Proceedings of ALGORITMY 2009
pp. 402–411

# A MONTE CARLO SOLUTION TO THE MINIMAL EUCLIDEAN MATCHING*

J. SKÁLA , I. KOLINGEROVÁ , AND J. HYKA†

**Abstract.** We present an approximate algorithm for the minimal Euclidean matching problem. It is intended as an alternative to the Edmonds optimal algorithm which we find too intricate. Our approach builds on the Kuhn-Munkres algorithm for matching in a bipartite graph and uses a Monte Carlo method to adapt the algorithm for general graphs.

We first document a few unsuccessful attempts to show that the matching problem is difficult to solve by a simple heuristics such as the greedy approach. Then the Monte Carlo modification of the Kuhn-Munkres algorithm is proposed. Finally, we present experiments that show the algorithm running time and describe how the approximation of the minimal Euclidean matching is improved over successive iterations of the Monte Carlo method.

**Key words.** minimal Euclidean matching, bipartite matching, greedy algorithm, Kuhn-Munkres algorithm, Monte Carlo method

**AMS subject classifications.** 68R10, 05C70, 65C05, 68W25

**1. Introduction.** Originally, we were working on a clustering algorithm to develop a method for manipulation with large geometric data. The clustering made quite a good impression and several people were interested in it. One of the inspiring ideas was to use the clustering for space partitioning for ray tracing acceleration. However, the partitioning is required to be binary such as in the $k$D-tree which is often used in such situations. It turned out that adapting the clustering algorithm to such demand would require substantial changes so we decided to develop another method.

The task is: Given a set of vertices, group all of them into pairs so that the sum of distances between the paired vertices is minimal. An example of such pairing is shown in Figure 1.1. After a brief research we identified the problem as the minimum

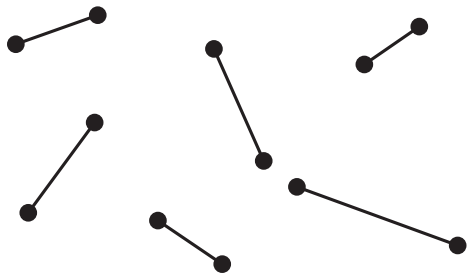

FIG. 1.1. *Example of a minimal Euclidean matching.*

Euclidean matching. It is a problem of graph theory, formally stated as follows. Given

*Work has been supported by the Ministry of Education, Youth and Sports of the Czech Republic under the research program LC-06008 (Centre for Computer Graphics).

†Department of Computer Science, University of West Bohemia, Pilsen; jskala@kiv.zcu.cz, kolinger@kiv.zcu.cz, hyka.jan@atlas.cz

a set of $2N$ vertices corresponding to nodes of a complete graph with edge weights equal to Euclidean distances between the respective vertices, find the minimum weight perfect matching. A *perfect matching* is a matching where every vertex is incident to exactly one edge of the matching.

**2. Practical application of the proposed method.** To familiarise the reader with the intended application of the proposed method, we briefly describe its composition with the clustering algorithm.

Due to the high complexity of the method, it is not practically possible to run it directly on large data. We use a clustering technique to process the data in pieces. First, clusters are identified in the data. The matching is then constructed within each cluster separately. Finally, a matching among particular clusters is constructed to merge the results together. The output is a binary tree that defines the space partitioning for the ray tracing. Our first implementation is capable of processing 5 million vertices in several hours. This method is to be described in detail in another paper.

**3. Related work.** This section gives a survey of methods related to space partitioning, clustering and the matching problem.

**3.1. Traditional space partitioning.** Traditional methods such as a quadtree [5] or an octree use a top-down approach, i.e., the data are recursively divided until a desired degree of partitioning is achieved. Such methods are rather simple and fast, nevertheless, they can lead to unnatural results as illustrated in Figure 3.1. The left frame shows a partitioning created by a common quadtree. You can see several groups of nearby points spread among different quads. The right frame shows a bottom-up partitioning done by a hierarchical matching. Points were matched two by two; the resulting pairs were matched again and again until the tree was created. Bolder lines in the figure correspond to higher levels of the tree. Other popular space partitioning

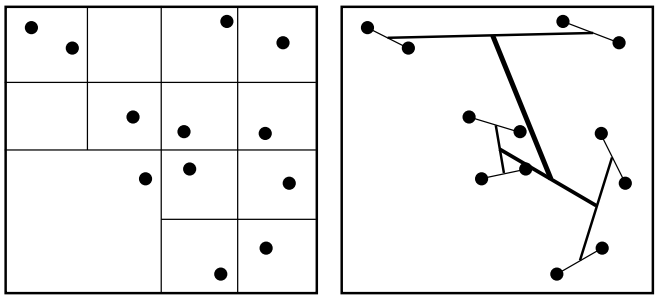

FIG. 3.1. *Example of partitioning the same data with a quadtree and with a hierarchical matching.*

approaches include a $k$D-tree [1] or a BSP (Binary Space Partitioning) [6, 3]. All of these methods are based on the top-down approach.

