so that within one group, one color suffices.

We now have multiple ways for each stroke to represent it. We try all combinations brute-force. For each combination we need to find out which groups can receive the same color, and which groups are forced to use different colors. The fifth step of computing the puzzle consists of assigning specific colors to the groups of points, when we already know that within one group we can use a single color. This step comes down to a graph coloring problem on a graph where each group is a node, because conflicting groups—represented by an edge between the nodes—cannot receive the same color.

Our solution to this graph coloring problem is a heuristic that also attempts to use the same colors at vertices where several strokes meet. This reduces the number of colors per point symbol, and the number of points with multiple colors.

Eventually we choose the solution that uses fewest colors, and among these, the one that has fewest points. Among these, we choose the one with fewest colors per point symbol.

The worst-case running time of the heuristic is high, due to brute-force testing of all combinations of ways to represent strokes. We can afford this in practice because the number of strokes in a line drawing is typically small, and the number of ways to represent a single stroke is also not large.

4.4 Connect-The-Unit-Dots

When generating Connect-The-Unit-Dots puzzles we face problems of a different nature. For example, there may not be pairs of input vertices at unit distance at all. Secondly, it is usually impossible to represent a single stroke using points only: consider a stroke that is single line segment of non-integer length. Then we will always have a short piece left that must be pre-drawn.

Because of these issues we use a different approach. On the theoretical side, we present an algorithm that decides whether a Connect-The-Unit-Dots puzzle exists for an input drawing if every stroke can have at most one piece pre-drawn whose length must be less than unit. This algorithm runs in $O(n \log n)$ time, where n is the total length of the strokes in the input graph. Since the result of the decision problem on a drawing is often negative, and we still want to generate a puzzle, we develop a variant of the algorithm that optimizes the requirements of this puzzle type.

We start by identifying some properties of the points that model a single stroke S. Consider a point set $p_1, .., p_n$, all points on S and ordered along S. This point set *u-models* the substroke $S[p_1, p_n]$ from p_1 to p_n if and only if (i) every consecutive pair has distance exactly one, (ii) no other points are at distance one. It thus follows that, for all i, the entire substroke $S[p_{i-1}, p_{i+1}]$ is contained in the disk of radius one centered at p_i, and that the boundary of this disk contains exactly two points: p_{i-1} and p_{i+1}.

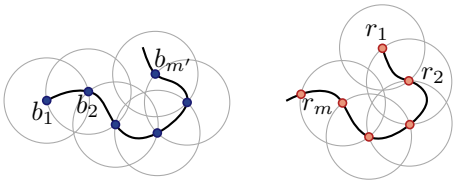


Figure 9: *A stroke with points chosen from the start or the end of the stroke.*

The crucial observation is now that there is a unique maximal point set that contains the start point p of S and u-models $S[p, r]$; the set of *red* points $p = r_1, .., r_m = r$. Similarly, there is a unique maximal point set that contains the end point q and u-models $S[b, q]$; the set of *blue* points $q = b_1, .., b_{m'} = b$. See Figure 9.

Since we require that structural vertices are present if they are not pre-drawn, and allow only one short pre-drawn segment per stroke, it follows that the set of points on each stroke is of the form $r_1, r_2, .., r_k, b_\ell, b_{\ell-1}.., b_1$, and that the pre-drawn segment connects the last red point r_k to the first blue point b_ℓ.

We can now decide whether there exists a point set P such that the solution to the corresponding Connect-The-Unit-Dots puzzle resembles a given input drawing and has at most one short pre-drawn segment per curve. We do this by constructing a 2-SAT formula that is satisfiable if and only if P exists. We generate the sets of red and blue points for each stroke, and represent each point by a boolean variable. The variable is set to TRUE if we include the point in the puzzle and to FALSE otherwise. We add clauses that force that all points on a stroke are of the form described above. Furthermore, for every pair of points we generate clauses expressing that their distance should be in the range $[\delta, 1-\gamma] \cup [1] \cup [1+\gamma, \infty)$.

This guarantees that the selected points satisfy the ambiguity and minimum-distance requirements from Section 3. The requirement that the (solution of the) resulting puzzle should be similar to the input drawing follows from the choice of unit-distance. We do not explicitly model the requirement that the solution is intersection free.

