Theory of Computing Systems (2026) 70:25
https://doi.org/10.1007/s00224-025-10254-7

Fast Approximation Algorithms for Euclidean Minimum Weight Perfect Matching

Stefan Hougardy[1] · **Karolina Tammemaa**[2]

Received: 31 March 2025 / Accepted: 6 December 2025

Abstract
We study the Euclidean minimum weight perfect matching problem for n points in the plane. It is known that any deterministic approximation algorithm whose approximation ratio depends only on n requires at least $\Omega(n \log n)$ time. We propose such an algorithm for the Euclidean minimum weight perfect matching problem with runtime $O(n \log n)$ and show that it has approximation ratio $O(n^{0.206})$. This improves the so far best known approximation ratio of $n/2$. We also develop an $O(n \log n)$ algorithm for the Euclidean minimum weight perfect matching problem in higher dimensions and show it has approximation ratio $O(n^{0.412})$ in all fixed dimensions.

Keywords Euclidean matching · Approximation algorithms

1 Introduction

A *perfect matching* in a graph G is a subset of edges such that each vertex of G is incident to exactly one edge in the subset. When each edge e of G has a real weight w_e, then the *minimum weight perfect matching problem* is to find a perfect matching M that minimizes the weight $\sum_{e \in M} w_e$. If the vertices are points in the Euclidean plane and we have a complete graph where each edge e has weight w_e equal to the Euclidean distance between its two endpoints, we call this the *Euclidean minimum weight perfect matching problem.*

Stefan Hougardy funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC-2047/1 – 390685813. Karolina Tammemaa supported by Deutsche Forschungsgemeinschaft, DFG grant MN 59/4-1.

✉ Stefan Hougardy
hougardy@or.uni-bonn.de

Karolina Tammemaa
karolina.tammemaa@tuhh.de

1 Research Institute for Discrete Mathematics and Hausdorff Center for Mathematics, University of Bonn, Bonn, Germany

2 Hamburg University of Technology, Institute for Algorithms and Complexity, Hamburg, Germany

Published online: 08 April 2026

The Euclidean minimum weight perfect matching problem can be solved in polynomial time by applying Edmonds' blossom algorithm [3, 4] to the complete graph where the edge weights are the Euclidean distances between the edge endpoints. Gabow [5] and Lawler [9] have shown how Edmonds' algorithm can be implemented to achieve the runtime $O(n^3)$ on graphs with n vertices. This implies an $O(n^3)$ runtime for the Euclidean minimum weight perfect matching problem on point sets of size n. By exploiting the geometry of the problem Vaidya [17] developed an algorithm for the Euclidean minimum weight perfect matching problem with runtime $O(n^{\frac{5}{2}} \log n)$. In 1998, Varadarajan [19] improved on this result by presenting an $O(n^{\frac{3}{2}} \log^5(n))$ algorithm that uses geometric divide and conquer. This is the fastest exact algorithm currently known for the Euclidean minimum weight perfect matching problem.

Faster approximation algorithms for the Euclidean minimum weight perfect matching problem are known. In [16] Vaidya presented a $1 + \varepsilon$-approximation algorithm with runtime $O(n^{\frac{3}{2}} \log^{\frac{5}{2}} n(1/\varepsilon^3)\sqrt{\alpha(n,n)})$, where α is the inverse Ackermann function and $\varepsilon \leq 1$. Even faster approximation algorithms are known when randomization is allowed. Arora [1] presented a randomized $1 + \varepsilon$ approximation algorithm with runtime $O(n \log^{O(1/\varepsilon)} n)$ and failure probability $1/2$. Varadarajan and Agarwal [20] improved on this, presenting a randomized $1 + \varepsilon$ approximation algorithm with runtime $O(n/\varepsilon^3 \log^6 n)$. Rao and Smith [12] gave a constant factor $\frac{3}{2}e^{8\sqrt{2}} \approx 122\,905.81$ randomized approximation algorithm with runtime $O(n \log n)$. In the same paper, they also propose a deterministic $\frac{n}{2}$-approximation algorithm with runtime $O(n \log n)$.

