THE TRAVELING SALESMAN PROBLEM FOR MOVING POINTS ON A LINE *

GÜNTER ROTE †

Abstract. A salesperson wants to visit each of n objects that move on a line at given constant speeds in the shortest possible time, subject to an upper bound on her own speed. We present a dynamic programming algorithm to solve this problem in $O(n^3)$ time.

1. Introduction. In the Euclidean Traveling Salesman Problem, n points (cities) in the plane (or in some Euclidean space of other dimension) are given, and a salesperson wants to visit all of them by a path which is as short as possible. We consider a generalization of this problem where the points are moving on straight lines at constant speeds, and the salesperson wants to visit all of them as fast as possible, subject to an upper bound on her speed. Since the Euclidean Traveling Salesman Problem in the plane is already NP-hard, even with fixed points, we restrict our attention to the one-dimensional problem, i. e., to points moving on a line.

2. Properties of the Optimal Solution. Henceforth we will use a more appealing language and consider a cat that can move at a given maximum speed and that wants to catch n mice which move with constant velocities (the mice-collecting traveling salescat problem). Formally, we assume that the initial position of the cat at time 0 is 0, and that the maximum speed is 1. Denoting the cat's position at time t by $C(t)$, we have $C(0) = 0$ and $|C(t) - C(u)| \le |t - u|$. The location of mouse i at time t is given by $c_i + v_i t$. Let m denote the number of mice which are initially to the left of the cat. We number these mice so that mouse 1 is the mouse moving fastest to the left and the mouse moving fastest to the right is mouse m, i. e., $-1 < v_1 < v_2 < \cdots < v_m < 1$, and $c_i < 0$ for $i = 1, 2, \ldots, m$ (see figure 1). If several of these have the same velocities v_i, it is clearly sufficient to retain only the leftmost of them. The mice on the right of the cat are ordered in the same way, i. e., $-1 < v_{m+1} < v_{m+2} < \cdots < v_n < 1$, and $c_i > 0$ for $i = m+1, m+2, \ldots, n$. (The mice with $c_i = 0$ obviously have no existence.)

We will say that the cat *meets* mouse i if $C(t) = c_i + v_i t$, the cat is *to the right of* mouse i if $C(t) \ge c_i + v_i t$, and the cat is *to the left of* mouse i if $C(t) \le c_i + v_i t$. Note that the possibility of equality is included in these definitions. Note also that the cat can meet a mouse even after it has eaten the mouse, i. e., we consider just the straight paths of the mice. Clearly, the cat can eat a mouse if it is to the right of it at some time instant and if it is to the left of it at some possibly different time instant. Thus, we are looking for a path $C\colon [0, T] \to \mathbb{R}$ of the cat which is to the left of every mouse $1, 2, \ldots, m$ at least once in the interval $[0, T]$ and to the right of every mouse $m + 1, m + 2, \ldots, n$ at least once in the interval $[0, T]$. The total duration T should be a short as possible.

LEMMA 2.1. *Assume that the cat can start in a position coinciding with mouse i any time after some given time t_0 and it wants to reach mouse j as early as possible. The fastest way to reach mouse j is to start at time t_0 and to travel into the direction towards mouse j at full speed, and moreover, this is the only way to reach mouse j in the shortest possible time.*

* This research was supported by the Leonardo Fibonacci Institute in Trento, Italy.

† Author's address: Institut für Mathematik, Technische Universität Graz, Steyrergasse 30, A-8010 Graz, Austria. Electronic mail: rote@ftug.dnet.tu-graz.ac.at

Proof. The lemma is intuitively obvious, and it can be checked by straightforward calculations, using the fact that $|v_i| < 1$ and $|v_j| < 1$. □