This approach allows us to solve the problem using 2-SAT in $O(n^2)$ time [Aspvall et al. 1979], where n is the total length of all curves (and also the number of points in a puzzle, if one exists). Using packing and algorithmic ideas we can improve the bound to $O(n \log n)$.

Our heuristic for the optimization variant solves a weighted MAX-2-SAT version. We use constraints that we want to enforce with weight ∞, and constraints that we prefer with a normal or low weight. The heuristic will pre-draw more parts if this is needed to satisfy the requirements, but maximization will lead to a small number of pieces that will be pre-drawn. Even though MAX-2-SAT is NP-hard in general, this approach still works well in practice.

5 Implementations, Results

We implemented the algorithms described in the previous section. In this section we show some generated puzzles, and present an evaluation of how the quality of the puzzles depends on parameters ε and γ.

Figures 8, 10, and 11 show generated puzzles for each of the four puzzle types. For these drawings, our generated puzzles are of a quality comparable to our own, manually designed ones.

Furthermore, we investigate the influence that the parameters ε and γ have on the quality of the puzzle. The results can be seen in Figures 19 and 20. In general, the results are as expected. When we increase ε, we allow the puzzle to deviate more from the input drawing. Thus, for all puzzles we see a decrease in the number of points in the solution.

For Connect-That-Dot puzzles we analyze the number of points in a puzzle and the percentage of pre-drawn length as functions of ε. Figure 13 shows the number of points per length unit, or, the point density. The black line shows the average of densities over 20 input

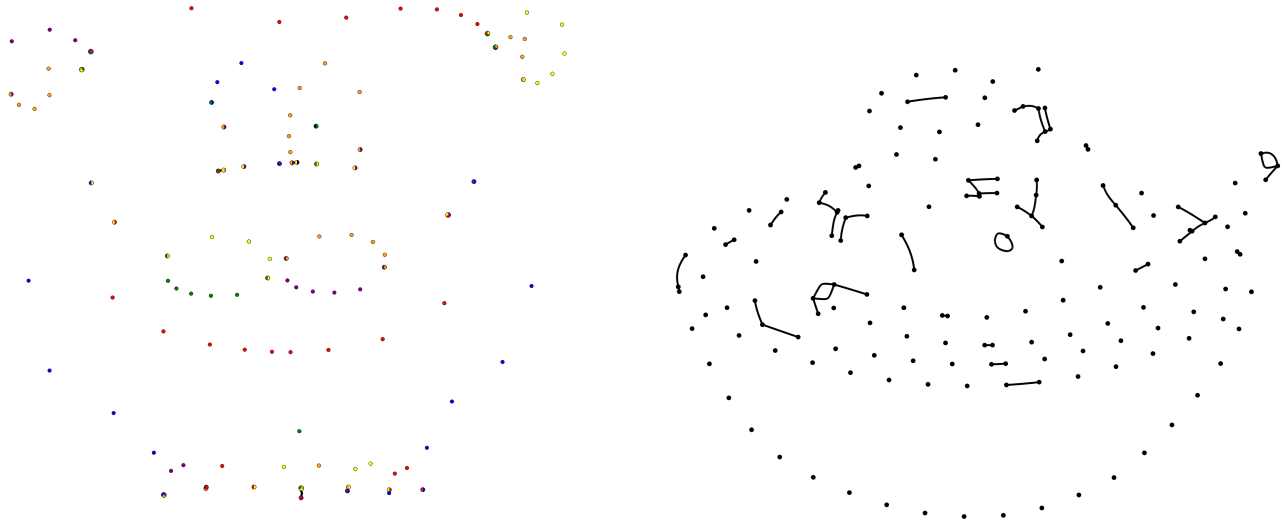


Figure 8: *Examples of generated puzzles. A Connect-The-Closest-Dot puzzle (left) and a Connect-The-Unit-Dots puzzle (right).*

Figure 10: *A generated Connect-The-Dots puzzle.*

drawings. The thin lines in the graph show the 20 drawings separately, and the green region indicates the standard deviation. The point density is up to twice as large for some puzzles as for others, for a value of ε. Within the shown range of ε that yields good puzzles, the smallest ε has about 50% more points than the largest ε. We also examined pre-drawn length as a function of ε, shown in Figure 14. There is no clear relation to be observed. On the average, less than 2% of the length of the input drawing is pre-drawn.