Grigoriadis and Kalantari [6] have shown that any deterministic approximation algorithm for the Euclidean minimum weight perfect matching problem with approximation factor depending only on n needs to have a runtime of at least $\Omega(n \log n)$. It is therefore natural to ask: *What approximation ratio can be achieved in $O(n \log n)$ by a deterministic approximation algorithm for the Euclidean minimum weight perfect matching problem for n points in the plane?* In this paper we will significantly improve the approximation ratio of $n/2$ due to Rao and Smith [12] and show:

Theorem 1 *For n points in $\mathbb{R}^2$ there exists a deterministic $O(n^{0.206})$-approximation algorithm for the Euclidean minimum weight perfect matching problem with runtime $O(n \log n)$.*

Our algorithm is based on the idea of clustering the given points in the Euclidean plane into components of even cardinality. We show that there is a way to compute these components such that for all but one of the components we can find a perfect matching with small enough weight. We then apply our algorithm iteratively on this single remaining component. After sufficiently many iterations the number of points contained in the remaining component is small enough to apply an exact algorithm. Note that in the Euclidean minimum weight perfect matching problem we do not have the property that the value of an optimum solution is at least as large as the optimum value for any even cardinality subset of the given points. Crucial for the analysis of our approach is therefore that we can show that the size of the remaining component decreases faster than the value of an optimum solution for the remaining component increases. The algorithm of Rao and Smith [12] can be extended to higher (fixed) dimension resulting in an approximation ratio of n in runtime $O(n \log n)$.

Our approach also extends to higher (fixed) dimensions and we obtain the following result:

Theorem 2 *For any fixed dimension d there exists a deterministic* $O(n^{0.412})$*-approximation algorithm for the Euclidean minimum weight perfect matching problem in* $\mathbb{R}^d$ *with runtime* $O(n \log n)$.

Theorem 1 and Theorem 2 improve earlier results we obtained in a preliminary version of this paper [8]. There, we proved approximation factors of $O(n^{0.2995})$ and $O(n^{0.599})$ for the 2-dimensional and the d-dimensional case. Crucial to obtain the improved results presented here is to allow the NODE-REDUCTION-ALGORITHM to make many iterations and that we are able to analyze this algorithm.

The paper is organized as follows. In Section 2 we introduce the nearest neighbor graph and study its relation to minimum spanning trees and matchings. A basic ingredient to our main algorithm is a subroutine we call the EVEN-COMPONENT-ALGORITHM. We explain this subroutine and prove some basic facts about it in Section 3. This subroutine was also used in the *Even Forest Heuristic* of Rao and Smith [12] that deterministically achieves in $O(n \log n)$ the so far best approximation ratio of $n/2$. We will apply the EVEN-COMPONENT-ALGORITHM as a subroutine within our NODE-REDUCTION-ALGORITHM. The idea of the NODE-REDUCTION-ALGORITHM is to partition a point set into some subsets of even cardinality and a single remaining set of points such that two properties hold. First, the even subsets should allow to compute a short perfect matching within linear time. Second, the remaining set of points should be sufficiently small. In Section 4 we will present our NODE-REDUCTION-ALGORITHM and will derive bounds for its runtime and the size of the remaining subset. The next idea is to iterate the NODE-REDUCTION-ALGORITHM on the remaining subset until it becomes so small that we can apply an exact minimum weight perfect matching algorithm to it. We call the resulting algorithm the ITERATED-NODE-REDUCTION-ALGORITHM and analyze its runtime and approximation ratio in Section 5 and also prove there Theorem 1. In Section 6 we extend the ITERATED-NODE-REDUCTION-ALGORITHM to higher (fixed) dimension and prove Theorem 2. Finally, in Section 7 we provide a lower bound example for the ITERATED-NODE-REDUCTION-ALGORITHM.

2 Preliminaries

A crucial ingredient to our algorithm is the so-called *nearest neighbor graph* [10]. For a given point set in R^d we first fix an arbitrary total ordering on all points, which we use to break ties when creating the nearest neighbor graph to avoid getting cycles. We compute for each point all other points that have the smallest possible distance to this point. Among all these points we select, as its nearest neighbor, a point that is minimal with respect to the total ordering. Now we get the nearest neighbor graph by taking as vertices all points and adding an undirected edge between each point and its nearest neighbor without adding parallel edges. We will denote the nearest neighbor graph for a point set $V \subseteq \mathbb{R}^d$ by $NN(V)$. Immediately from the definition we get that the nearest neighbor graph is a forest. It is well known that the nearest neighbor graph for a point set in $\mathbb{R}^d$ is a subgraph of a Euclidean minimum spanning tree for this point set [10].

For $k \geq 2$ the nearest neighbor graph can be generalized to the k-nearest neighbor graph. To obtain this graph choose for each point its k nearest neighbors (ties broken arbitrarily) and connect them with an edge.

