LINEAR-TIME APPROXIMATION ALGORITHMS FOR FINDING THE MINIMUM-WEIGHT PERFECT MATCHING ON A PLANE

Masao IRI, Kazuo MUROTA and Shouichi MATSUI
Department of Mathematical Engineering and Instrumentation Physics, University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo, Japan 113

Received 20 October 1980; revised version received 20 February 1981

Approximation algorithms, minimum-weight matching, computational complexity, graphics

1. Introduction

We consider the problem of determining the minimum-weight perfect matching of n (even) points on a plane, i.e., determining how to match the n points in pairs so as to minimize the sum of the distances between the matched points.

This problem is of fundamental importance in finding the optimal sequence of drawing edges of a connected graph by a mechanical plotter; as is easily confirmed, the wasted plotter-pen movement is minimized by finding a minimum-weight perfect matching of the vertices of an odd degree, adding the edges between the matched pairs to the original graph and traversing an Eulerian path on the extended graph, where the added edges are to be traversed with the pen off the paper [1,3].

The algorithm [2] which exactly solves this problem in $O(n^3)$ time seems to be too complicated from the practical point of view. Even approximation algorithms of $O(n^2)$ or $O(n \log n)$ [4] would not be satisfactory for the application to plotters, since an Eulerian path can be found in time linear in the number of edges. In fact, our Monte Carlo experiments for 1400 problems with 32 to 2048 points have shown that the strip algorithm of $O(n \log n)$ [4] requires 3 to 10 times as much computing time as our linear-time algorithms do.

In this paper, linear-time approximation algorithms are proposed for the matching problem on a unit square and the worst-case performance is analyzed theoretically. The quality of an approximate solution is measured by the *absolute cost* of the matching, i.e., the sum of the distances between the paired points, and not by the ratio of the cost to that of the exact optimal solution. As the distance, not only the L_2-distance (Euclidean distance) but also the L_∞-distance (maximum norm) is considered, the latter being more appropriate when the running time of a mechanical plotter is in question.

2. Algorithm

To be specific, we describe here 'the serpentine algorithm with tour'. It is among the several linear-time algorithms we have considered, and may be regarded as a linear-time variant of the strip algorithm in [4].

We partition the unit square into k^2 square cells by dividing each side into k equal parts. Each point belongs to one of the k^2 cells. We determine the cell to which a point belongs by multiplying the coordinates (abscissa and ordinate) of the point by k and then truncating the fractional parts off. The cells are ordered in a prescribed order, called 'the serpentine cell order', as shown in Fig. 1. We number the n points to form a sequence which is consistent with the cell order. This means that points in one and the same cell may *arbitrarily* be ordered among themselves, but

0020-0190/81/0000–0000/$02.50 © 1981 North-Holland

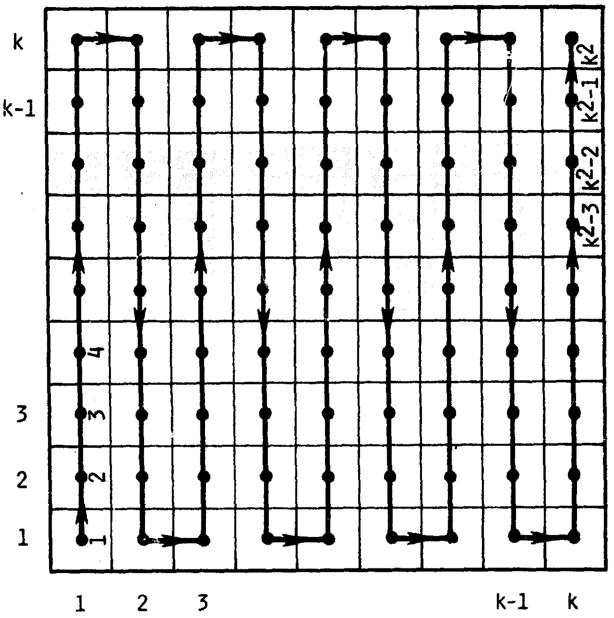

Fig. 1. The serpentine cell order.

points in different cells must be ordered consistently with the cell order. By visiting all the points in this order, we construct a travelling-salesman tour, which obviously contains two perfect matchings for n even. We adopt the one with less cost for the solution.

Provided k is taken as large in order as $\sqrt{n}$, it is evident that the complexity of this algorithm is O(n) both in time and in space.

Input. Real array X[1 : n], Y[1 : n]. (X[p], Y[p] = coordinate of the pth point in the unit square.

Output. Integer array PAIR[1 : n]. PAIR[p] = index of the point matched with the pth point.

Working space. Integer array CELL[1 : k, 1 : k]. CELL[i, j] = index of the first point in cell (i, j) (or 0 if the cell is empty). At Stage (1), PAIR is used for the lists of points belonging to respective cells; at Stage (2), those lists are concatenated into a single circular sequence (tour) according to the cell order; at Stage (3), the output information is made up in PAIR.

Notes. The pair of functions (XINDEX(c), YINDEX(c)) gives the index pair (i, j) of the cell which is the cth in the prescribed cell order. Parameter α is to be determined in Section 3.