We produced similar graphs for a varying value of γ, the ambiguity parameter, and a fixed value of $\varepsilon = 15$. These are shown in Figures 15 and 16. Starting at $\gamma = 10^\circ$ the puzzle becomes solvable by hand without additional aids. We observe that the point density grows only slowly with γ, implying that we do not need many extra points to make the puzzle easy. Similarly, for most puzzles, less than 2% of the input drawing length need be pre-drawn for the interesting range of γ between 10° and 20°.

For Connect-The-Closest-Dot puzzles we are primarily interested in the number of colors needed in the puzzles. We show this in graphs with ε and γ on the horizontal axis in Figures 17 and 18. We observe that the number of colors is usually between 6 and 8, and depends at best mildly on ε. Only one puzzle required more than 10 colors, which is too many to be useful. We also observe that the number of colors needed grows slowly in γ; starting at $\gamma = 20\%$ the puzzles become solvable without aids. In other tests we observed that the pre-drawn length is typically slightly higher than for Connect-That-Dot puzzles, but still within acceptable limits for a good puzzle.

For the Connect-The-Unit-Dots puzzles, the input drawing becomes quickly recognizable for a small unit, or, equivalently, a high point density, while a large unit often results in many parts to be pre-drawn. At the same time, a value of 10% for ε is too small to solve the puzzle without mistakes, while a larger value quickly yields many pre-drawn parts. It is therefore more difficult to find parameter settings for good puzzles, and good puzzles may not exist for all input drawings. A more flexible approach may be needed in an implementation that consistently produces good puzzles.

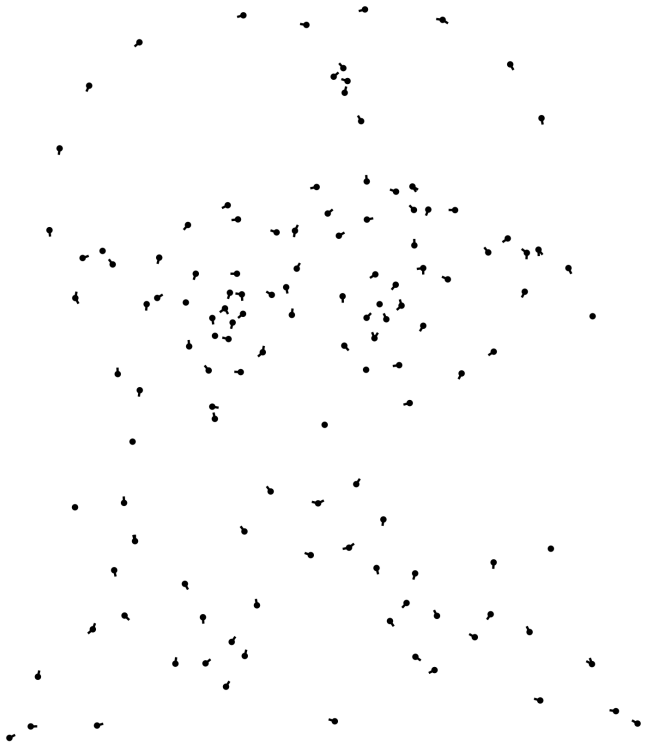


Figure 11: *A generated Connect-That-Dot puzzle.*

6 Conclusion

We introduced three new point puzzle types that are related to Connect-The-Dots puzzles. A geometric rule determines which pairs of points should be connected, rather than annotating points with labels. This allows for more interesting puzzles, and results in a cleaner final drawing. We identified several criteria for good point puzzles, and captured these in a mathematical model. We showed that for all our new puzzle types as well as the classical Connect-The-Dots puzzles, generating a good puzzle from a given input drawing is NP-hard. However, we presented heuristic algorithms that appear to work well in practice.