Shamos and Hoey [14] have shown that the nearest neighbor graph and a Euclidean minimum spanning tree for a point set of cardinality n in $\mathbb{R}^2$ can be computed in $O(n \log n)$. For the nearest neighbor graph this result also holds in higher (fixed) dimension as was shown by Vaidya [18]. The algorithm of Vaidya [18] even allows to compute the k-nearest neighbor graph for point sets in $\mathbb{R}^d$ in $O(n \log n)$ as long as k and d are fixed.

We call a connected component of a graph an *odd connected component* if it has odd cardinality. Analogously, we define an *even connected component*. We will denote by $\ell(e)$ the Euclidean length of an edge e and for a set E of edges we define $\ell(E) := \sum_{e \in E} \ell(e)$. For a point set V we denote a Euclidean minimum weight perfect matching for this point set by $MWPM(V)$. Clearly, the point set V must have even cardinality for a perfect matching to exist. Throughout this paper by log we mean the logarithm with base 2.

There is a simple connection between the length of the nearest neighbor graph for a point set V and a Euclidean minimum weight perfect matching for this point set:

Lemma 3 *For a point set $V \subseteq \mathbb{R}^d$ we have $\ell(NN(V)) \leq 2 \cdot \ell(MWPM(V))$.*

Proof In a Euclidean minimum weight perfect matching for a point set V each point $v \in V$ is incident to an edge that is at least as long as the distance to a nearest neighbor of v. Assign to each point $v \in V$ the length of the edge that it is incident to in a Euclidean minimum weight perfect matching of V. This way a total length of $2 \cdot \ell(MWPM(V))$ is assigned to the vertices. For each vertex the assigned edge length is at least as large as the distance to a nearest neighbor. Thus, by summing over all vertices we get $\ell(NN(V)) \leq 2 \cdot \ell(MWPM(V))$. □

3 The Even Component Algorithm

The currently best deterministic $O(n \log n)$ approximation algorithm for the Euclidean minimum weight perfect matching problem is the Even Forest Heuristic due to Rao and Smith [12]. It achieves an approximation ratio of $n/2$ and tight examples achieving this approximation ratio are known [12]. The Even Forest Heuristic first computes a minimum spanning tree of the graph. It then removes all so-called *even edges*, where an edge is even if its removal splits the tree into two connected components of even cardinality. After removing all such edges, the remaining graph is a forest whose connected components all have even cardinality. Within each connected component of the forest the Even Forest Heuristic then computes a Hamiltonian cycle by first doubling all edges of the tree and then short-cutting a Eulerian cycle. One obtains a matching from the Hamiltonian cycle by taking every second edge. We also make use of this second part of the Even Forest Heuristic and call it the EVEN-COMPONENT-ALGORITHM (see Algorithm 1). In line 5 of this algorithm we shortcut the edges of a Eulerian cycle. By this we mean that we iteratively replace for three consecutive vertices x, y, z the edges xy and yz by the edge xz if the vertex y has degree more than two. Notably, Rao and Smith applied this algorithm to a forest derived from a minimum spanning tree, we will apply it to a forest derived from the nearest neighbor graph.

Algorithm 1 EVEN-COMPONENT-ALGORITHM.

Input: a forest F where all connected components have even cardinality
Output: a perfect matching M
1: $M := \emptyset$
2: **for** each connected component of F **do**
3: double the edges in the component
4: compute a Eulerian cycle in the component
5: shortcut the edges in the Eulerian cycle to get a Hamiltonian cycle
6: add the shorter of the two perfect matchings inside the Hamiltonian cycle to M.
7: **end for**

For completeness we briefly restate the following two results and their proofs from [12]:

Lemma 4 ([12]) *The* EVEN-COMPONENT-ALGORITHM *applied to an even connected component returns a matching with length at most the total edge length of all edges in the even component.*

Proof By doubling all edges in a component we double the total edge length. The Eulerian cycle computed in line 4 has therefore exactly twice the length of the edges in the connected component. The triangle inequality implies that shortcutting this cycle in line 5 cannot make it longer. The Hamiltonian cycle therefore has length at most twice the length of all edges in the connected component. One of the two perfect matchings into which we can decompose the Hamiltonian cycle has length at most half of the length of the Hamiltonian cycle. Therefore, the length of the matching computed by the EVEN-COMPONENT-ALGORITHM is upper bounded by the length of all edges in the connected component. □

