

## Fun with Geometric Duality

Michael Jünger      Michael Schulz      Wojciech Zychowicz

Dedicated to Jack Edmonds on the occasion of his 75<sup>th</sup> birthday  
on April 5, 2009

### Abstract

We present GEODUAL, a software for creating and solving geometric instances of the Minimum Spanning Tree problem, the Perfect Matching problem, and the Traveling Salesman problem, along with visual proofs of optimality.

## 1 Introduction

Almost 20 years ago, William R. Pulleyblank and the first author started assigning geometric interpretations to dual solutions of certain combinatorial optimization problems. Among other things, this resulted in colorful pictures that were not only aesthetically pleasing but also of educational value, because certain theorems can be appreciated visually without any formalism. The graphics software produced then by teams at Bellcore (“BINKY”), Simon Fraser University (“VisualMatching”) and the University of Cologne (“DUST” and “CATBOX”) does not run well on today’s systems, with the exception of VisualMatching by Michael Maguire [8] and the recent CATBOX-related Gato software [9] whose emphasis is on teaching algorithms and that is restricted to Euclidean perfect point matching and spanning trees. For many years the first author has been sorry for not being able to spice up his lectures in the subject area with the pictures and animations provided by “DUST”. The GEODUAL software [10] closes this gap. Using the multi-platform Qt 4 open source library, it is a widely usable tool for creating and solving geometric instances of the Minimum Spanning Tree problem, the Perfect Matching problem, and the Traveling Salesman problem, along with visual proofs of optimality.

In this brief user guide, we explain GEODUAL’s functionality for beginners, followed by some remarks for specialists. But even for the beginners, we do assume basic knowledge of the theory of Linear Programming.

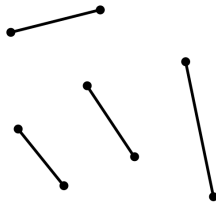
## 2 Basics

### 2.1 Perfect Point Matchings

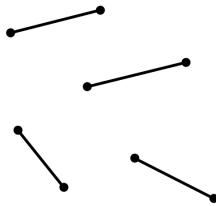
Consider an even cardinality set  $P$  of  $n$  points in the plane. The distance of any pair of points  $p$  and  $q$  with Cartesian coordinates  $(x_p, y_p)$  and  $(x_q, y_q)$  is the Euclidean distance  $d_{pq} = \sqrt{|x_p - x_q|^2 + |y_p - y_q|^2}$ .

A *perfect point matching* of the  $n$  points is a partitioning of the points into pairs. A perfect point matching is of *minimum length* if the sum of the distances of all  $\frac{n}{2}$  point pairs is minimum. This is a special version of the famous *perfect matching problem*.

Here is a perfect point matching on six points:

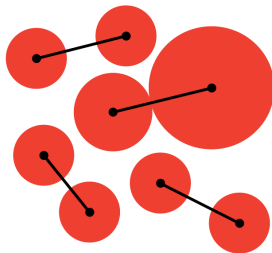


Its total length is the sum of the lengths of all straight line segments connecting the pairs. We claim it is not of minimum length, and we can easily prove it by showing a shorter one:

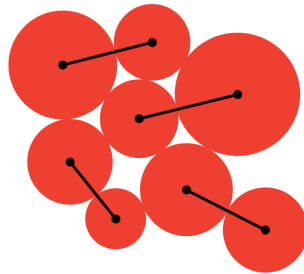


Now we claim this is a minimum length perfect point matching. How can we prove it?

Let us draw disks of radii  $r_p$  centered at the points  $p \in P$ . The disks may touch but not overlap (no two of them may have a common interior point):

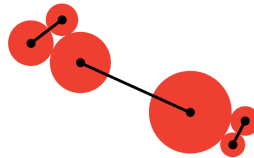


Now take *any* perfect point matching. Each of its straight line segments connecting pairs  $p$  and  $q$  is partially covered by the disks around  $p$  and  $q$ , so its length  $d_{pq}$  is at least the sum  $r_p + r_q$ . Each disk radius is accounted for exactly once. Therefore the total length of any perfect point matching is at least the sum of all  $n$  disk radii. Here is our perfect point matching along with a disk packing whose sum of the radii equals the sum of the lengths of all representing line segments:

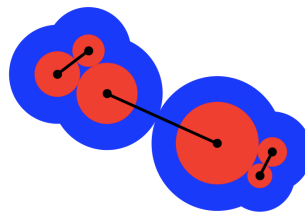


Therefore this perfect point matching is indeed of minimum length.

Unfortunately, this technique does not always work. Consider this instance:



While the point matching is clearly the shortest possible, there is no way to construct a disk packing that proves this. An additional observation helps. While each point is touched by exactly one line, each odd cardinality subset of the points is left by at least one line (that represents a pair one of whose members is not in the subset). That is, if we introduce *moats* that surround odd cardinality sets of points and keep insisting on no overlaps, the sum of all disk radii plus the *widths* of all moats is still a lower bound for the length of any perfect point matching:



We omit a formal description of moat construction and just appeal to intuition, the details can be found in [5, 2]. Since the sum of the disk radii plus the widths of the two three point moats equals the total length of our point matching, the picture “proves” that our point matching is indeed a shortest possible.

This technique allows us to construct “graphic optimality proofs” for any instance of the perfect point matching problem. Some basic knowledge of the theory of Linear Programming is needed here. Let us write  $r_p$  for the radius of the disk surrounding point  $p \in P$ , and, for any odd cardinality subset  $S \subset P$  of the points ( $3 \leq |S| \leq \frac{n}{2}$ ), let  $w_S$  denote the width of the moat around  $S$ . Then the problem of finding a feasible disk/moat packing that results in the best possible lower bound on the total length of any perfect point matching can be written as a linear programming problem as follows:

$$\text{maximize } \sum_{p \in P} r_p + \sum_{S \subset P, |S| \text{ odd and } 3 \leq |S| \leq \frac{n}{2}} w_S$$

such that

$$\begin{aligned} r_p + r_q + \sum_{|S \cap \{p,q\}|=1} w_S &\leq d_{pq} && \text{for all } p, q \in P, q \neq p, \\ r_p &\geq 0 && \text{for all } p \in P, \\ w_S &\geq 0 && \text{for all } S \subset P, |S| \text{ odd and } 3 \leq |S| \leq \frac{n}{2}. \end{aligned}$$

The restrictions make sure that the disks have non-negative radii and the moats have non-negative widths. They also guarantee that the disks and moats are non-overlapping by stipulating that for any pair of points, the sum of the radii of the two associated disks plus the sum of the widths of all moats separating them never exceeds their distance. The objective function asks for a disk/moat packing that provides the best possible lower bound on the length of any perfect point matching.