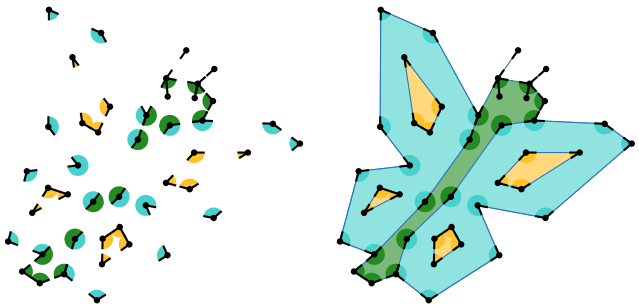


Figure 12: *A Connect-That-Dot puzzle with colored links forms a coloring page.*

An interesting remaining question is to determine how difficult a puzzle is. There are several aspects to this, for example how apparent the original drawing is, but also what ambiguity parameters still allow solving the puzzle without measurement devices. To answer these questions, we need user studies. Furthermore, an intriguing open question is to find a suitable quantification for how apparent the original drawing is in the puzzle. This may help in improving the mathematical model, and allow for better algorithms or heuristics to generate puzzles.

Other extensions to our work are other geometric rules describing which points to connect, and annotating the point puzzles with colors to produce coloring pages. See for example Figure 12. We presented our techniques to generate pencil-and-paper puzzles. However, our geometric rules also allow for digital variants. There are many additional options for digital variants. For example, we can use built-in measurement devices to allow for a smaller tolerance, and thus better looking puzzles.

Acknowledgements

This work has been partially supported by the Netherlands Organisation for Scientific Research (NWO) under grants 639.021.123 and 612.001.022.

References

AMENTA, N., BERN, M. W., AND EPPSTEIN, D. 1998. The crust and the beta-skeleton: Combinatorial curve reconstruction. *Graphical Models and Image Processing 60*, 2, 125–135.

ASPVALL, B., PLASS, M. F., AND TARJAN, R. E. 1979. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Inf. Process. Lett. 8*, 3, 121–123.

BRODAL, G. S., AND JACOB, R. 2002. Dynamic planar convex hull. In *Foundations of Computer Science, 2002. Proceedings. The 43rd Annual IEEE Symposium on*, IEEE, 617–626.

BROWNE, C. 2011. *Evolutionary Game Design*. SpringerBriefs in Computer Science. Springer.

BURCH, M., VEHLOW, C., KONEVTSOVA, N., AND WEISKOPF, D. 2011. Evaluating partially drawn links for directed graph edges. In *Graph Drawing - 19th International Symposium, GD 2011*, Springer, vol. 7034 of *Lecture Notes in Computer Science*, 226–237.

CHAN, W. S., AND CHIN, F. 1996. Approximation of polygonal curves with minimum number of line segments or minimum error. *Int. J. Comput. Geometry Appl. 6*, 1, 59–77.

CHRISTENSEN, J., MARKS, J., AND SHIEBER, S. 1995. An empirical study of algorithms for point-feature label placement. *ACM Trans. Graphics 14*, 3, 203–232.

COLTON, S. 2002. Automated puzzle generation. In *Proceedings of the AISB'02 Symposium on AI and Creativity in the Arts and Science*.

DE BERG, M., VAN KREVELD, M., AND SCHIRRA, S. 1998. Topologically correct subdivision simplification using the bandwidth criterion. *Cartography and Geographic Information Systems 25*, 243–257.

DEY, T. K., MEHLHORN, K., AND RAMOS, E. A. 2000. Curve reconstruction: Connecting dots with good reason. *Comput. Geom. 15*, 4, 229–244.

DOUGLAS, D. H., AND PEUCKER, T. K. 1973. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization 10*, 2, 112–122.

ESTKOWSKI, R., AND MITCHELL, J. S. B. 2001. Simplifying a polygonal subdivision while keeping it simple. In *Proc. 17th Annu. ACM Symposium on Computational Geometry*, 40–49.

GUIBAS, L. J., HERSHBERGER, J., MITCHELL, J. S. B., AND SNOEYINK, J. 1993. Approximating polygons and subdivisions with minimum link paths. *Int. J. Comput. Geometry Appl. 3*, 4, 383–415.