Lemma 5 ([12]) *The* EVEN-COMPONENT-ALGORITHM *has linear runtime.*

Proof We can use depth first search to compute in linear time the connected components and double the edges in all components. A Eulerian cycle can be computed in linear time using for example Hierholzer's algorithm [7]. Shortcutting can easily be done in linear time by running along the Eulerian cycle. We find the smaller of the two matchings by selecting every second edge of the Hamiltonian cycle. □

4 The Node Reduction Algorithm

A central part of our algorithm is a subroutine we call the NODE-REDUCTION-ALGORITHM. This algorithm gets as input a point set $V \subseteq \mathbb{R}^2$ of even cardinality and returns a subset $W \subseteq V$ and a perfect matching M for $V \setminus W$. The idea of this algorithm is to first compute the nearest neighbor graph $NN(V)$ of V. If $NN(V)$ has many odd connected components, then we will reduce this number by adding additional edges. More precisely, we will carry out up to r rounds of iteratively adding edges to $NN(V)$. In round $i \in \{1, \ldots, r\}$ we will add edges if the current number of odd connected components is larger than $|V|/x_i$. We will later choose appropriate values for r and $x_1, \ldots, x_r$. From each of the remaining odd connected components we remove one leaf vertex and put it into W. We are left with a set of even connected components and apply the EVEN-COMPONENT-ALGORITHM to each of these. Algorithm 2 shows the pseudo code of the NODE-REDUCTION-ALGORITHM. It uses as a subroutine the algorithm EDGES-FROM-TREE which is shown in Algorithm 3.

Algorithm 2 NODE-REDUCTION-ALGORITHM.

Input: a set $V \subseteq \mathbb{R}^2$ of even cardinality, $r \in \mathbb{N}$, $x_1 < x_2 < \ldots < x_r$ with $x_i \in \mathbb{R}$
Output: $W \subseteq V$ and a perfect matching M for $V \setminus W$

$G_0 := NN(V)$
compute a Euclidean minimum spanning tree T of V
$i := 0$
while $i < r$ and the number of odd connected components of G_i is $> |V|/x_{i+1}$ **do**
  $i = i + 1$
  $G_i := G_{i-1} \cup$ EDGES-FROM-TREE(G_{i-1}, T)
end while
$W := \emptyset$
for each odd connected component of G_i **do**
  choose one leaf node in the component and add it to W
end for
$M :=$ EVEN-COMPONENT-ALGORITHM$(G_i[V \setminus W])$

Algorithm 3 EDGES-FROM-TREE.

Input: a graph G on vertex set $V \subseteq \mathbb{R}^2$ and a tree T on V
Output: a subset of the edges of T

$S := \emptyset$
Choose a bijection $f : E(T) \to \{1, \ldots, |T| - 1\}$ s.t. $f(e_1) < f(e_2)$ implies $l(e_1) \leq l(e_2)$
for each odd connected component of G **do**
  add to S an edge from T with minimum f-value that leaves this component
end for
return the set S

We start by giving a bound on the length of the edges returned by the subroutine EDGES-FROM-TREE.

Lemma 6 *For a point set $V \subseteq \mathbb{R}^2$ let G be a graph on V and T be a Euclidean minimum spanning tree on V. The subroutine* EDGES-FROM-TREE(G, T) *returns a set of edges of length at most* $2 \cdot \ell(MWPM(V))$.

Proof Let S denote the edges returned from the subroutine EDGES-FROM-TREE(G, T). For each odd connected component of G there must exist an edge e in $MWPM(V)$ that leaves this component. By the cut property of minimum spanning trees the tree T contains an edge that leaves the component and has at most the length of e. As each edge in $MWPM(V)$ can connect at most two odd connected components we get the upper bound $\ell(S) \leq 2 \cdot \ell(MWPM(V))$. □

In the following we analyze the performance and runtime of the NODE-REDUCTION-ALGORITHM. One crucial part is to bound the size of the set W returned by the NODE-REDUCTION-ALGORITHM. For this it will be useful to assume a certain structure on the even and odd components in the graphs G_i that are computed within the NODE-REDUCTION-ALGORITHM. We make this more precise in the following definition.