We introduce  $\binom{n}{2}$  variables  $x_{pq}$  for all pairs of distinct points  $p$  and  $q$ . The dual of the above linear programming problem is then

$$\text{minimize } \sum_{p, q \in P, q \neq p} d_{pq} x_{pq}$$

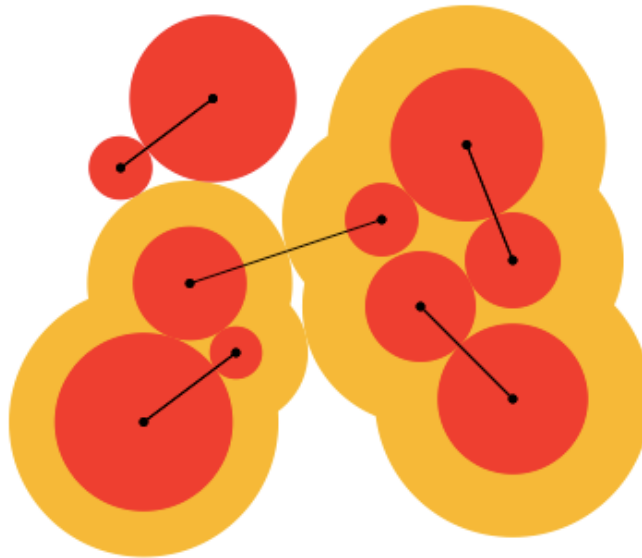
such that

$$\begin{aligned} \sum_{q \in P, q \neq p} x_{pq} &\geq 1 && \text{for all } p \in P, \\ \sum_{|S \cap \{p,q\}|=1} x_{pq} &\geq 1 && \text{for all } S \subset P, |S| \text{ odd and } 3 \leq |S| \leq \frac{n}{2}, \\ x_{pq} &\geq 0 && \text{for all } p, q \in P, q \neq p. \end{aligned}$$

We can represent any perfect point matching by setting  $x_{pq} = 1$  if  $p$  and  $q$  are matched and  $x_{pq} = 0$  otherwise. Then the  $x_{pq}$  will satisfy all restrictions of

the latter linear programming problem. The “miracle” is that it follows from the groundbreaking work of Jack Edmonds [3, 4] and the geometric nature of our problem that such “characteristic vectors” of perfect point matchings are the only solutions that the simplex method for linear programming can return. This is way beyond “basics”. But once this result is accepted, it is clear that our construction always works, and all we have to do is solve a linear programming problem (along with its dual). A second “miracle” (also beyond this exposition) is that there is no need to worry about the (exponentially) large set of variables in the first and restrictions in the second linear programming problem. Indeed, both linear programming problems can be solved in polynomial time due to the results of Jack Edmonds [3, 4].

Here is a 10 point example of a pair of optimum solutions to the primal/dual pair of linear programs.



Just by visual inspection, we can argue as follows:

- The black lines display indeed a perfect point matching. Therefore the dual solution is feasible for the dual linear program.
- The red disks and the orange moats are non-overlapping. On any straight-line connection between two points, the radii of the two disks assigned to them plus the widths of the moats separating them add up to no more than the distance of the two points. Therefore the primal solution is feasible for the primal linear program.

Now we see the optimality of the perfect point matching by applying one of two basic theorems of Linear Programming:

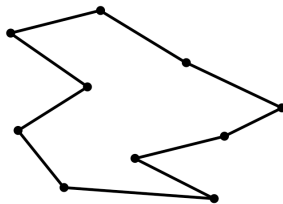
1. The sum of all disk radii and all moat widths equals the sum of the lengths of all black lines. Therefore we obtain optimality from the Weak Duality Theorem of Linear Programming.
2. For each black line, there is no white gap, i.e., whenever a variable  $x_{pq}$  has a positive value ( $=1$ ), the associated primal constraint holds with equality. And whenever a radius or a moat width is positive, it is traversed by exactly one black line, so the associated dual constraint holds with equality. Therefore we obtain optimality from the Complementary Slackness Theorem of Linear Programming.

“Geometric Duality” refers to assigning such geometric interpretations to pairs of primal and dual solutions of certain linear programs.

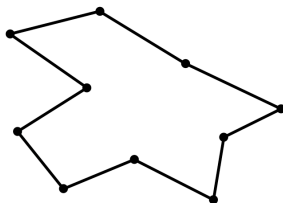
## 2.2 Tours

Now let us apply the same techniques to tours. A *tour* is a set of  $n$  point pairs  $\{(p_{t_1}, p_{t_2}), (p_{t_2}, p_{t_3}), \dots, (p_{t_{n-1}}, p_{t_n}), (p_{t_n}, p_{t_1})\}$  where  $\langle p_{t_1}, p_{t_2}, \dots, p_{t_n} \rangle$  is a permutation of the  $n$  points. Each tour corresponds to exactly  $2n$  permutations, so there are  $\frac{(n-1)!}{2}$  many different tours a traveling salesman can take when he wants to visit each point exactly once and return home. This is a special case of the famous *traveling salesman problem*. We are looking for tours of minimum length.