The top-down approach usually strives for a balanced division, e.g., all cells should have approximately the same number of points, or cells are subdivided into equally sized parts. As seen in Figure 3.1, this can result in some groups of nearby points being split apart. The bottom-up approach can produce more natural partitioning because it is based on putting nearby points together, so points belonging to some logical entity are likely to be together. The cost for that is usually a higher algorithm complexity along with a longer running time.

**3.2. Clustering algorithms.** The idea to solve the Euclidean matching originated in a special assignment for our clustering algorithm. A large amount of various clustering approaches is known, see for example [8]. The problem is that they focus on finding bigger groups rather than just pairs in the data.

The very popular $k$-means algorithm [11] could be guided to make pairs by setting $k$ – the desired number of clusters – to $N/2$, assuming $N$ is the number of vertices. However, this does not guarantee to create pairs (i.e., clusters with exactly two members). Much more likely is that some clusters will contain just a single point while others will be more populated.

Our clustering uses the facility location approach. There are several methods how to construct the clustering, see [14] for an overview. Unfortunately, facility location is quite unfeasible for matching since there is no direct control over the number of elements in a cluster.

As far as we know there is no clustering algorithm that can be effectively used or adapted to solve the minimal Euclidean matching.

**3.3. Nearest neighbours.** There are algorithms, often based on Voronoi diagrams [2], for finding nearest neighbours among set of points. Results can be used for greedy approaches such as those described in Section 4.1. However, the nearest neighbour is computed independently for every particular point. The Euclidean matching is a global optimisation problem and greedy algorithms do not work well, see Section 4.1 for further details.

**3.4. Matching algorithms.** A polynomial algorithm for the minimal Euclidean matching is known due to Edmonds [4]. The time complexity is $\mathcal{O}(N^4)$. Gabow [7] proposed an implementation running in $\mathcal{O}(N^3)$ time. The algorithm was designed for arbitrary graphs and is rather complex. Our task supposes a complete graph which is a simplifying condition, so we decided to develop a simpler approximate solution.

The matching method we finally use is the Hungarian algorithm developed by Kuhn [9] and adapted by Munkres [13]. The algorithm is known as the Kuhn-Munkres algorithm and it works only for bipartite graphs. A *bipartite graph* is a graph whose vertices can be divided into two disjoint sets such that every edge connects a vertex in one set to a vertex in the other set. The time complexity of the Kuhn-Munkres algorithm is $\mathcal{O}(N^3)$.

The algorithm uses a special vertex labelling to select a subset of edges and create a factor of the graph. A *factor* of a graph is a subgraph with the same vertices and a subset of edges. The algorithm starts with an arbitrary matching. Then it repeatedly attempts to enlarge the matching by swapping suitable edges that belong and do not belong to the matching. If the matching cannot be further enlarged but it is still not a perfect matching, the vertex labelling is adapted, a new factor is constructed and the algorithm starts over. When the matching is a perfect matching in the factor, it is also the minimal weight matching in the original graph.

For the exact formulation of the Kuhn-Munkres algorithm we need a few definitions. Let $G = (X \cup Y, E)$ be a weighted bipartite graph, where $X$ and $Y$ are equally sized sets of vertices and $E$ is a set of edges. We are looking for a matching $M$ from $X$ to $Y$.

Given the graph $G$, a *neighbourhood* of vertex set $S \subset X$ is a vertex set $N_G(S) = \{y \in Y | \exists x \in S \text{ so that } \{x, y\} \in E\}$.

Let $w_{xy}$ be the weight of edge $\{x, y\}$. A *feasible vertex labelling* is a function $l$

such that

$$l(x) + l(y) \geq w_{xy} \quad \forall \{x, y\} \in E \tag{3.1}$$

An *equality subgraph* $G_l$ is a factor of $G$ which contains only those edges where $l(x) + l(y) = w_{xy}$.