Definition 1 Let $V \subseteq \mathbb{R}^2$ be a point set, T a minimum spanning tree for V, and $r \in \mathbb{N}$. Set $G_0 := NN(V)$ and recursively define $G_i := G_{i-1} \cup$ EDGES-FROM-TREE(G_{i-1}, T) for $i = 1, \ldots, r$. The set V is *r-well structured* if the following holds:

- all connected components of G_0 have size 2 or 3
- for $i = 1, \ldots, r$ each even connected component in G_i is either an even connected component in G_{i-1} or it consists of exactly two odd connected components of G_{i-1}
- for $i = 1, \ldots, r$ each odd connected component in G_i consists either of exactly one odd and one even connected component of G_{i-1} or it consists of exactly three odd connected components of G_{i-1}

Figure 1 shows an example of a 2-well structured point set of size 22. The graph G_0 contains six connected components of size 3 and two connected components of size 2. The graph G_1 is obtained from G_0 by adding the four red edges. G_1 contains two even connected components, one of size 2 and one of size 6. The connected component of size 2 was already a connected component in G_0. The connected component of size 6 in G_1 is obtained by joining two odd connected components of G_0. Moreover, G_1 contains two odd connected components, one of size 5 and one of size 9. The connected component of size 5 in G_1 consists of one odd and one even connected component from G_0; the connected component of size 9 in G_1 consists of three odd connected components of G_0. Finally, the graph G_2 is obtained from G_1 by adding the

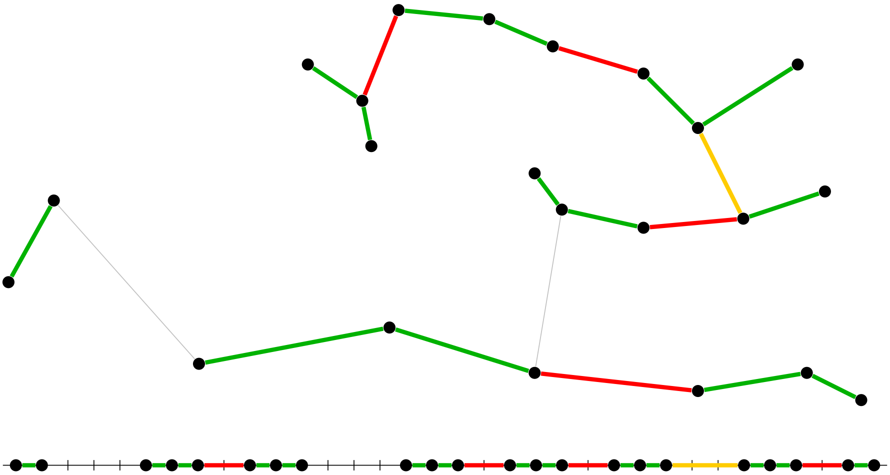

Fig. 1 (Top:) A 2-well structured point set. The edges of the nearest neighbor graph G_0 are shown in green. The graph G_1 contains in addition the red edges; the graph G_2 contains all colored edges. The two gray edges are edges of the minimum spanning tree. (Bottom:) The rearranged point set obtained after the first step of the construction described in the proof of Lemma 7

yellow edge between the two odd connected components of G_1. Note that all connected components of G_2 are even and therefore the point set is not only 2-well-structured but it is r-well structured for all $r \in \mathbb{N}$.

The following lemma shows that for each point set V there exists an r-well structured point set V' with the same size that results in the same number of odd connected components in each iteration of the NODE-REDUCTION-ALGORITHM. This result will allow us to prove in Corollary 8 that for analyzing the NODE-REDUCTION-ALGORITHM it is enough to consider r-well structured point sets.

Lemma 7 *Let $V \subseteq \mathbb{R}^2$ be a point set, $r \in \mathbb{N}$ with $r \geq 0$, and $2 < x_1 < x_2 < \ldots < x_r$ with $x_i \in \mathbb{R}$ such that the* NODE-REDUCTION-ALGORITHM *makes exactly r iterations. Let G_i be the graph computed by the* NODE-REDUCTION-ALGORITHM *in iteration i of the while-loop. Then there exists a well structured point set $V' \subseteq \mathbb{R}^2$ with $|V| = |V'|$ such that for each $i = 0, \ldots, r$ the graph G'_i computed by the* NODE-REDUCTION-ALGORITHM *on input V' has the same number of odd connected components as the graph G_i.*

Proof We will prove this result in two steps. First we show that one can rearrange the points in V so that they all lie on a horizontal line and the NODE-REDUCTION-ALGORITHM will still have the same set of vertices in each connected component of the graphs G_i for $i = 0, \ldots, r$. This is clear for the vertices of a single connected component of the graph G_0: if the cardinality of the component is k then for some number $a \in \mathbb{N}$ we can place the k vertices at coordinates $(a+1, 0)$, $(a+2, 0)$, $\ldots$, $(a+k, 0)$ and just have to make sure that all other vertices of V are placed left of $(a-1, 0)$ or right of $(a+k+2, 0)$. By defining an appropriate tie breaking function the nearest neighbor graph G_0 will contain exactly all edges of length 1 and therefore the k vertices still lie in a connected component of G_0. Using a recursive approach this idea can easily be extended to work for all connected components of all graphs G_i for $i = 0, \ldots, r$.