HARBORTH, H. 1994. Match sticks in the plane. In *The Lighter Side of Mathematics: Proceedings of the Eugéne Strens Memorial Conference of Recreational Mathematics and its History*, R. K. Guy and R. E. Woodrow, Eds. Mathematical Association of America, Washington DC, 281288.

HENDRIKX, M., MEIJER, S., VELDEN, J. V. D., AND IOSUP, A. 2013. Procedural content generation for games: A survey. *ACM Transactions on Multimedia Computing, Communications, and Applications 9*, 1, 1–24.

HOPPE, H., DEROSE, T., DUCHAMP, T., MCDONALD, J. A., AND STUETZLE, W. 1992. Surface reconstruction from unorganized points. In *SIGGRAPH*, ACM, J. J. Thomas, Ed., 71–78.

IMAI, H., AND IRI, M. 1988. Polygonal approximations of a curve formulations and algorithms. In *Computational Morphology*, Elsevier Science, 71–86.

JIN, J.-H., SHIN, H. J., AND CHOI, J.-J. 2013. SPOID: a system to produce spot-the-difference puzzle images with difficulty. *The Visual Computer 29*, 6-8, 481–489.

O'ROURKE, J., BOOTH, H., AND WASHINGTON, R. 1987. Connect-the-dots: A new heuristic. *Computer Vision, Graphics, and Image Processing 39*, 258–266.

ORTÍZ-GARCÍA, E. G., SALCEDO-SANZ, S., LEIVA-MURILLO, J. M., PÉREZ-BELLIDO, Á. M., AND PORTILLA-FIGUERAS, J. A. 2007. Automated generation and visualization of picture-logic puzzles. *Computers & Graphics 31*, 5, 750–760.

PATERSON, M., AND YAO, F. F. 1992. On nearest-neighbor graphs. In *ICALP*, Springer, W. Kuich, Ed., vol. 623 of *Lecture Notes in Computer Science*, 416–426.

XU, J., AND KAPLAN, C. S. 2007. Image-guided maze construction. *ACM Trans. Graph. 26*, 3, 29.

YOON, J.-C., LEE, I.-K., AND KANG, H. 2008. A hidden-picture puzzles generator. *Comput. Graph. Forum 27*, 7, 1869–1877.