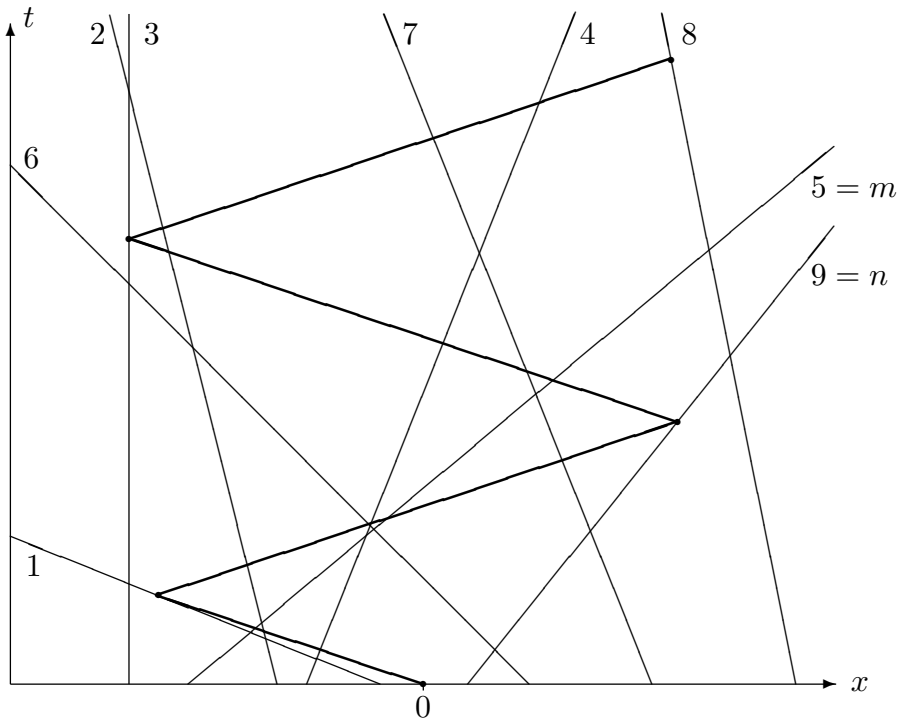


FIG. 1. *A possible shape of an optimal path according to theorem 2.2. The x-axis and the time axis are not drawn to scale. Therefore the cat's path (bold line) does not have slope* ± 1.

The following theorem gives a necessary condition for an optimal path of the cat.

THEOREM 2.2. *The optimal path of the cat is a piecewise linear function with at most n linear pieces, whose slopes are alternately* $+1$ *and* -1. *The breakpoints (turns of the cat) are at positions where the cat meets a mouse (see figure 1). For each turn and also for the final position there is a mouse so that this is the only position where the cat meets this mouse.*

Proof. Let $j_1, j_2, \ldots, j_n$ be the sequence of mice that the cat eats. Using the previous lemma inductively, it follows that the fastest way to meet these mice in the given order is to move with full speed in one direction between any two instants when the cat eats a mouse.

This argument also establishes the existence of an optimal solution for any given eating order $j_1, j_2, \ldots$ of the mice. Since there are only finitely many permutations $j_1, j_2, \ldots, j_n$ of the mice, we also obtain the existence of a globally optimum solution, which must look as claimed in the first part of the theorem.

To prove the last sentence of the theorem, consider a turning point such that every mouse that is met at this point is also met at some other time during the cat's path. By reassigning these mice to those other points as the points where the cat eats them, we get a turning point where the cat does not eat a mouse. By the above proof of the first part, this contradicts optimality. □

From the theorem we can draw the following conclusion: Suppose that the cat starts with a movement to the left. There is a sequence of $l \geq 1$ "left" mice $j_1, j_2, \ldots, j_l$ with $j_1 < j_2 < \cdots < j_l \leq m$ and a sequence of r "right" mice $k_1, k_2, \ldots, k_r$, with $l-1 \leq r \leq l$ and $k_1 > k_2 > \cdots > k_r \geq m+1$, so that the cat starts by going left until it meets j_1; there it turns to the right until it meets k_1, where it turns left to meet j_2, etc. The cat stops when it eats the final mouse: this is mouse j_l if $l = r + 1$ and

mouse k_r if $l = r$. In case the cat starts with a movement to the right an analogous statement holds. Moreover, each mouse j_i and k_i is met by the cat only once.

We can plot the path of the cat and the mice in the plane with the horizontal axis corresponding to position and the vertical axis corresponding to time, as in figure 1. The paths of the mice become straight lines and the cat moves on a zigzag curve. The paths of the mice j_i and k_i are "tangent" to the cat's path. It follows that the turning points form two convex chains from the starting point to the endpoint.