Let $G$ be a graph, $M$ be a matching in $G$ and $P$ be a path in $G$. We call $P$ an *$M$-alternating path* if its edges belong alternatively to the matching and not to the matching. Matching created by *flipping $M$ along $P$* is a matching created from $M$ by removing all edges in $P$ that belonged to $M$ and adding all edges in $P$ that did not belong to $M$.

Now, if $l$ is a feasible vertex labelling and $M$ is a perfect matching in $G_l$, then $M$ is the minimal weight matching in $G$. See for example [10] for a proof.

The Kuhn-Munkres algorithm can now be summarised as follows. Start with an arbitrary feasible vertex labelling $l$ and an arbitrary matching $M$ in $G_l$.

1. If $M$ is a perfect matching in $G_l$, it is the minimal weight matching in $G$. Stop. Otherwise, there is some unmatched $x \in X$. Assign $S := \{x\}$ and $T := \emptyset$.
2. If $N_{G_l}(S) = T$, go to 4. Otherwise, choose $y \in N_{G_l}(S) \setminus T$.
3. If $y$ is matched in $M$, say with $z \in X$, set $S := S \cup \{z\}$ and $T := T \cup \{y\}$, and go to 2. Otherwise, there will be an $M$-alternating path $P$ from $x$ to $y$. Create a larger matching $M'$ by flipping $M$ along $P$. Set $M := M'$ and go to 1.
4. Compute

$$\alpha_l = \min_{x \in S, y \notin T} \{l(x) + l(y) - w_{xy}\} \tag{3.2}$$

and construct a new feasible labelling $l'$ by

$$l'(v) = \begin{cases} l(v) - \alpha_l & \text{for } v \in S, \\ l(v) + \alpha_l & \text{for } v \in T, \\ l(v) & \text{otherwise.} \end{cases} \tag{3.3}$$

Set $l := l'$ and $G_l := G_{l'}$, choose $y \in N_{G_l}(S) \setminus T$ and go to 3.

As stated at the beginning of this section, the Kuhn-Munkres algorithm is applicable only for bipartite graphs. Our task is to find a matching in a complete graph so we must either adapt the algorithm or develop another one. Section 4 presents our first attempt with a simple greedy approach, followed by two adaptations of the Kuhn-Munkres algorithm.

**4. The proposed matching algorithm.** We tried several approaches to solve the Euclidean matching. The greedy edge removing and the matching in a bipartite graph did not end in a complete success. The proposed randomised partitioning & matching is a reliable method and gives satisfactory results.

**4.1. Greedy edge removing.** This is a very simple approach. Greedy algorithms work in such a way that they successively accept the option which is currently the best. Thus they always pursue a local optimum. However, the solution may not be optimal from a global perspective.

The first method that comes to one's mind is to find the closest pair of vertices, declare them as a pair, and continue with the remaining vertices until all of them have been paired. This approach can produce absolutely wrong results in some situations

as seen for example in Figure 4.1. On top there are the input vertices and their mutual distances. In the middle there is a matching created by the greedy algorithm; the sum of distances is 40. At the bottom there is the minimal Euclidean matching with a sum of distances of 22.

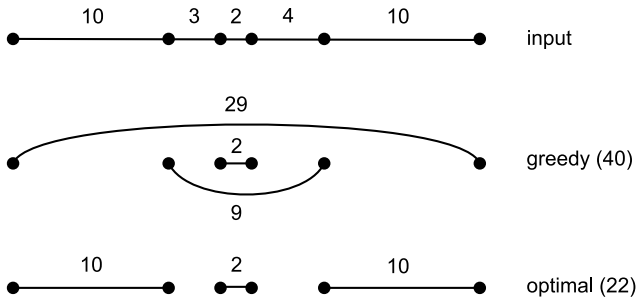


FIG. 4.1. *A failure of the simple greedy algorithm.*

Our greedy algorithm works in an opposite way. It starts with a complete graph and successively discards the longest edge. If any edge is the last one incident to any vertex, the edge is fixed, i.e., it is declared as a part of the matching and is never removed. The algorithm terminates when all edges have been either removed or fixed. The hope is that the resulting pairs will be a good approximation of the minimum Euclidean matching.