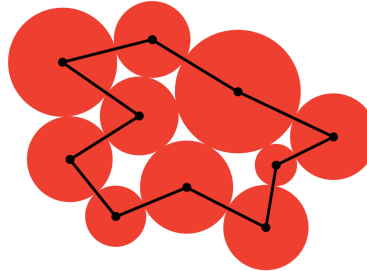
Here is a tour on 10 points:



We can easily convince ourselves that it is not a shortest tour by showing a shorter one:

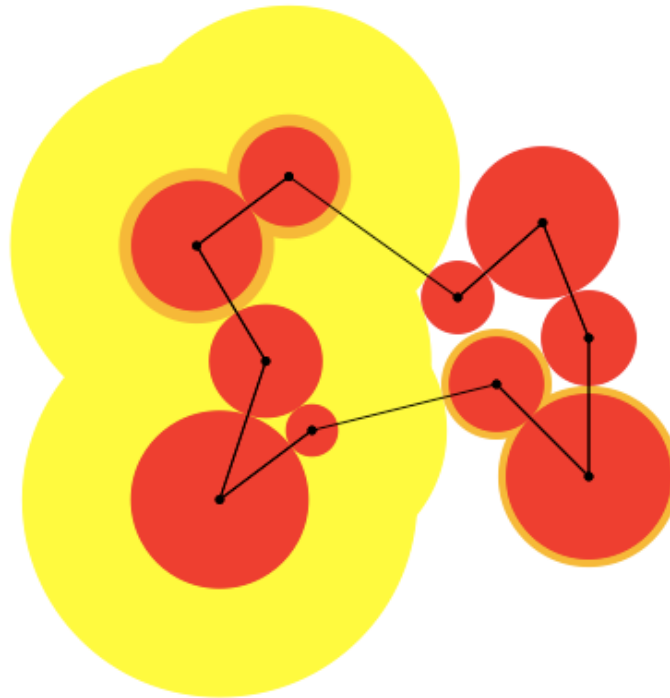


This one is a shortest possible tour, and we can prove it by showing a non-overlapping system of disks:



Since each point is paired with exactly two other points, the disk surrounding it is traversed twice by any tour. Therefore, twice the sum of the radii of all  $n$  disks is always at most the total length of any tour.

Like for perfect point matching, we can introduce moats as well, with the proviso that every moat is traversed at least twice, and there is no odd cardinality restriction here. For the same 10 point example we used in our last perfect point matching example, we do need moats for an optimality proof:



The pair of linear programs for the tour problem look very similar to the one we gave for perfect point matching. The primal task is to

$$\text{maximize } 2 \sum_{p \in P} r_p + 2 \sum_{S \subset P, 3 \leq |S| \leq \frac{n}{2}} w_S$$

such that

$$\begin{aligned} r_p + r_q + \sum_{|S \cap \{p,q\}|=1} w_S &\leq d_{pq} && \text{for all } p, q \in P, q \neq p, \\ r_p &\geq 0 && \text{for all } p \in P, \\ w_S &\geq 0 && \text{for all } S \subset P, 3 \leq |S| \leq \frac{n}{2} \end{aligned}$$

and the dual task is to

$$\text{minimize } \sum_{p, q \in P, q \neq p} d_{pq} x_{pq}$$

such that

$$\begin{aligned} \sum_{q \in P, q \neq p} x_{pq} &\geq 2 && \text{for all } p \in P, \\ \sum_{|S \cap \{p,q\}|=1} x_{pq} &\geq 2 && \text{for all } S \subset P, 3 \leq |S| \leq \frac{n}{2}, \\ x_{pq} &\geq 0 && \text{for all } p, q \in P, q \neq p. \end{aligned}$$

However, unlike for perfect point matching, this construction does not always work, in fact, it is quite unlikely to work for instances of more than, say, 20 points. Again, the reason is beyond “basics”, see [5, 2, 1].