The following crucial observation is important for the algorithm:

LEMMA 2.3. *When the cat turns right at the mouse j_i and this is the only instant when it meets this mouse, the cat must have been to the left of all mice $j' < j_i$; and when the cat turns left at the mouse k_i and this is the only instant when it meets this mouse, the cat must have been to the right of all mice $k' > k_i$.*

Proof. If $v_{j'} < v_{j_i}$, i. e., mouse j' moves to the left relative to mouse j_i and mouse j' has not been eaten by the time mouse j_i is met, the cat has no chance to catch j' later without crossing the path of j_i again. The second statement of the lemma is analogous. □

In figure 1, the respective mice 1, 9, 3, and 8 fulfill the conditions of the above lemma. The lemma opens the possibility for a dynamic programming approach that considers all possibilities for the last two turning points of a partial solution of the cat's path.

3. The Algorithm. We define quantities $\vec{P}(j,k)$ for $0 \le j \le m$ and $m+1 \le k \le n$ and $\overleftarrow{P}(j,k)$ for $1 \le j \le m$ and $m+1 \le k \le n+1$ as follows:

DEFINITION 3.1. If $j \ge 1$, $\vec{P}(j,k)$ is the duration of the shortest zigzag path whose last movement is a right movement terminating when it meets mouse k, whose last turn was a right turn at the point where it meets mouse j, and which satisfies to the following conditions:

(i) The path meets mouse k only in the last point. For each turn on the path there is a mouse so that this is the only position where the cat meets this mouse.
(ii) The path has been to the left of every mouse j'' with $1 \le j'' < j$ at least once.
(iii) The path has been to the right of every mouse k'' with $k < k'' \le n$ at least once.

If no such path exists, we set $\vec{P}(i,k) = \infty$. $\vec{P}(0,k)$ is the duration of the shortest path consisting of a single right movement from the cat's starting position to the point where it meets mouse k, subject to condition (iii) from above. Again, we set $\vec{P}(0,k) = \infty$ if no such path exists.

The quantities $\overleftarrow{P}(j,k)$ for $1 \le j \le m$ and $m+1 \le k \le n+1$ are defined in the same way, except that the paths have to end with a left movement to the meeting point with mouse j, and the last turn was at mouse k (or the starting point, for $k = n+1$). □

By theorem 2.2 the optimal path is clearly one of the paths that are considered in $\vec{P}(j_l,k_r)$ or $\overleftarrow{P}(j_l,k_r)$, and every partial solution consisting of some initial zigzags is considered in $\vec{P}(j_i,k_i)$, $\overleftarrow{P}(j_i,k_i)$, $\vec{P}(j_i,k_{i\pm1})$, or $\overleftarrow{P}(j_i,k_{i\pm1})$, respectively.

To explain the dynamic programming recursion by which $\vec{P}(j,k)$ and $\overleftarrow{P}(j,k)$ can be computed we have to introduce some notation. Consider a path whose length is