For a more formal description we show how to rearrange the points in V such that they all lie on a horizontal line and for $i = 0, \ldots, r$ each connected component of the graph G_i computed by the NODE-REDUCTION-ALGORITHM on the rearranged point set contains the same vertices as the connected components of the graph G_i for the original point set V. For this we use the algorithm PLACE-POINTS (Algorithm 4). It gets as input an integer a, a point set S and an index i. Starting at x-coordinate a the algorithm PLACE-POINTS recursively places all points in S on a horizontal line according to the connected components in the graph G_j for $j \leq i$. All points in S that belong to the same connected component of G_i are placed recursively by PLACE-POINTS such that each point has distance 1 to some point from the same connected component but distance of at least $i+2$ to points from a different connected component of G_i. At the lowest level of the recursion the points in a connected component of G_0 are placed one after the other on a horizontal line such that each point has distance one to the previous one. The first point of the next connected component of G_0 is then placed with larger distance.

We get all the coordinates of the rearranged points by the call PLACE-POINTS$(0, V, r)$. As a tie breaking function for the nearest neighbor graph, we use an ordering of the vertices on the line from left to right. A minimum spanning tree on the rearranged point

Algorithm 4 PLACE-POINTS.

```
Input: an integer a, a set of points S, and an index i
Output: x-coordinates for all points in S
for each connected component C of G_i within S do
  if i = 0 then
    place the points in C at x-values a, a + 1, ... a + |S| − 1
  else
    call PLACE-POINTS(a, V(C), i − 1)
  end if
  a := i + 2 + the largest x-value of a point in V(C)
end for
```

set is simply obtained by taking all edges between neighboring points on the line. As a tie breaking function f for edges of equal length, needed in line 2 of the algorithm EDGES-FROM-TREE, we use the ordering of the edges of the minimum spanning tree from left to right. Then, the NODE-REDUCTION-ALGORITHM applied to the rearranged point set will have exactly the same vertices in each connected component of G_i as for the original point set. Figure 1 shows a possible rearrangement obtained by the above procedure for the set of 22 points shown in the figure.

Let us call V'' the set of rearranged vertices we obtained from V by the above procedure. In a second step we modify the point set V'' to obtain the r-well structured point set V' by relocating some of the points. We start by setting $V' := V''$ and use the following recursive procedure for $i = r, \ldots, 0$ to modify the location of some points in V' (see Fig. 2 for an example). To avoid a special treatment for the case $i = 0$ we assume that G_{-1} is the graph where each vertex is an odd component. If a connected component C of G_i contains more than three odd connected components of G_{i-1} then we choose the two left most odd connected components C_1 and C_2 of G_{i-1} that lie in C and translate the points in C_1 and C_2 to the right of all points in V' such that the points in C_1 have distance $r + 2$ to the right most point in V' and the points in C_2 are right of all the points in C_1 and have distance $i + 1$ to the right most point in C_1 (see Fig. 2 (b)). Then we compact the space that the points in C_1 and C_2 had occupied before on the line. If the left most point of C_1 was at position a and the right neighbor of the right most point in C_1 is at position b then we shift all points that lie right of a by $b - a$ to the left. We do the same for C_2 (see Fig. 2 (c)). We apply this procedure as long as possible to G_i. We apply as long as possible a similar procedure to all odd connected components of G_i that contain an even connected component of G_{i-1} and