Algorithm

```
begin
  initialize: k := $\alpha\lfloor\sqrt{n}\rfloor$; set each CELL[i, j] := 0,
    (i, j = 1, ..., k);
  (1) for p := 1 to n do
        [[i := $\lceil$X[p] * k$\rceil$; j := $\lceil$Y[p] * k$\rceil$;
          PAIR[p] := CELL[i, j]; CELL[i, j] := p;]]
  (2) $p_0$ := 0;
      for c := 1 to $k^2$ do
        [[q := CELL[XINDEX(c), YINDEX(c)];
          if q ≠ 0 then [[if $p_0$ = 0 then $p_0$ := q
                            else PAIR[p] := q;
                          p := q; while PAIR[p] ≠ 0
                            do p := PAIR[p];]]]]
      PAIR[p] := $p_0$;
  (3) cost1 := cost 2 := 0; p := 1; q := PAIR[p];
      for e := 1 to n do
        [[d := distance between (X[p], Y[p]) and
            (X[q], Y[q]);
          if e is odd then [[cost1 := cost1 + d;
                             p := PAIR[q]]]
                      else [[cost2 := cost2 + d;
                             q := PAIR[p]]]]]
      if cost1 < cost2 then p := 1 else p := PAIR[1];
      for e := 1 to n/2 do
        [[q := PAIR[p]; r := PAIR[q]; PAIR[q] := p;
          p := r]]
end
```

3. Worst-case analysis

For a fixed algorithm with k^2 cells, let $\overline{M}_n(k)$ be the supremum of the costs of solutions over all possible configurations of n given points in the unit square, and put

$$\overline{\mu}(\alpha) = \lim_{n\to\infty} \overline{M}_n(\alpha\sqrt{n})/\sqrt{n},$$

$$\overline{\mu}_0 = \overline{\mu}(\overline{\alpha}_0) = \min_{\alpha} \overline{\mu}(\alpha).$$

In the following, the *cell-distance* of two cells with respect to the serpentine cell order will mean the difference of the cell numbers in that ordering; e.g., two consecutive cells are at cell-distance 1.

An upper bound for $\overline{M}_n(k)$ of the serpentine algorithm with tour can be obtained by means of the linear programme with variables n_j, where n_j is the number of edges in a travelling-salesman tour which connect points in two cells at cell-distance $j - 1$ in the serpentine cell order; in particular, n_1 denotes the number of edges within one and the same cells.

In order to obtain an upper bound for $\overline{M}_n(k)$ in the L_∞-distance, we consider the primal programme:

$$\text{maximize} \quad f \equiv \sum_{j=1} c_j n_j/2,$$

$$\text{subject to} \quad \sum_{j=1} n_j = n,$$

$$\sum_{j=1} (j-1) n_j \leqslant k^2,$$

$$n_j \geqslant 0,$$

where $c_j = j/k$ is an upper bound for the L_∞-distance between two points contained in two cells at cell-distance $j - 1$. The first constraint is concerned with the number of edges in a travelling-salesman tour through n points, and the second with the number of cells. Obviously, the value $\hat{f}$ of the objective function of the optimal solution to the primal programme gives an upper bound for $\overline{M}_n(k)$.

The dual programme is then

$$\text{minimize} \quad g \equiv nx + k^2 y,$$

$$\text{subject to} \quad x + (j-1)y \geqslant c_j/2 \quad (j = 1, 2, ...),$$
$$y \geqslant 0.$$

As is well known, the value g of the objective function of any feasible solution of the dual programme is not smaller than $\hat{f}$, i.e.,

$$\overline{M}_n(k) \leqslant \hat{f} \leqslant g,$$

so that g can be used as an upper bound for $\overline{M}_n(k)$. In particular, the minimum value $\hat{g}$ of the dual programme gives a good upper bound, which, in the present case, turns out to be asymptotically tight.

In fact, the optimal solution of the dual programme is easily known to be

$$(\hat{x}, \hat{y}) = \left(\frac{1}{2k}, \frac{1}{2k}\right),$$

where

$$\hat{x} + (j-1)\hat{y} = c_j/2 \quad \text{for all } j = 1, 2, ...,$$

and

$$\hat{g} = n/(2k) + k/2.$$

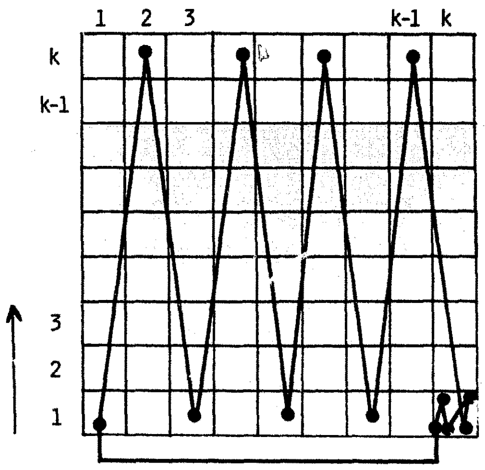

Fig. 2. The asymptotic worst-case for the serpentine algorithm with tour (L_∞-distance).