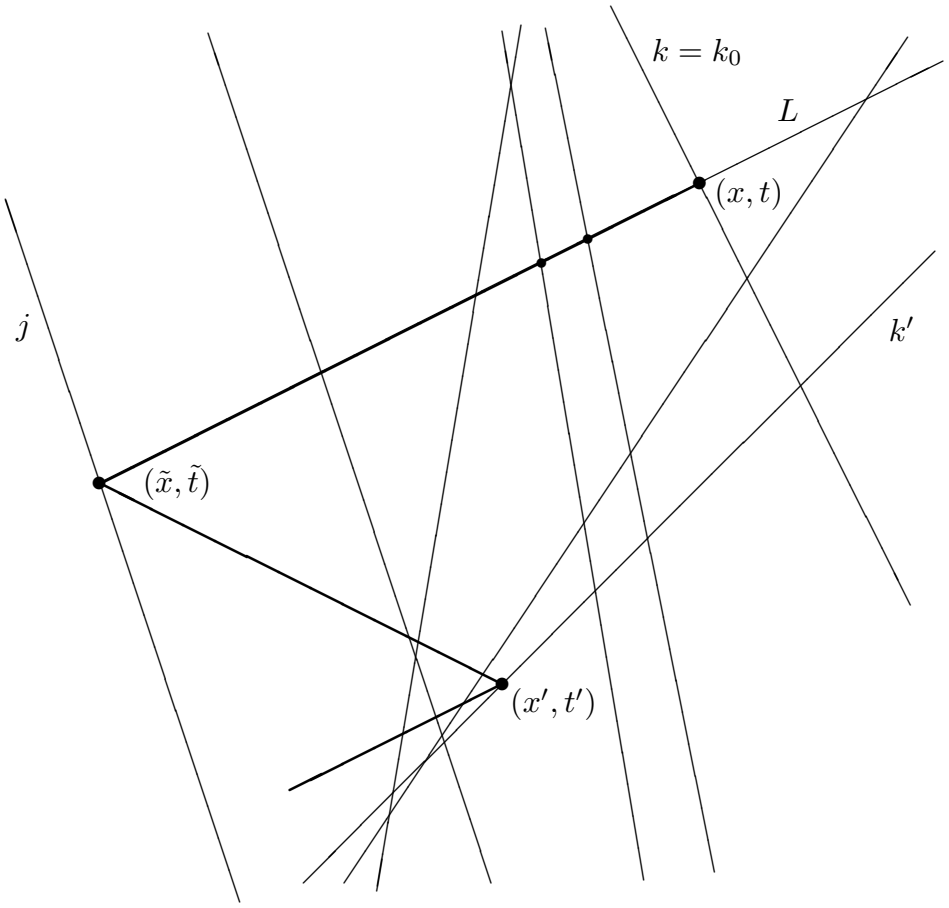


FIG. 2. *The dynamic programming recursion.*

stored in $\vec{P}(j,k)$, see figure 2. If we remove the last straight movement from mouse j to mouse k from this path, we get a path whose length $\tilde{t}$ is stored in $\overleftarrow{P}(j,k')$, where $k' > k$ is the mouse where the next-to-last turn occurred. For any k' we can immediately compute from $\tilde{t} = \overleftarrow{P}(j,k')$ the time t' and the position $x' = c_{k'} + v_{k'}t'$ where the candidate path in $\overleftarrow{P}(j,k')$ has met mouse k', and the time t and the position $x = c_k + v_k t$ where it will meet mouse k if it is extended by a right movement. Let us denote these by $t'(k')$, $x'(k')$, $t(k')$, and $x(k')$, respectively, regarding j and k as fixed for the moment. For $k' = n+1$, we will set $t'(k') = 0$ and $x'(k') = 0$ corresponding to the initial position.

If the path obtained by extending the path $\overleftarrow{P}(j,k')$ to the right until meeting mouse k fulfills all three conditions of definition 3.1, we will say that k' is *permissible* for $\vec{P}(j,k)$.

LEMMA 3.1. *The index k' is permissible for $\vec{P}(j,k)$ if the following conditions are satisfied:*

(a) *mouse k passes to the right of point $x'(k')$ at time $t'(k')$;*
(b) *if $k' \le n$ then mouse k' passes to the right of point $x(k')$ at time $t(k')$;*
(c) *all mice k'' with $k < k'' < k'$ pass either to the left of the turning point $x'(k')$ on line k' at time $t'(k')$ or to the left of the turning point $x(k')$ on line k at time $t(k')$.*

Proof. The resulting extended path ending at mouse k clearly fulfills condition (ii) above. For checking (i) it suffices to consider mice k' and k, which is done in conditions (a) and (b). Moreover, assuming condition (b) is satisfied, condition (iii)

is already fulfilled for all mice $k'' \geq k'$. Therefore, in order to be sure that we have a permissible candidate path for $\vec{P}(j,k)$ we explicitly have to test condition (iii) for the mice k'' between k' and k. However, their slope is between that of k and k'. Being to the left of some part of the path is therefore equivalent to being to the left of either the point $x'(k')$ at time $t'(k')$ or the point $x(k')$ at time $t(k')$ because these are the points where a supporting line with the slope of k, with the slope of k', or with any slope between these two slopes touches the path. So the test in condition (c) is sufficient. □