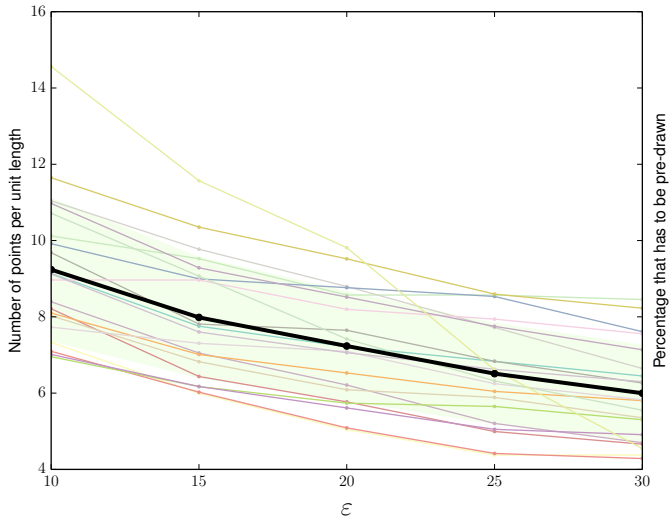


Figure 13: *Dependency of the point density in Connect-That-Dot puzzles on ε.*

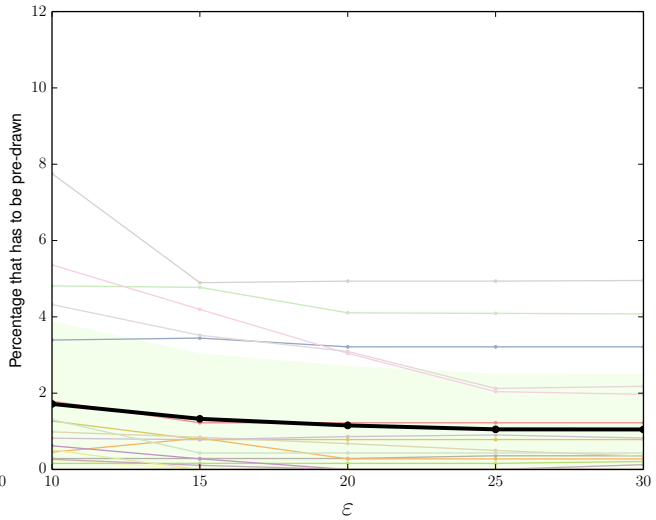


Figure 14: *Percentage of the pre-drawn length in Connect-That-Dot puzzles as a function of ε.*

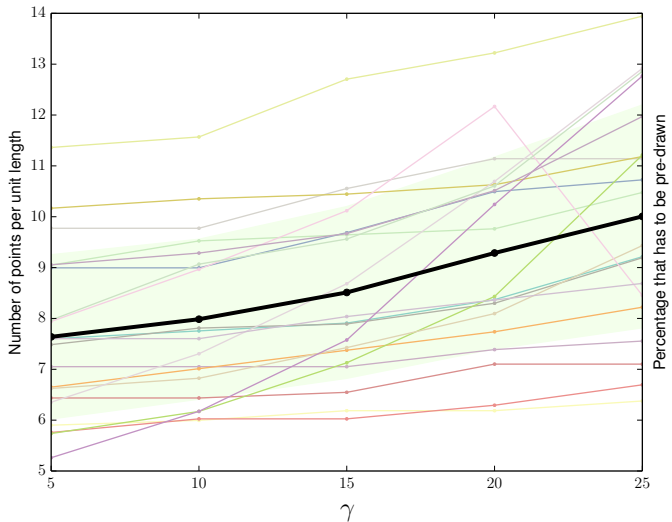


Figure 15: *Dependency of the point density in Connect-That-Dot puzzles on γ.*

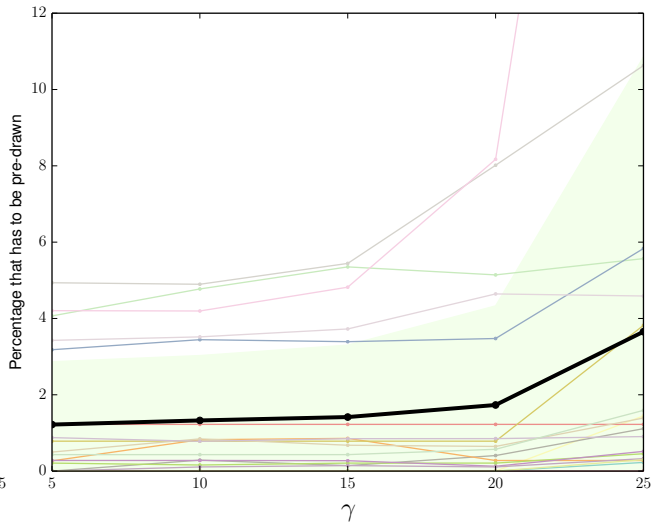


Figure 16: *Percentage of the pre-drawn length in Connect-That-Dot puzzles as a function of γ.*

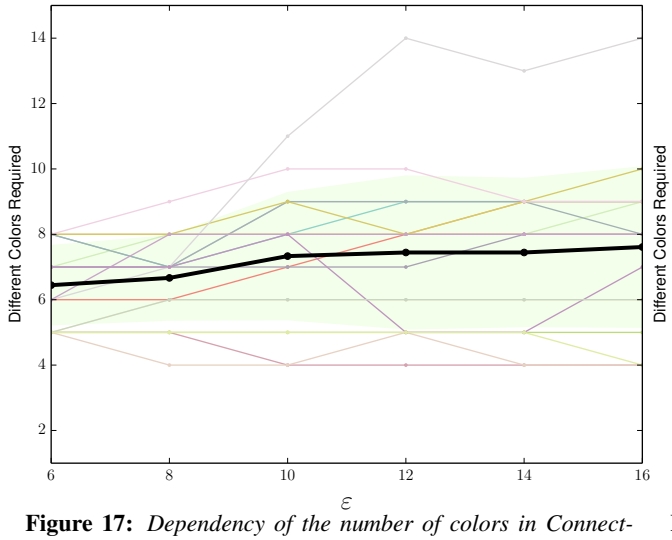


Figure 17: *Dependency of the number of colors in Connect-The-Closest-Dot puzzles on ε.*