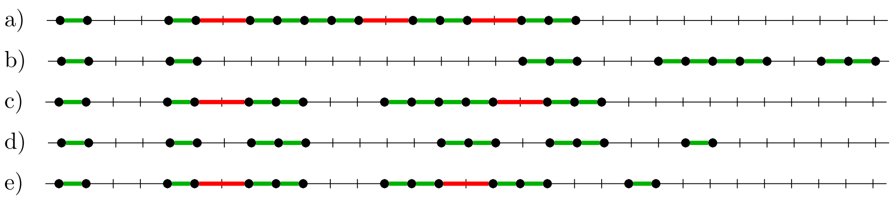

Fig. 2 Relocation of a point set to create a 1-well structured set. **a**) shows the start state with the graph G_1 containing the red and green edges while G_0 contains the green edges only. In **b**) two odd components of G_0 have been relocated to the right. In **c**) compaction has taken place. In **d**) two points from an odd connected component of size 5 in G_0 have been relocated to the right. **e**) shows the final state after compaction

more than one odd connected component of G_{i-1}. We apply a similar procedure to all even components of G_i that contain more than 2 odd connected component of G_{i-1} and no even connected component of G_{i-1}. We also apply a similar procedure to all even connected components of G_i that contain at least 2 odd connected component of G_{i-1} and one even connected component of G_{i-1}. Figure 2 shows how this method would rearrange a given point set of size 15.

We start applying these procedures to all connected components of G_r and then continue with the connected components of $G_{r-1}, G_{r-2}, \ldots, G_0$. When we apply these procedures to all connected components of G_i we do not change the relative positions of points in the same connected component of G_{i-1}. Therefore, for each $i = 0, \ldots, r$ the NODE-REDUCTION-ALGORITHM applied to V' will have exactly the same number of odd connected components in G'_i as the graph G_i. □

Corollary 8 *For proving a bound on the size of the set W returned after r iterations by the* NODE-REDUCTION-ALGORITHM *it is enough to prove a bound for r-well structured point sets.*

Proof The size of the set W returned by the NODE-REDUCTION-ALGORITHM after r iterations is equal to the number of odd connected components in the graph G_r. By Lemma 7 there exists an r-well structured point set V' with $|V'| = |V|$ such that the set W' returned by the NODE-REDUCTION-ALGORITHM on input set V' has the same size as the set W. Thus, a bound on the size of W' also yields the same bound on the size of W. □

The next lemma gives a bound on the length of the matching M and the size of the set W returned by the NODE-REDUCTION-ALGORITHM depending on the number of calls of the subroutine EDGES-FROM-TREE.

Lemma 9 *For a point set $V \subseteq \mathbb{R}^2$, $r \in \mathbb{N}$ with $r \geq 0$, $1 < x_1 < x_2 < \ldots < x_r$ with $x_i \in \mathbb{R}$, let W be the point set and M be the matching returned by the* NODE-REDUCTION-ALGORITHM. *If the subroutine* EDGES-FROM-TREE *is called q times within the* NODE-REDUCTION-ALGORITHM *then we have:*

(a) $|W| \leq \begin{cases} |V|/x_{q+1} & \text{if } q < r \\ |V|/\frac{3}{1-2\sum_{i=1}^{r}\frac{1}{x_i}} & \text{if } q = r \end{cases}$

(b) $\ell(M) \leq (2q+2) \cdot \ell(MWPM(V))$

(c) $\ell(MWPM(W)) \leq (2q+3) \cdot \ell(MWPM(V))$

Proof We first prove statement (a) for $q < r$. In this case the while-loop in the NODE-REDUCTION-ALGORITHM is terminated because G_q has at most $|V|/x_{q+1}$ odd connected components. By lines 8–11 the set W contains exactly one node from each odd connected component of G_q. We therefore get $|W| \leq |V|/x_{q+1}$.

We now prove statement (a) for the case $q = r$. By Corollary 8 we may assume that the point set V is r-well structured. The graph G_i contains the graph G_{i-1} as a subgraph. Moreover, the graph G_0 is $NN(V)$. Therefore, for all $0 \leq i \leq r$ each even connected component in G_i contains at least two vertices from V and each odd connected component in G_i contains at least three vertices from V.

In the following we denote by e_i the number of even connected components in G_i. Similarly, o_i denotes the number of odd connected components in G_i.

Claim 1 For $1 \leq i \leq r$ we have: $2o_{i-1} + 3o_i + 2e_i \leq |V|$.