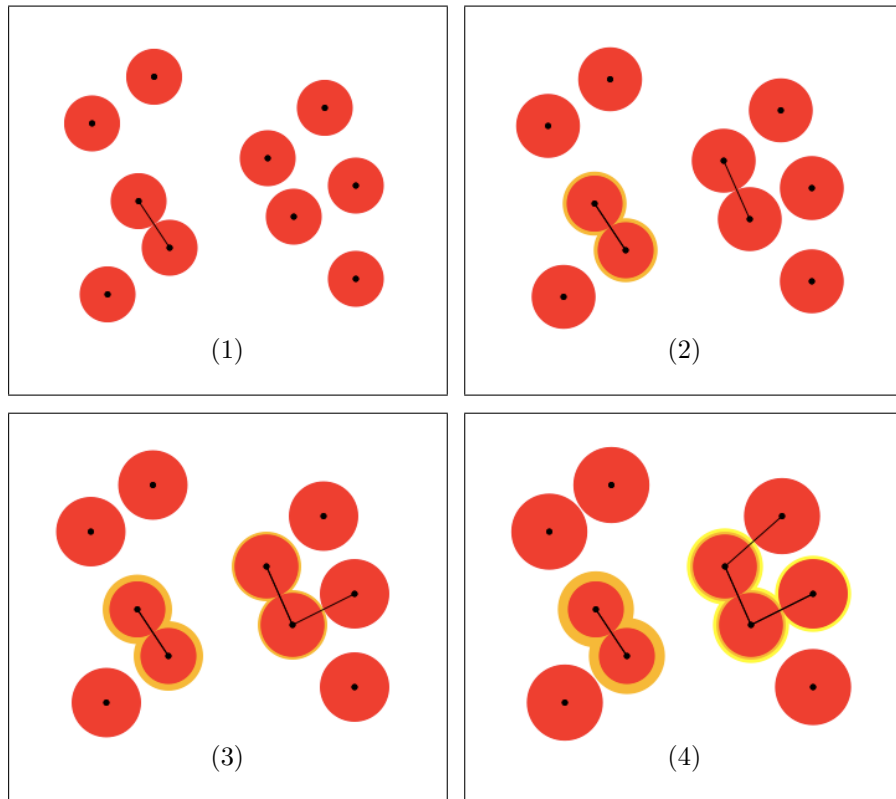
### 2.3 Spanning Trees

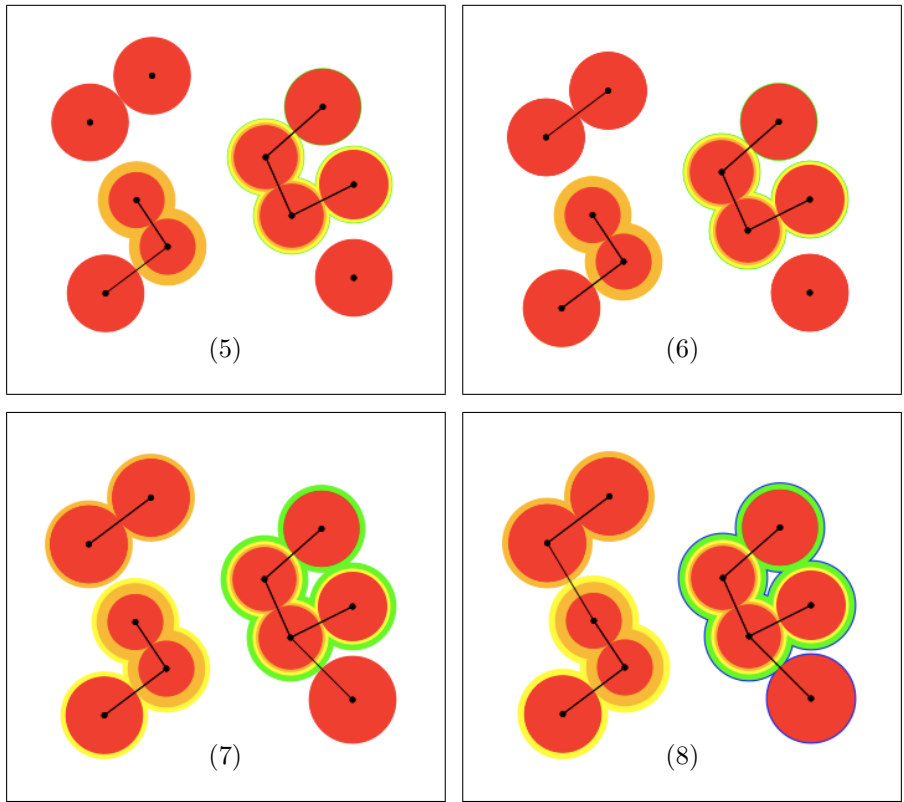
We wish to connect point pairs by straight lines, such that, if the lines were streets, we could drive from any point to any other point. And we would like the total length of all such lines to be minimum. We may assume that a solution cannot have a cycle of the form  $\{(p_{t_1}, p_{t_2}), (p_{t_2}, p_{t_3}), \dots, (p_{t_{k-1}}, p_{t_k}), (p_{t_k}, p_{t_1})\}$  for some  $1 \leq k \leq n$  because removing any of its point pairs would still be a solution of at most the same total length. Therefore, there should be no more than  $n - 1$  lines. On the other hand, we do need at least  $n - 1$  lines, because otherwise at least one point would not be reachable from every other point. So we are looking for solutions consisting of  $n - 1$  lines with no cycles, and these are called *spanning trees*. Our problem is the *minimum length spanning tree problem*, a special case of the *minimum weight spanning tree problem*.

There are several very efficient algorithms for solving this problem (in a more general setting than considered here), and we shall concentrate on the one given by Kruskal in 1956 [7]. Kruskal’s algorithm builds a minimum length spanning

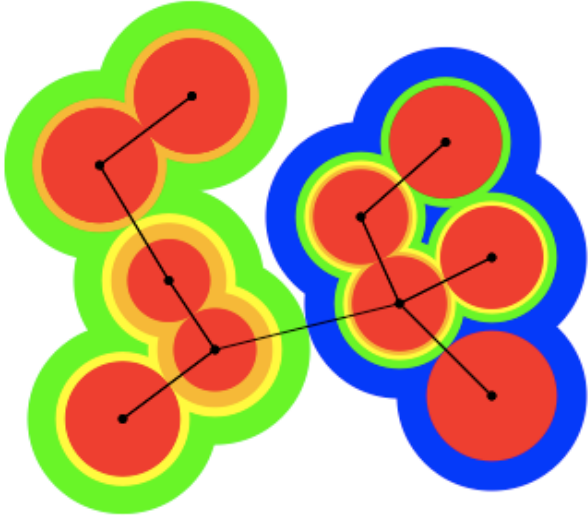


tree by starting with the shortest line and then adding successively the shortest remaining line that produces no cycle. It stops as soon as  $n - 1$  lines are chosen. We shall mimic Kruskal's algorithm for minimum length spanning trees on a hypothetical analog computer that is able to "blow up" disks around points and moats around point sets at uniform speed. Whenever there is a collision (i.e., blowing up further would produce overlapping disks/moats), we draw a line connecting two points causing the collision, and treat the newly connected point set as a new unit around which a new moat is growing now. Let us run the analog computer for our standard example on 10 points. We take snapshots whenever a new line is produced, i.e., a total of 9 snapshots:





Here is the final minimum length spanning tree along with its graphic optimality proof in its full beauty:



The correctness of Kruskal’s algorithm (that is quite easy to see) makes sure that we have indeed found a minimum length spanning tree (whose correct construction is easily verified from the final picture). In addition, we can interpret this picture like those for perfect point matchings and tours, yet we must take a little detour that is beyond “basics”. We will sketch it in Section 4.

## 2.4 Other Metrics

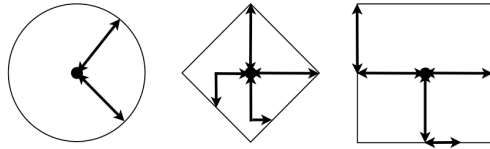
So far, we have considered Euclidean distances  $d_{pq} = \sqrt{|x_p - x_q|^2 + |y_p - y_q|^2}$  for point pairs  $p$  and  $q$  with Cartesian coordinates  $(x_p, y_p)$  and  $(x_q, y_q)$ , i.e., the distance of two points is equal to the length of the straight line segment connecting them. If  $p$  and  $q$  are on the grid-like streets of Manhattan,



a New York taxi driver would go straight from  $p$ , make a perpendicular right or left turn, and then go straight again to reach  $q$  (forget about one-way-streets and the Broadway). The distance travelled is the *Manhattan distance*  $d_{pq} = |x_p - x_q| + |y_p - y_q|$ .