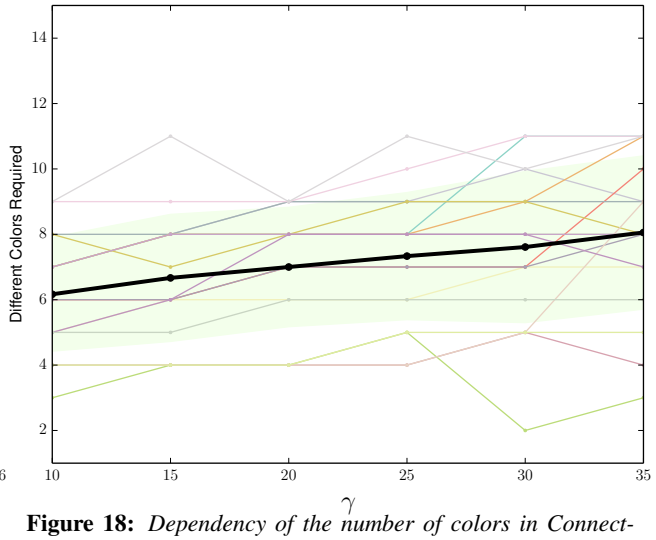


Figure 18: *Dependency of the number of colors in Connect-The-Closest-Dot puzzles on γ.*