After several experiments it turned out that the algorithm can often remove an edge so that the underlying graph breaks into more components. If any such component is odd, i.e., it has an odd number of vertices, it is impossible to achieve a perfect matching any more. So we added a test which – before removing any edge – checks whether the graph would break into odd components. If so, the edge is fixed rather than removed. The algorithm can be stated as follows:

1. Start with a complete graph.
2. Find the longest edge $e$.
3. If discarding $e$ would create an odd component, make $e$ a pair; otherwise discard $e$.
4. If not all edges removed or fixed, go to 2.

Note that the step 3 intrinsically ensures that the last edge incident to a vertex will be fixed. Because if such an edge would be removed, the vertex would be left alone, thus forming an odd component.

An example of the algorithm progress is shown in Figure 4.2. The algorithm performs well in some situations. For the example used in Figure 4.1, it successfully creates the minimal Euclidean matching.

However, the algorithm may still run in a situation, from where a perfect matching can not be achieved. An example is shown in Figure 4.3. The leftmost edge in Figure 4.3 middle is the longest and thus should be removed. No odd component will emerge, so the edge is actually removed. From the result it is, however, obvious that a perfect matching cannot be achieved. Sure it would be possible to add more tests, detect such situations and avoid them. But the tests here are getting rather intricate, other problems are likely to occur and there is no guarantee of final success. We decided to leave the greedy approach.

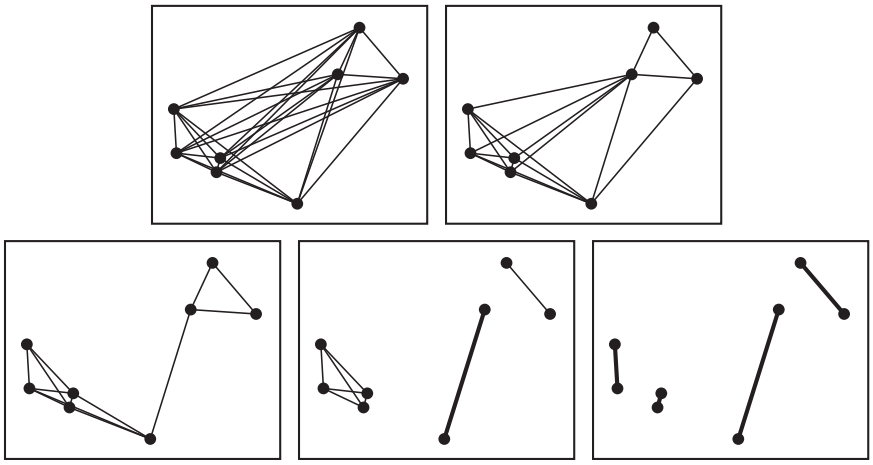

FIG. 4.2. *The progress of the greedy edge discarding algorithm.*

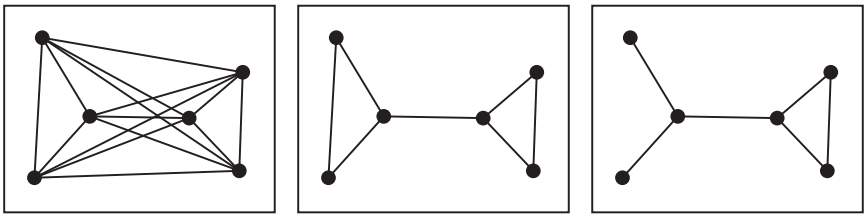

FIG. 4.3. *The greedy algorithm gets stuck after removing the leftmost edge.*

**4.2. Matching in a bipartite graph.** As a second attempt we addressed the problem as a matching in a bipartite graph. The minimal weight matching in a bipartite graph is relatively easy to solve by the Kuhn-Munkres algorithm [9, 13]. The problem is that we generally do not have a bipartite graph.

Our idea was to construct a bipartite graph $G = (X \cup Y, E)$ by putting all the input vertices into the first set $X = \{a, b, c, \ldots\}$ and then duplicate them into the second set $Y = \{a', b', c', \ldots\}$. We then create edges (i.e., allow pairing) from each vertex in one partition to all vertices in the other partition, except to the duplicate image of the vertex. That is, we disallow pairs of type $[x, x']$. The resulting graph will look as in Figure 4.4. We then apply the Kuhn-Munkres algorithm to solve the

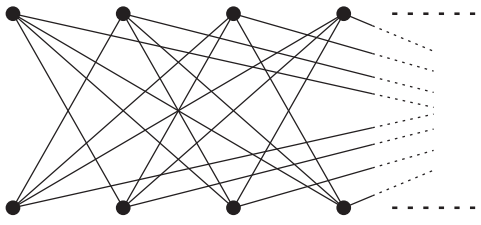

FIG. 4.4. *The bipartite graph where the matching is constructed.*

matching. The matching in the original general graph is easily extracted from the bipartite matching by simply eliminating duplicates such as $[a, b']$ and $[b, a']$ which becomes just $[a, b]$.