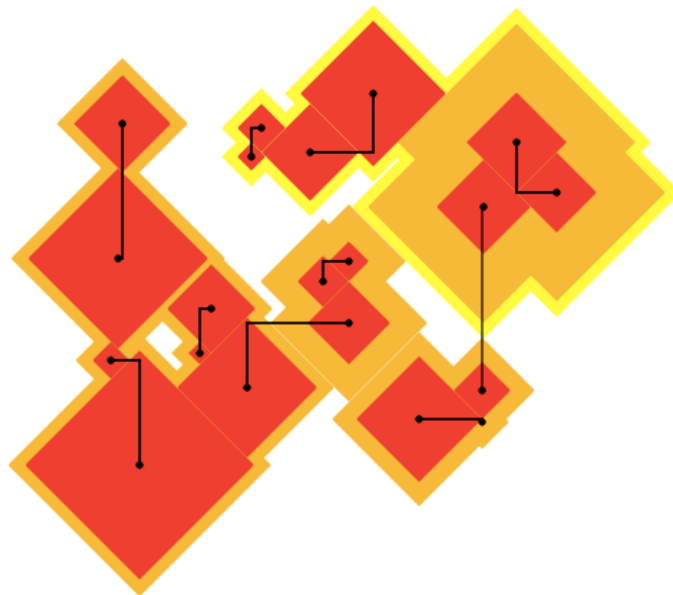
Now imagine a pen moving from  $p$  to  $q$  driven by two motors, one responsible for the horizontal, the other for the vertical movement. When both motors run simultaneously at the same speed, the time this takes only depends on the maximum of the horizontal and vertical distances travelled. The appropriate “distance” is the *Maximum distance*  $d_{pq} = \max\{|x_p - x_q|, |y_p - y_q|\}$ .

Here are the points of equal distance from a given point when distances are measured in Euclidean, Manhattan, and Maximum metric, respectively:

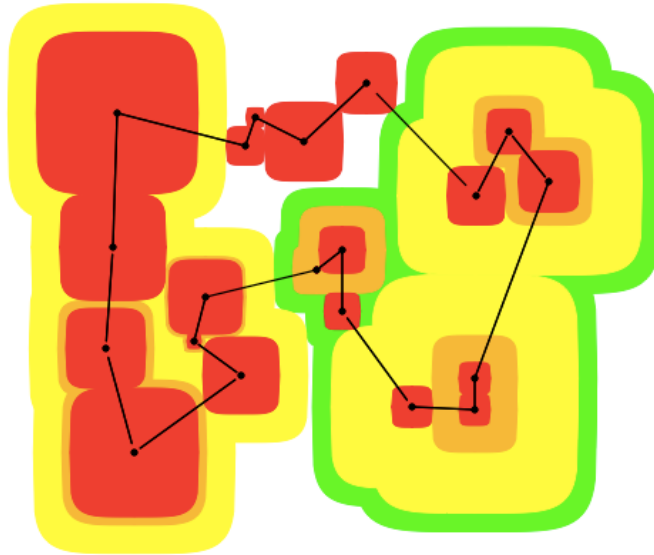


For distance 1, the sets of points on the circle, diamond, and square, respectively, are called the *unit balls* for the respective metrics. In fact, we can consider infinitely many metrics called “ $L_k$ -metrics” in which  $p$  and  $q$  have distance  $d_{pq} = \sqrt[k]{|x_p - x_q|^k + |y_p - y_q|^k}$ . The Manhattan metric is  $L_1$ , the Euclidean metric is  $L_2$ , and the Maximum metric is  $L_\infty$ . For  $k \geq 3$ , the unit balls look like beer mats, and with increasing  $k$ , their shape converges to the square “unit ball” of the Maximum metric.

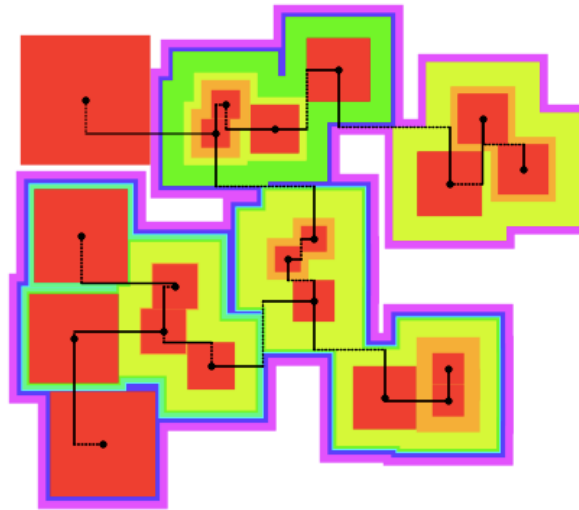
Everything said so far for the Euclidean metric applies to any of these metrics as well. We take a 20 point example to demonstrate this. Here is a Manhattan ( $L_1$ ) perfect point point matching with optimality proof:



It is not clear how to draw the “lines” for  $L_k$  with  $k \geq 3$ . We use straight lines whose lengths equal the  $L_k$ -distances of their end points, that is why there may be gaps at either end. Here is an  $L_5$  tour with optimality proof:

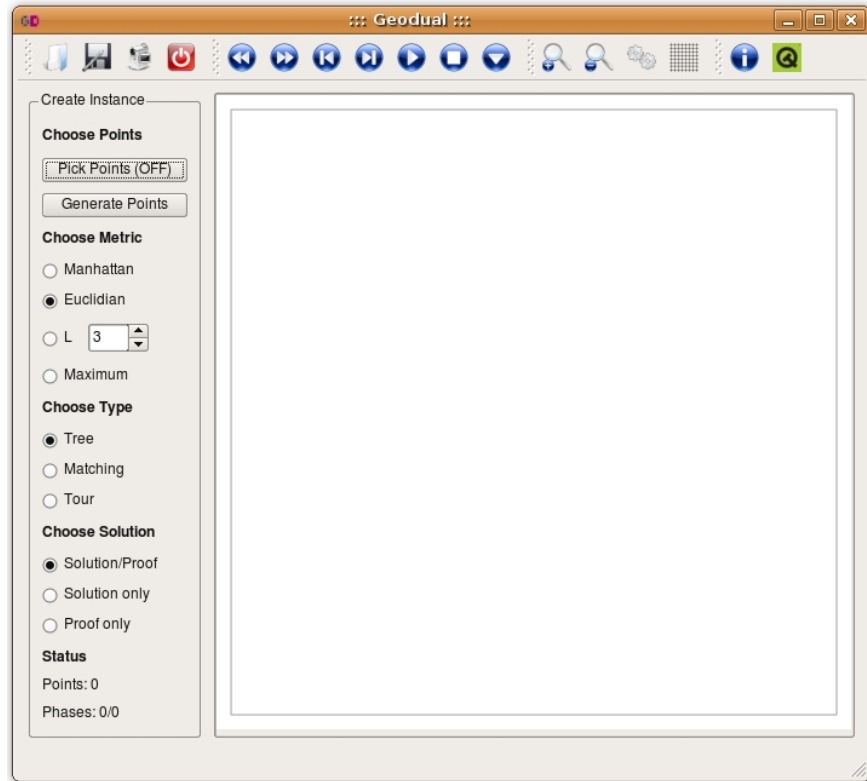


“ $L_\infty$ -lines” look almost like “ $L_1$ -lines”, except that the segments representing the longer of the horizontal and vertical distances (that determines the  $L_\infty$ -distance of the end points) are drawn solid and the others are drawn dotted. Here is a Maximum ( $L_\infty$ ) spanning tree with optimality proof:



### 3 Using GEODUAL

When GEODUAL starts, it looks like this:



The application's main window is divided into three parts, the largest of which is the *painting area*. In addition, there are the *groupbox* at the left hand side and the *toolbar* on top. In the following, we will explain each part.

#### 3.1 Painting area

The painting area is the most important part of the application. All painting happens on the white canvas at a resolution of  $500 \times 500$  pixels within the grey rectangle that represents the unit square in the real plane. The only user-controlled interactions with the canvas are

- creating or deleting points,
- zooming in for a larger image,
- zooming out for a smaller image,
- switching a guiding grid on or off.

## 3.2 Groupbox

The groupbox provides the functionality for creating an instance consisting of points in the unit square, specifying a metric for distance measurement, and choosing the type of problem and solution. Finally, it displays status information.

There are two possibilities for creating an instance under “Choose Points”:

- The first push-button toggles between “Pick Points (Off)” and “Pick Points (On)”. In the latter mode the user may manually select points in the unit square by clicking the left mouse-button at the desired locations. Selected points can be removed by clicking the right mouse-button close to their location.
- An alternative way of generating an instance is provided by the “Generate Points” push-button. When pushed, the user is first asked for the number of points, and subsequently for a random seed. Once these values are provided, the program generates a pseudo-random instance and displays it on the canvas.

Users should not try instances with more than 30 points for perfect point matching or tour computations: Larger instances are likely to result in a “not enough memory” message. Unfortunately, this may happen occasionally also for smaller instances, see also Section 4. For tour computations, no more than 20 points should be tried, because a successful optimization is quite unlikely for larger instances.

When an instance is available, the next step is to choose the desired metric. The “Choose Metric” part provides four radio-buttons, the first two for the Manhattan ( $L_1$ ) and the Euclidean metric ( $L_2$ ). The third radio-button allows to choose a metric in the range from  $L_3$  to  $L_9$  by adjusting the spinbox next to the “L” at the radio-button. Finally there’s a fourth radio-button for the Maximum metric ( $L_\infty$ ).

The next step is the specification of which problem the user wishes to solve. The section “Choose Type” provides radio-buttons for selecting one of the three problems: “Tree” for the spanning tree problem, “Matching” for the perfect point matching problem, and “Tour” for the tour problem.

In the “Choose Solution” section, the user can select which type of solution s/he wants to see. The default selection is the “Solution/Proof” option in which optimum solutions are displayed along with optimum disk/moat packings. The other two choices are to display “Solution only” or “Proof only”. The former is of interest if the programs gives the “moats are not nested” error message (see Section 4), the latter for æsthetic reasons: disk/moat packings are beautiful by themselves, even if nothing is proved.

The last section “Status” provides some information about the current instance and the state of solution display on the canvas. The point counter always shows the current number of points on the canvas. For minimum length spanning trees, “Phase” refers to the progress in terms of Kruskal’s algorithm as

explained in Section 2, i.e., there are  $n - 1$  phases for an  $n$  point instance, in each of which one line is drawn. For the perfect point matching problem and the tour problem, there are no natural phases in the solution process, yet we have introduced artificial phases that correspond to the “depths” of the disks and moats. We will see below how one can navigate between phases. When “0/0” is displayed, this simply means that no solution has been calculated yet. Calculation is triggered by any of the “◀◀”, “▶▶”, “◀”, “▶”, and “▶” buttons in the toolbar, as well as when a radio-button in the groupbox is clicked.

Before we turn to the toolbar, we should discuss what can go wrong with a given instance:














- The instance may be too large.
- It may not be possible to compute a shortest tour due to reasons explained in Section 4.
- The calculation of a disk/moat packing may fail for perfect point matchings or tours, even though an optimum solution has been found. This is a problem that will be fixed in the next version of GEODUAL, see Section 4. When this happens, there is still the possibility to choose “Solution only” in the “Choose Solution” section.