Hence we have

$$\overline{M}_n(k) \leqslant n/(2k) + k/2. \tag{3.1}$$

The optimal solution of the primal programme is degenerate; for example, we have

$$\hat{n}_1 = n - k, \qquad \hat{n}_{k+1} = k,$$
$$\hat{n}_j = 0 \quad \text{for the other } j\text{'s}, \tag{3.2}$$

which gives

$$\hat{f} = n/(2k) + k/2 = \hat{g}.$$

There is a configuration of given points, such as shown in Fig. 2, for which the cost of the matching obtained by the algorithm can be as large as $(k + (n-k)/k)/2$, since there are about $k/2$ edges of L_∞-length 1 and about $(n-k)/2$ edges of L_∞-length $1/k$ in the resulting matching. This value is asymptotically equal to $\hat{f} = \hat{g}$ as $k \to \infty$. Therefore, the upper bound (3.1) is asymptotically tight; i.e., we have in the L_∞-distance

$$\overline{\mu}(\alpha) = \frac{1}{2\alpha} + \frac{\alpha}{2}$$

and

$$\overline{\mu}_0 = 1 \quad \text{for } \overline{\alpha}_0 = 1.$$

The worst-case analysis above suggests that a better algorithm may be obtained by modifying the cell order so as to make c_j increase as slowly as possible as j

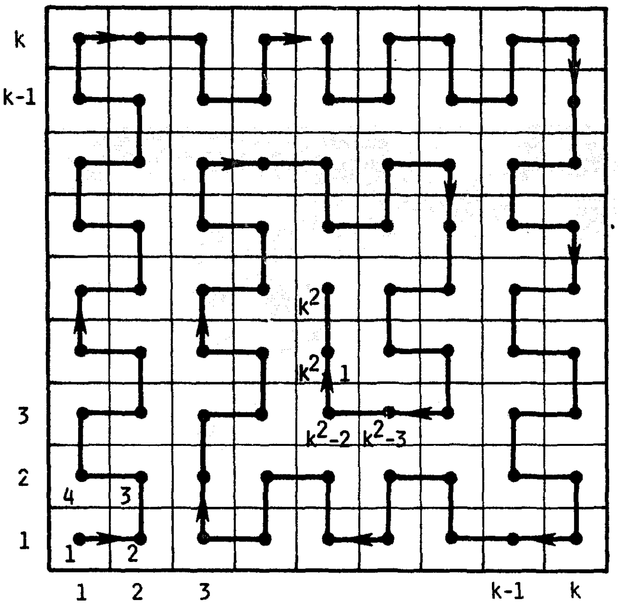

Fig. 3. The spiral rack cell order (k: odd).

increases. Thus we are led to another algorithm with 'the spiral rack cell order' shown in Fig. 3, in place of 'the serpentine cell order' in the serpentine algorithm with tour. The algorithm thus obtained is named 'the spiral rack algorithm with tour'. The worst-case analysis for this cell order may be performed in a way similar to the above (but somewhat more complicated), and we actually have an improved bound in the L_∞-distance:

$$\bar{\mu}(\alpha) = \min\left(\frac{1}{2\alpha} + \frac{\alpha}{2}, \frac{3}{4\alpha} + \frac{\alpha}{4}\right),$$

$$\bar{\mu}_0 = \sqrt{3}/2 \doteqdot 0.866 \quad \text{for } \bar{\alpha}_0 = \sqrt{3}.$$

The case of the L_2-distance can be treated quite similarly by modifying the expressions for c_j. For example, for the spiral rack algorithm with tour we have

$$\bar{\mu}_0 \leqslant \sqrt{(2\sqrt{5} - 1)(\sqrt{13} + 1 - \sqrt{5})/8} \doteqdot 1.014,$$

in the L_2-distance. This bound would be favourably compared with the known best upper bound 0.707 for an algorithm of complexity O(n log n) [4].

4. Concluding remarks

We proposed *linear-time* approximation algorithms for finding the minimum-weight perfect matching on a plane whose performances are nearly as good as those of the algorithms of higher complexity, and analysed the worst-case behaviour of the proposed algorithms by means of linear programming. From the practical point of view, the average-case behaviours of the algorithms are more important. The behaviours of the proposed algorithms are closer to those of the algorithms of higher complexity in the average case than in the worst case. They will be reported soon.

Acknowledgement

The authors thank Prof. Z. Galil of Tel Aviv University and Prof. D. Avis of McGill University for providing them with valuable materials concerning the subject and for stimulating discussions.

References

[1] M. Iri and A. Taguchi, The determination of the pen-movement of an XY-plotter and its computational complexity (in Japanese), Proc. Spring Conference of the Operations Research Society of Japan 8 (1980) 204–205.
[2] E.L. Lawler, Combinatorial Optimization – Networks and Matroids (Holt, Rinehart and Winston, New York, 1976).
[3] E.M. Reingold and R.E. Tarjan, On a greedy heuristic for complete matching (1978).
[4] K.J. Supowit, D.A. Plaisted and E.M. Reingold, Heuristics for weighted perfect matching, Proc. 12th Annual ACM Symposium on Theory of Computing (1980) 398–414.