The problem is that the result in the bipartite graph is not always symmetric. If $[a, b']$ is a pair, then not necessarily [b,a'] is a pair too. An example of such a situation can be seen in Figure 4.5. We did not find a solution for that, so we tried yet another approach.

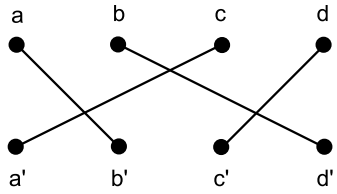


FIG. 4.5. *An unfeasible assignment resulting from the Kuhn-Munkres algorithm.*

**4.3. Randomised partitioning & matching.** Despite the previous bad success, we stayed with the Kuhn-Munkres algorithm and tried a different approach to construct the two sets $X$ and $Y$. We developed a Monte Carlo method [12]. It is a stochastic technique for solving problems for which an analytical solution is unknown or is too complex. In general, a Monte Carlo method performs statistical simulations using random numbers. To be more specific, in our case it repeatedly generates random possible solutions and evaluates their cost. At the end, the best solution is selected.

Each iteration of the Monte Carlo method starts by randomly distributing all the vertices into two equally sized sets. Each vertex in one set is connected to all vertices in the other set. The matching is then constructed by the Kuhn-Munkres algorithm without any problem. This process is run repeatedly; see Section 5 for experiments with the number of iterations. At the end, the solution with the lowest sum of pairwise distances is accepted. Figure 4.6 shows several instances of the matching with the best one framed in bold. The number at lower right shows the sum of distances.

The algorithm can be formally stated as follows. We are looking for the minimal Euclidean matching $M_{\min}$.

1. Initialise the minimal cost $c_{\min}$ to positive infinity.
2. Randomly distribute the vertices into two equally sized sets $X$ and $Y$. Make the set of edges $E = \{\{x, y\} | x \in X, y \in Y\}$, i.e., make an edge from every vertex $x \in X$ to every vertex $y \in Y$.
3. In the bipartite graph $G = (X \cup Y, E)$ construct matching $M$ by the Kuhn-Munkres algorithm. Let the cost of the matching be $c$.
4. If $c < c_{\min}$, set $c_{\min} := c$ and $M_{\min} := M$.
5. While a stopping condition has not been met, go to 2.
6. Output $M_{\min}$.

**5. Experiments.** The experiments discussed in this section were made on random data with uniform distribution. The program was written in C# for the .NET Framework 2.0. The testing computer was a Pentium 4 3.2 GHz with 2 GB RAM, running Windows XP.

The time complexity of the Kuhn-Munkres algorithm is $\mathcal{O}(N^3)$. This was proven in practice as seen in Figure 5.1. Unfortunately, we have no competitive program to

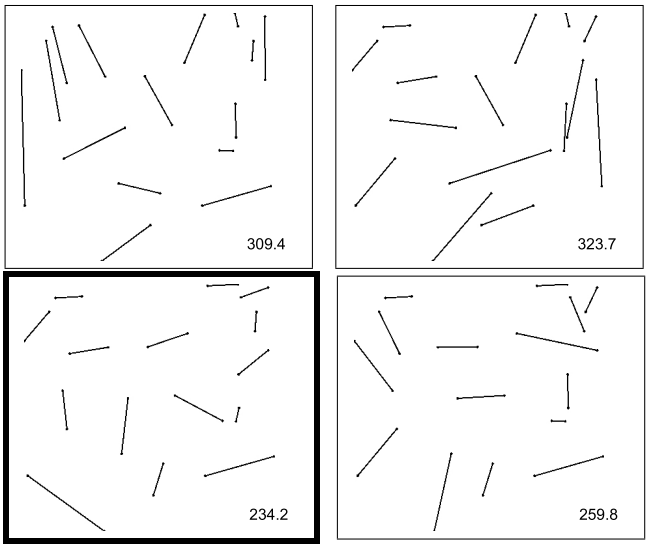


FIG. 4.6. *Example of several matching instances tried by the Monte Carlo method.*