For $1 \leq j \leq m$ and $m+1 \leq k \leq n$ we have therefore:

$$\vec{P}(j,k) = \min\{\, t(k') \mid k < k' \leq n+1 \text{ and } k' \text{ is permissible for } \vec{P}(j,k) \,\}, \tag{1}$$

and with an analogous definition of permissibility for $\overleftarrow{P}(j,k)$,

$$\overleftarrow{P}(j,k) = \min\{\, t(j') \mid 0 \leq j' < j \text{ and } j' \text{ is permissible for } \overleftarrow{P}(j,k) \,\}. \tag{2}$$

To initialize the recursion, we set

$$\vec{P}(0,k) = \begin{cases} c_k/(1-v_k), & \text{if all mice } k'' \text{ with } k < k'' \leq n+1 \text{ pass} \\ & \quad \text{strictly to the left of point } c_k/(1-v_k) \\ & \quad \text{at time } c_k/(1-v_k), \\ \infty, & \text{otherwise,} \end{cases} \tag{3}$$

for $m+1 \leq k \leq n$, and

$$\overleftarrow{P}(j,n+1) = \begin{cases} -c_j/(1+v_j), & \text{if all mice } j'' \text{ with } 1 \leq j'' < j \text{ pass} \\ & \quad \text{strictly to the right of point } c_j/(1+v_j) \\ & \quad \text{at time } -c_j/(1+v_j), \\ \infty, & \text{otherwise,} \end{cases} \tag{4}$$

for $1 \leq j \leq m$.

To determine the optimal solution, consider the value t of $\vec{P}(j,k)$. As above, let $x = c_k + tv_k$ be the final position and let $\tilde{t}$ be the time and $\tilde{x} = c_{k'} + v_{k'}\tilde{t}$ be the position where the candidate path has met mouse j. For $j = 0$ we set $\tilde{x} = \tilde{t} = 0$. We say that $\vec{P}(j,k)$ is a *candidate for the optimum* if the following conditions are satisfied:

(a) all mice j'' with $j < j'' \leq m$ pass to the right of the point $\tilde{x}$ at time $\tilde{t}$; and

(b) all mice k'' with $m+1 \leq k'' < k$ pass to the left of the point x at time t.

(For $j = 0$ condition (a) means that m must be 0, i. e., $\vec{P}(0,k)$ can only be a candidate for the optimum if all mice are initially on the right side of the cat.) In an analogous way we define when $\overleftarrow{P}(j,k)$ is a candidate for the optimum.

LEMMA 3.2. *The optimal duration is given by*

$$\min\{\min_{j,k} \vec{P}(j,k),\ \min_{j,k} \overleftarrow{P}(j,k)\}, \tag{5}$$

where the minima are taken over all candidates for the optimum.

Proof. From theorem 2.2 and the discussion following it we know that the optimal path length is among the values considered in (5).

On the other hand, by lemma 3.1 each value $\overleftarrow{P}(j,k)$ and $\vec{P}(j,k)$ defined by the recursions (1–4) corresponds to some path on which the mouse has eaten all mice $j'' \leq j$ and $k'' \geq k$. By the discussion before lemma 3.2, any path whose length

is considered in (5) is a path of the cat where all mice have been eaten and thus a feasible solution. □

A straightforward evaluation of the recursions (1–5) using just the definitions and lemma 3.1 would take $O(n^4)$ steps. In order to reduce this to $O(n^3)$ we consider for each pair (j, k') and consider the possible contribution to all $\vec{P}(j,k)$ for which k' is permissible.