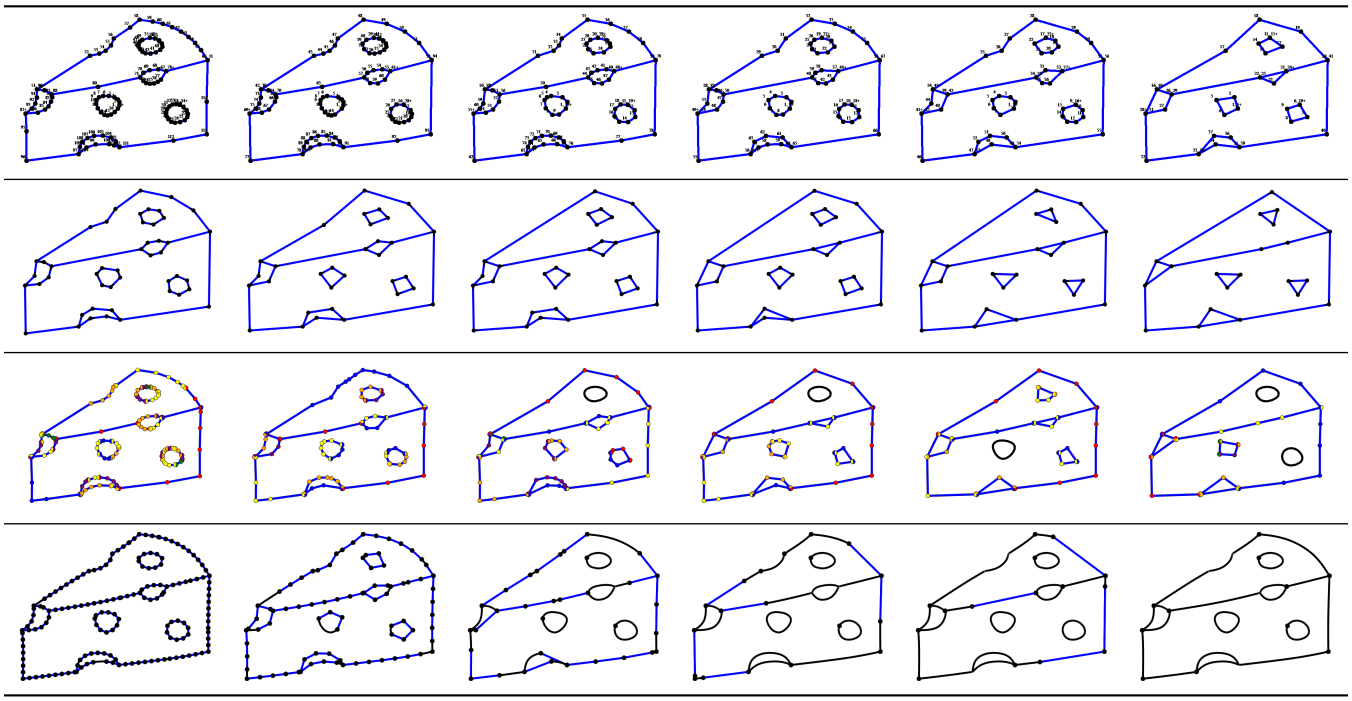


Figure 19: *Point puzzles for increasing values of ε. The puzzle types from bottom to top: Connect-The-Dots, Connect-That-Dot, Connect-The-Closest-Dot, and Connect-The-Unit-Dots. The pre drawn segments are shown in black and the solution to the puzzle is shown in blue.*

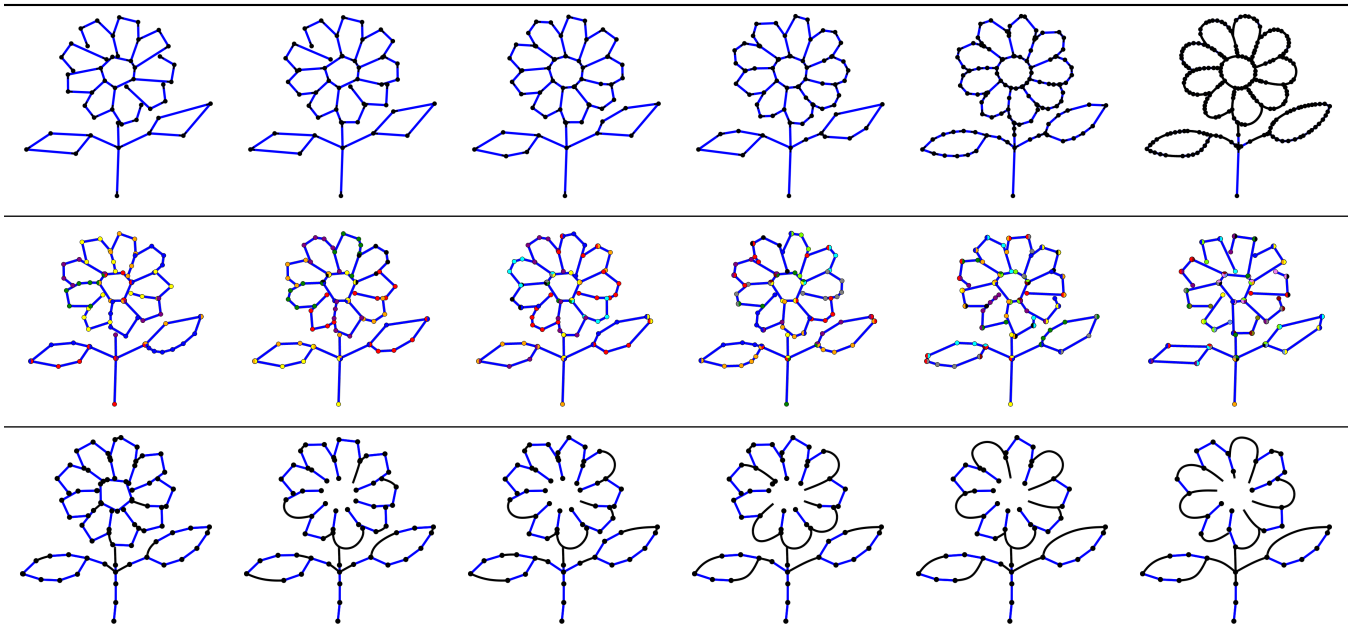


Figure 20: *Point puzzles for increasing values of γ. The puzzle types from bottom to top: Connect-That-Dot, Connect-The-Closest-Dot, and Connect-The-Unit-Dots. The pre drawn segments are shown in black and the solution to the puzzle is shown in blue.*