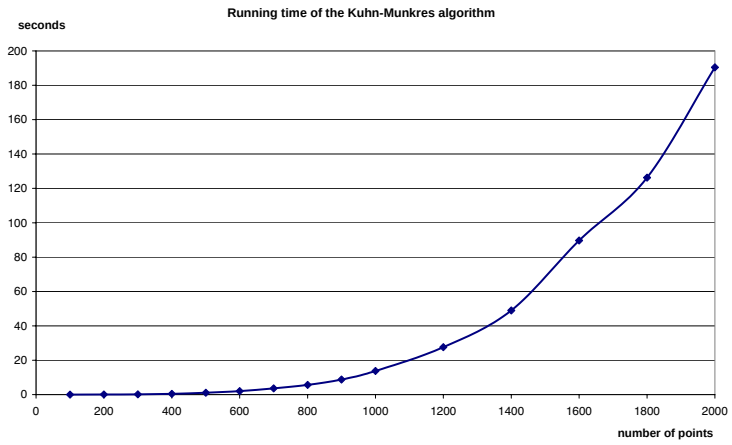


FIG. 5.1. *The running time of the Kuhn-Munkres algorithm.*

compare with. The optimal Edmonds algorithm is what we wanted to avoid due to its complex implementation. We did not find any other method either, so we developed our approximate solution which we believe is unique in this way.

Quite an unpleasant property of the Monte Carlo method is that it does not converge. Consecutive matchings generated by the algorithm have rather random cost. The process does not converge to the minimal cost matching. The graph in Figure 5.2 shows the first 100 iterations of the Monte Carlo algorithm for a data with 100 vertices.

So it is hard to find a stopping condition when the solution is close enough to the optimum. Our experiments show that $\mathcal{O}(N)$ iterations should be done, assuming $N$ is the number of input points. For practical applications, we recommend $N$ or $10N$ iterations. The improvements in later iterations are less significant. The graphs in

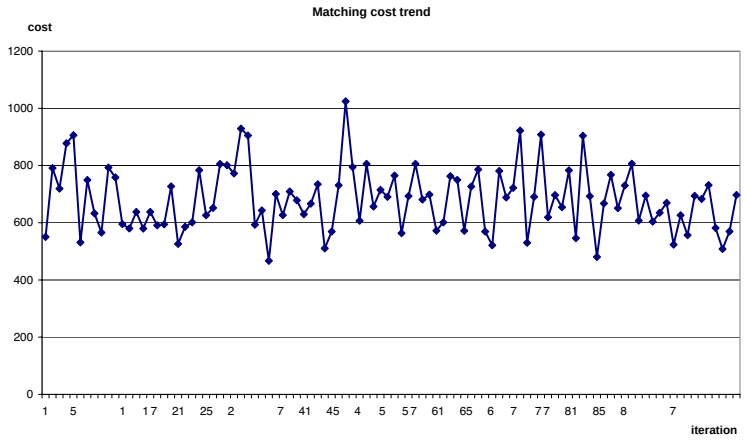


FIG. 5.2. *The trend of the matching cost over successive iterations of the Monte Carlo method.*

Figures 5.3 and 5.4 show the progress of improvements, i.e., decreases in the matching cost. The graphs show the results from three random data sets of 100 points after 10 000 iterations (blue, purple and brown line) and one data set of 500 points after 50 000 iterations (green line). Figure 5.3 shows how the minimal known cost decreases over the iterations. Figure 5.4 shows the decrease in percent of the previous minimal value. Please note the logarithmic scale on the $x$ axis in both graphs. The number of iterations should be determined also with respect to the time we can spend by running the algorithm; the more the better.