### 3.3 Toolbar





The toolbar





supports the following actions:

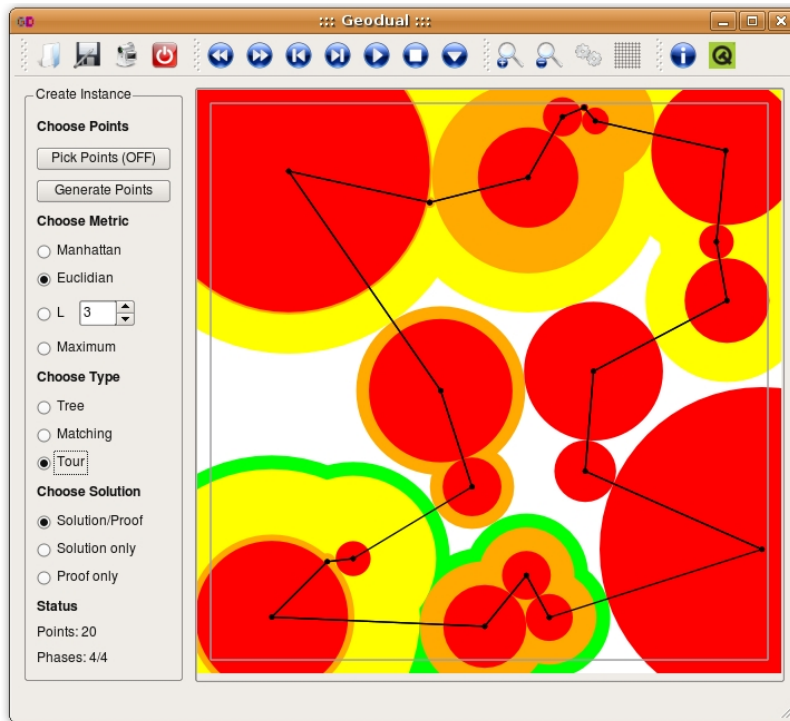
-  “Load”: Load a set of stored points from a text file.
-  “Save”: Save the current set of points to a text file.
-  “Screenshot”: Take a screenshot of the current canvas.
-  “Close”: Quit the application.
-  “Previous”: Go to the previous phase.
-  “Next”: Go to the next phase.
-  “Start”: Return to the initial view, showing only the points on the canvas.
-  “Final”: Jump to the last phase and display the final result.
-  “Play”: Run through the rest of the phases.
-  “Stop”: Stop playing after the current phase is painted.
-  “Clear”: Clear the canvas (including the point set).
-  “Zoom in”: Zoom into the canvas.
-  “Zoom out”: Zoom out of the canvas.



- 
 “Animation”: Switch animation on/off (if on, the symbol is pressed).  
 When animation is on, the transitions between phases are smooth, i.e., disks and moats “grow” rather than “jump” from one state to the next.
- 
 “Grid”: Switch the grid on/off (if on, the symbol is pressed).  
 When the grid is visible and points are generated with the mouse, they will snap to the grid. This allows for regular patterns.
- 
 “About”: Get information about the authors and the web site.
- 
 “About Qt”: Get some information about Qt.

### 3.4 Example

Now we are ready to use GEODUAL. We generate a random instance by pushing the “Generate Points” button followed by “20 <return>” for the number of points and “9 <return>” for the random seed. Twenty random points appear on the canvas. We wish to see an optimum solution of the tour problem with proof when distances are measured in the Euclidean metric. Therefore we push the “Euclidean” radio-button, the “Tour” radio-button, and the “Solution/Proof” radio-button. We would like a nice animation in our first experiment. Therefore, we push the “” button as well. Now we push “” and watch a little movie that ends like this:



## 4 Explanations for Experts

### 4.1 Geometric Duality and Spanning Trees

How can we interpret the disk and moat packings for minimum length spanning trees of section 2 in terms of geometric duality? We give an interpretation that has been introduced in [6]:

We choose an arbitrary point  $\hat{p}$  and consider the following pair of dual linear programming problems. The primal task is to

$$\text{maximize } \sum_{\emptyset \neq S \subset P, \hat{p} \notin S} 2w_S$$

such that

$$\sum_{\emptyset \neq S \subset P, |S \cap \{p, q\}|=1} w_S \leq d_{pq} \quad \text{for all } p, q \in P, q \neq p, \quad (1)$$

$$\sum_{\emptyset \neq S \subset P, p \in S} w_S = \alpha \quad \text{for all } p \in P, \quad (2)$$

$$w_S \geq 0 \quad \text{for all } \emptyset \neq S \subset P, \quad (3)$$

and the dual task is to

$$\text{minimize } \sum_{p, q \in P, q \neq p} d_{pq} x_{pq}$$

such that

$$\sum_{p, q \in P, q \neq p, |S \cap \{p, q\}|=1} x_{pq} + \sum_{q \in S} y_q \geq \begin{cases} 2 & \text{if } \hat{p} \notin S \\ 0 & \text{if } \hat{p} \in S \end{cases} \quad \text{for all } \emptyset \neq S \subset P, \quad (4)$$

$$\sum_{q \in P} y_q = 0, \quad (5)$$

$$x_{pq} \geq 0 \quad \text{for all } p, q \in P, q \neq p. \quad (6)$$

The primal problem is another variant on the disk/moat packing problems we have studied for perfect point matchings and tours:

- (i) We drop the special treatment of cardinality one sets and treat the previous disks as one point moats.
- (ii) We can construct moats surrounding any set of points, not just odd sets.
- (iii) We must “balance” the packing in that (2) requires that the sum of the widths of the sets of moats surrounding each point be equal.
- (iv) The objective function ignores the moats surrounding one point  $\hat{p}$  (the choice of which does not matter by (iii)), but doubles the rest.

Surprisingly, the dual problem is just the minimum length spanning tree problem, slightly disguised. Kruskal’s algorithm with minor modifications will build optimum solutions to both linear programming problems.

For the primal problem, we will build a solution  $w$  as we perform Kruskal’s algorithm, starting with  $w_S = 0$  for all  $\emptyset \neq S \subset P$ . At each stage, for each tree  $T$  that we have built, all points  $p$  of  $T$  will satisfy  $\sum_{\emptyset \neq S \subset P, p \in S} w_S = \alpha(T)$  where  $\alpha(T)$  equals one half of the length of a longest line of  $T$ , respectively  $\alpha(T) = 0$  if  $T$  has only one point.