The algorithm looks then as follows:

begin
  initialize all $\vec{P}(j,k)$ and $\overleftarrow{P}(j,k)$ to ∞;
  compute $\vec{P}(0,k)$ for $m+1 \le k \le n$ by (3);
  compute $\overleftarrow{P}(j,n+1)$ for $1 \le j \le m$ by (4);
  for $j := 0$ **to** m **do**
    for $k := n+1$ **downto** $m+1$ **do**
      if $j \neq 0$ **then**
        $k' := k$;
(†)   **for** $k_0 := k'-1$ **downto** $m+1$ **do**
          compute $t'(k')$, $x'(k')$, $t(k')$, and $x(k')$ for $\vec{P}(j,k_0)$, using $\overleftarrow{P}(j,k')$;
          if k' is permissible for $\vec{P}(j,k_0)$
            then $\vec{P}(j,k_0) := \min\{\vec{P}(j,k_0),\, t(k')\}$;
        end for;
      end if;
      if $k \neq n+1$ **then**
        $j' := j$;
(‡)   **for** $j_0 := j'+1$ **to** m **do**
          compute $t'(j')$, $x'(j')$, $t(j')$, and $x(j')$ for $\overleftarrow{P}(j_0,k)$, using $\vec{P}(j',k)$;
          if j' is permissible for $\overleftarrow{P}(j_0,k)$
            then $\overleftarrow{P}(j_0,k) := \min\{\overleftarrow{P}(j_0,k),\, t(j')\}$;
        end for;
      end if;
    end for;
  end for;
  compute the optimum solution value using (5);
end;

THEOREM 3.3. *The fastest path for catching n moving objects on a line can be computed in $O(m(n-m)n) = O(n^3)$ time and $O(m(n-m)) = O(n^2)$ space, if m objects are initially on the left side and $n-m$ objects are initially on the right side.*

Proof. The initial sorting and renumbering of the mice can be done in $O(n \log n)$ time. Each of the $O(n)$ initialization equations (3) and (4) can be computed trivially in $O((m-n)^2)$ time or in $O(m^2)$ time, respectively. In the final expression (5), we have to check $O(m(n-m))$ expressions, whether they are candidates for the optimum, and each check can be carried out in $O(n)$ time, by just using the definition of candidates for the optimum. It remains to show how to carry out each of the $O(m(n-m))$ **for**-loops marked (†) and (‡) in linear time.

Consider an instance of the **for**-loop marked (†) for some fixed j and k'. We want to check permissibility for each value $k_0 > k'$ fast. We will use the fact that $\overleftarrow{P}(j,k')$ is fixed and therefore $t'(k')$ and $x'(k')$ on line k' and $\tilde{t} = \overleftarrow{P}(j,k')$ and $\tilde{x}$ on line j are

also fixed, see figure 2. For each k_0, $t(k')$ and $x(k')$ can be computed in constant time by intersecting the fixed line L which describes the right movement of the cat starting from $(\tilde{x}, \tilde{t})$ with the path of mouse k_0. Thus, conditions (a) and (b) of lemma 3.1 can be checked in constant time.

Condition (c) can be reformulated as follows: The point $(x(k'), t(k'))$ on the line corresponding to mouse k must lie on the right side of all those lines $x = c_{k''} + tv_{k''}$ for $k_0 < k'' < k'$ which do not pass to the left of the point $(x'(k'), t'(k'))$. If we intersect these lines with line L, we can also say that the point $(x(k'), t(k'))$ must lie to the right of these intersection points. We process the values k_0 in decreasing order. As we decrease k_0 we add more and more mice k'' that we have to check. The mice k'' passing to the left of the fixed point $(x'(k'), t'(k'))$ can be ignored. For the remaining mice, we intersect them with the line L and remember the right-most (highest) intersection point $(x_{\max}, t_{\max})$. Condition (c) is fulfilled if $(x(k'), t(k'))$ lies higher than $(x_{\max}, t_{\max})$, which can be checked in constant time. Formally, the first **for**-loop is carried out as follows:

```
    k' := k;
    compute t̃, x̃, t'(k') and x'(k') from t̃ = ←P(j, k');
    let L be the line of slope 1 through (t̃, x̃);
    t_max := t̃;   (* t_max is the highest intersection point with line L so far. *)
(†) for k_0 := k' − 1 downto m + 1 do
      compute t(k') and x(k');
      if the line of mouse k_0 passes strictly to the right of the point (x(k'), t(k'))
      then
        let (t, x) be the intersection of mouse k_0 with line L;
        if t ≥ t_max then
          (* k' is permissible for →P(j, k_0). *)
          →P(j, k_0) := min{→P(j, k_0), t(k')};
          t_max := t;
        end if;
      end if;
    end for;
```

This loop can clearly be carried out in $O(n)$ total time, and the time bound of the theorem follows.