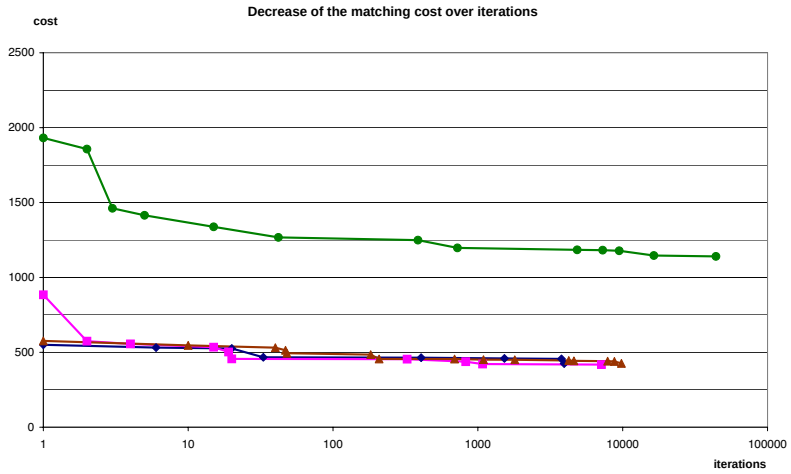


FIG. 5.3. *The trend of the minimal cost of matching over the iterations.*

**6. Conclusion.** We have introduced an alternative method to the optimal-but-complicated Edmonds algorithm for the minimal Euclidean matching. We discussed two unsuccessful attempts and then present a Monte Carlo modification of the Kuhn-Munkres algorithm. The proposed method is relatively simple and finds an approximate solution to the matching. In the experiments we documented the algorithm

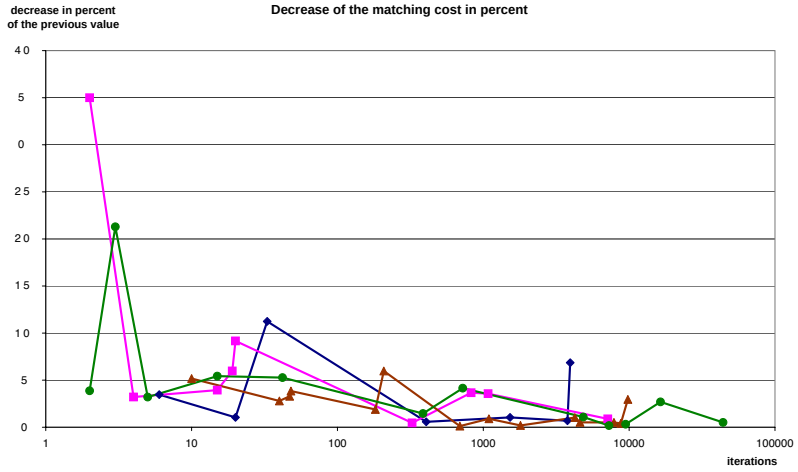


FIG. 5.4. *Decrease of the minimal cost expressed in percent of the previous minimum.*

running time and demonstrated the trend of the minimal cost of matching over successive iterations of the Monte Carlo method.

## REFERENCES

[1] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975.

[2] Mark de Berg, Marc van Kreveld, Mark Overmars, , and Otfried Schwarzkopf. *Computational Geometry*, chapter Voronoi Diagrams, pages 147–163. Springer-Verlag, 2nd edition, 2000.

[3] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry*, chapter Binary space partitions, pages 215–265. Springer-Verlag, 2nd edition, 2000.

[4] Jack Edmonds. Maximum matching and a polyhedron with $0,1$ vertices. *J. of Res. the Nat. Bureau of Standards*, 69B:125–130, 1965.

[5] Raphael A. Finkel and Louis Bentley. Quad trees: A data structure for retrieval on composite keys. *Acta Inf.*, 4:1–9, 1974.

[6] Henry Fuchs, Zvi M. Kedem, and Bruce F. Naylor. On visible surface generation by a priori tree structures. In *SIGGRAPH '80: Proceedings of the 7th annual conference on Computer graphics and interactive techniques*, pages 124–133, New York, NY, USA, 1980. ACM.

[7] Harold N. Gabow. An efficient implementation of Edmonds' algorithm for maximum matching on graphs. *J. ACM*, 23(2):221–234, 1976.

[8] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3):264–323, 1999.

[9] H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistic Quarterly*, 2:83–97, 1955.

[10] L. Lovász and M. D. Plummer. *Matching Theory*. Elsevier Science, North-Holland, Amsterdam, 1986.

[11] J. B. Macqueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.

[12] Nicholas Metropolis and S. Ulam. The monte carlo method. *Journal of the American Statistical Association*, 44(247):335–341, 1949.

[13] James Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, 1957.

[14] David B. Shmoys. Approximation algorithms for facility location problems. In *APPROX '00: Approximation Algorithms for Combinatorial Optimization*, volume 1913 of *Lecture Notes in Computer Science*, pages 27–33, London, UK, 2000. Springer-Verlag.