Now suppose we add line  $t$  joining points  $p_1 \in T_1$  and  $p_2 \in T_2$ . We construct a moat of width  $\frac{1}{2}d_{p_1 p_2} - \alpha(T_i)$  around  $T_i$  for  $i = 1, 2$ . (Since  $d_{p_1 p_2}$  is at least as great as the length of the longest line in  $T_1$ , resp.  $T_2$ , these widths are nonnegative.) Let  $\alpha(T) = \frac{1}{2}d_{p_1 p_2}$ , where  $T$  is the new tree produced. It is clear that when we terminate,  $w$  satisfies (1) and (2) with  $\alpha(T)$  equal to one half of the length of a longest line of  $T$ , the minimum length spanning tree produced. Notice that the solution  $w$  satisfies (1) with equality for every line of  $T$ .

Now we construct a feasible solution to the dual problem. Let  $x_{pq} = 0$  if  $(p, q)$  is not a line of  $T$  and let  $x_{pq} = 1$  if  $(p, q)$  is a line of  $T$ . Choose arbitrarily some point  $\hat{p}$  of  $T$ . Orient all lines of  $T$  towards  $\hat{p}$ . For each  $q \in P$ , define  $y_q$  equal to the outdegree of  $q$  in  $T$  (i.e. the number of arrows leaving  $q$ ) minus its indegree (i.e. the number of arrows entering  $q$ ). Using induction on  $T$ , we can prove that this is a feasible solution to the dual problem. Since  $x_{pq} = 1$  only for lines in  $T$ , for every such pair  $p$  and  $q$  the inequality (1) holds with equality. Again, using induction, we can show that (4) holds with equality for all  $\emptyset \neq S \subseteq P$  with  $w_S > 0$ . (Show that it holds for two point trees, start with an arbitrary tree, remove one pendent point, apply induction.)

Therefore, these solutions satisfy the complementary slackness conditions for optimality.

## 4.2 Implemented Algorithms

For the minimum length spanning trees, we implemented Kruskal’s algorithm with the add-ons outlined above. It should always work, even for large point sets. For perfect point matchings and tours, we use cutting plane algorithms. For tours, we had little choice, really. But for perfect point matchings, this is debatable, of course. We could (should?) have used an implementation of Edmond’s blossom shrinking algorithm. The reason for our choice is that the cutting plane implementations for perfect matchings and for traveling salesman tours are very similar. In the former, separation amounts to finding odd minimum capacity cuts, and in the latter, general minimum capacity cuts. The cutting plane approach is flexible enough to give rise to our hope that the community will extend the software with more examples of combinatorial optimization problems.

## 4.3 Shortcomings and Plans for the Future

Clearly, GEODUAL is bound to fail when trying to compute tours for instances that cannot be solved on the sub-tour relaxation of the traveling salesman prob-

lem. In this case an “optimization failed” message is rightfully issued. In fact, such a result is very likely for instances with more than 20 points. However, tour as well as perfect point matching computations may currently also fail with the message “moats are not nested”. The reason is that, unlike for Edmonds’ blossom shrinking algorithm, the cutting plane algorithms are not guaranteed to find nested families of blossom/subtour constraints (our moats) for point matchings or tours. Our current implementation (version 1.0) contains only primitive heuristic measures to prevent this. But it is certainly possible to enforce nested families as required for our purposes, and we are currently working on the next version that will guarantee primal/dual optima for any perfect point matching instance (of reasonable size) and for any tour instance that can be solved on the sub-tour relaxation.

The amount of memory GEODUAL 1.0 allocates for perfect point matching and tour computations only depends on the number of points. Even though this works for most instances with no more than 30 points, there may be an occasional memory overflow when too many cutting planes are generated.

GEODUAL 1.0 is available as Linux, Mac, and Windows executables. For further developments, please see the “release notes” on page 21.

Beyond our own development efforts, we very much hope for contributions from the Mathematical Programming Community!

## References

- [1] D. L. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook, *The Traveling Salesman Problem: A Computational Study*, Princeton Univ. Press, 2006.
- [2] W. J. Cook, W. H. Cunningham, W. R. Pulleyblank, and A. Schrijver, *Combinatorial Optimization*, John Wiley and Sons, 1998.
- [3] J. Edmonds, Paths, Trees, and Flowers, *Can. J. Math.* 17 (1965), 449–467.
- [4] J. Edmonds, Maximum Matching and a Polyhedron with 0,1-Vertices, *J. Res. Nat. Bur. Standards* 69B (1965), 125–130.
- [5] M. Jünger and W. R. Pulleyblank, Geometric Duality and Combinatorial Optimization, in: Chatterji et al. (eds.), *Jahrbuch Überblicke Mathematik 1993*, Vieweg (1993), 1–24.
- [6] M. Jünger and W. R. Pulleyblank, New Primal and Dual Matching Heuristics, *Algorithmica* 13 (1995), 357–380.
- [7] J. B. Kruskal, On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem, in: *Proceedings of the American Mathematical Society* 7 (1956), 48–50.
- [8] [http://www.math.sfu.ca/~goddyn/Courseware/Visual\\_Matching.html](http://www.math.sfu.ca/~goddyn/Courseware/Visual_Matching.html)
- [9] <http://www.gato.sourceforge.net>
- [10] [http://www.informatik.uni-koeln.de/ls\\_juenger/research/geodual](http://www.informatik.uni-koeln.de/ls_juenger/research/geodual)

## Release Notes

### February 2010

GEODUAL 1.0 has been released in April 2009 on the occasion of Jack Edmonds' 75th birthday. One shortcoming that we could not fix by that time has bothered us very much, namely the failure message "Moats are not nested." We are still working on an elegant solution, but for the time being, we have added an "untangling" post-processing step in release GEODUAL 1.1 of February 2010. You should be able to solve all matching instances up to about 100 points now. In any case, there should be no "Moats are not nested" message, neither for matching nor for tour computations.

In the meantime, Michael Schulz and Wojciech Zychowicz have left, and Martin Gronemann has joined the GEODUAL development team.