By storing with each computed value $\vec{P}(j,k)$ and $\overleftarrow{P}(j,k)$ a pointer indicating the value where the minimum was achieved, it is possible to recover the optimal solution after the optimal value has be computed. □

4. Concluding remarks. In our solution we have minimized the time necessary to catch all mice. Instead of this, we can also minimize the total distance traveled, by just ignoring all mice that move towards the cat, and solving the problem of minimizing the time to catch the mice that run away. After all these mice have been caught, the cat can just remain where it is and wait for any remaining mice. It is easy to see that this solution minimizes the total distance.

It is also possible to consider mice which move faster than the cat. If such *fast mice* move away from the cat they obviously cannot be caught. Thus we only have to deal with fast mice moving towards the cat. The problem in this case is that lemma 2.1 no longer holds for fast mice. In fact, the cat will never make a turn

after eating a fast mouse, it may only terminate at such a mouse. If fast mice are approaching both from the left and from the right, there is a point P when the last fast mouse from the left crosses the last fast mouse from the right. Clearly the cat cannot finish before this happens. Thus there is one other possibility for an optimal solution in addition to the paths characterized in theorem 2.2: After eating all slower mice the cat moves to point P and waits for the fast mice to arrive. Lemma 3.2 can easily be modified to take this into account.

The problem considered in this paper can obviously be generalized in several ways, by considering variable speeds or mice moving in other spaces than the line, for example on a circle or in a graph. We can always take time as a second axis and plot the path of the mice and the cat in the plane. This eliminates the time component and transforms the problems into purely geometric problems. Thus all these problems are instances of the following general problem type:

> Given a set of objects, find the best curve of a civen class that meets all objects.

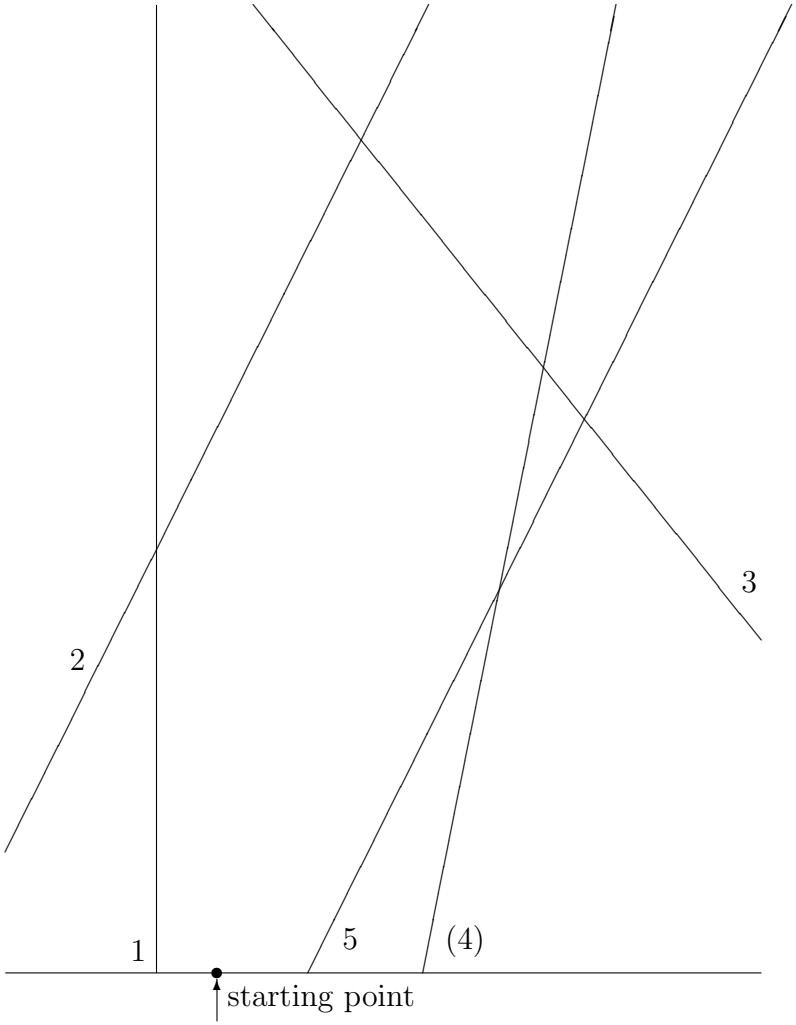
This class of problems encompasses such diverse problems as the Traveling Salesman Problem and the smallest enclosing circle. I believe that many more interesting problems can be formulated in this framework.

Acknowledgements. I thank Mordecai Golin for posing the problem, and I thank Danny Chen and Mikhail Atallah for helpful discussions. I also thank the Leonardo Fibonacci Institute in Trento for the hospitality during my stay there.

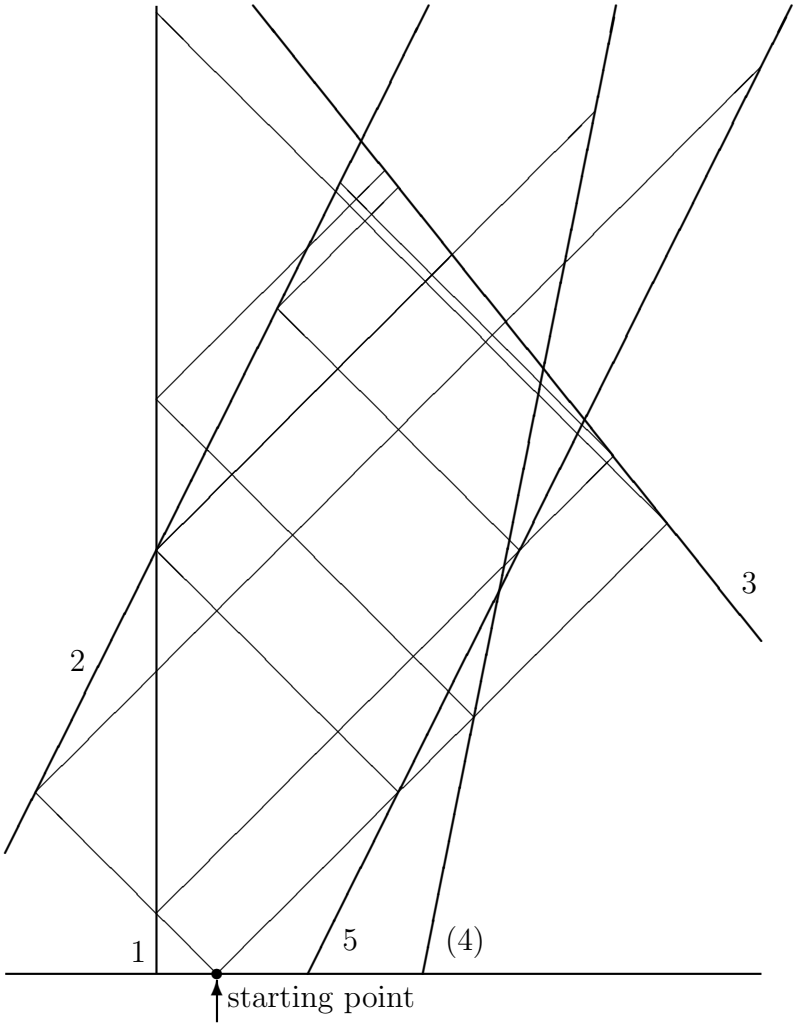
Addendum to

"The Traveling Salesman Problem for Moving Points on a Line"

Here is an example where you can try out my algorithm:



Here are some 45°-lines that help you to evaluate the different paths: (For hand calculations, it is easier to just apply the recursions (1)–(5) directly.)



If you want to get some more insight you might also try to solve the problem without mouse 4. Send your entry with your guess of the correct solution to `rote@opt.math.tu-graz.ac.at`.