We prove this claim by showing that within V we can choose three pairwise disjoint sets of size $2o_{i-1}$, $3o_i$, and $2e_i$. Within each odd connected component of G_{i-1} we choose two vertices. This results in a set of size $2o_{i-1}$. If an even connected component in G_i contains an even connected component from G_{i-1} we can choose two vertices from this even connected component of G_{i-1}. Otherwise, the even connected component in G_i must contain at least two odd connected components from G_{i-1} and we can choose from two such odd connected components a vertex that is distinct to the two already chosen vertices. This gives us in total a set of $2e_i$ vertices. If an odd connected component in G_i contains at least three odd connected components from G_{i-1}, then we can choose from three of these odd connected components a vertex that is different to the two already chosen vertices. Otherwise an odd connected component in G_i contains exactly one odd connected component from G_{i-1} and one even connected component from G_{i-1}. We can now choose one vertex from the odd connected component and two vertices from the even connected component. Thus, for each odd connected component in G_i we can choose three vertices that are different from vertices we have chosen before. Therefore we get $3o_i$ additional vertices. This proves the claim.

For $i = 1, \ldots, r$ we partition the odd and even connected components of G_i each into two sets as follows:

- $e_{i,0}$ denotes the number of even connected components in G_i containing no odd connected components of G_{i-1}.
- $e_{i,2}$ denotes the number of even connected components in G_i containing exactly two odd connected components of G_{i-1}.
- $o_{i,1}$ denotes the number of odd connected components in G_i containing exactly one even and one odd connected component of G_{i-1}.
- $o_{i,3}$ denotes the number of odd connected components in G_i containing exactly three odd connected components of G_{i-1}.

Immediately from this definition and because we have an r-well structured point set we get $e_i = e_{i,0} + e_{i,2}$ and $o_i = o_{i,1} + o_{i,3}$. Moreover, we have

$$o_{i,1} + 2e_{i,2} + 3o_{i,3} = o_{i-1} \quad \text{and} \quad e_{i,0} + o_{i,1} = e_{i-1}.$$

Using these two equations we get

$$e_i = e_{i,0} + e_{i,2} = e_{i-1} - o_{i,1} + o_{i-1} - o_{i,1} - e_{i,2} - 3o_{i,3} = e_{i-1} + o_{i-1} - 2o_i - e_{i,2} - o_{i,3} \tag{1}$$

Claim 2 For $1 \leq i \leq r$ we have $o_{i-1} - o_i = 2(e_{i,2} + o_{i,3})$

We know that $o_{i,1} + 2e_{i,2} + 3o_{i,3} = o_{i-1}$ and $o_i = o_{i,1} + o_{i,3}$ giving us $o_{i-1} - o_i = 2(e_{i,2} + o_{i,3})$. This proves the claim.

Claim 3 For $1 \leq i \leq r$ we have $o_i + 2e_i = -2o_i + (o_{i-1} + 2e_{i-1})$

By equation (1) and Claim 2 we have

$$\begin{aligned} o_i + 2e_i &= o_i + 2(e_{i-1} + o_{i-1} - 2o_i - e_{i,2} - o_{i,3}) \\ &= -3o_i + 2e_{i-1} + 2o_{i-1} - 2(e_{i,2} + o_{i,3}) \\ &= -3o_i + 2e_{i-1} + 2o_{i-1} - (o_{i-1} - o_i) \\ &= -2o_i + (o_{i-1} + 2e_{i-1}) \end{aligned}$$

Claim 4 For $r \in \mathbb{N}$ we have $3o_r \leq |V| - 2\sum_{t=0}^{r-1} o_t$.

For $r = 1$ we get from Claim 1 that:

$$3o_1 \leq |V| - 2o_0 - 2e_1 \leq |V| - 2o_0$$

For $r \geq 2$ we get by recursively applying Claim 3 and Claim 1 that:

$$\begin{aligned} 3o_r &\leq o_{r-1} + 2e_{r-1} \\ &= -2o_{r-1} + (o_{r-2} + 2e_{r-2}) \\ &= -2\sum_{t=1}^{r-2} o_{r-t} + o_1 + 2e_1 \\ &\leq -2\sum_{t=1}^{r-2} o_{r-t} + o_1 + |V| - 2o_0 - 3o_1 \\ &= |V| - 2\sum_{t=0}^{r-1} o_t \end{aligned}$$

This proves the claim.

As the algorithm did not terminate in earlier iterations we have for $1 \leq i \leq r$ $o_{i-1} > |V|/x_i$. Thus, using Claim 4 we get:

$$\begin{aligned} o_r &\leq \frac{1}{3}(|V| - 2\sum_{i=0}^{r-1} o_i) \\ &\leq \frac{1}{3}(|V| - 2|V|\sum_{i=0}^{r-1} 1/x_{i+1}) \\ &= |V|\frac{1 - 2\sum_{i=1}^{r} 1/x_i}{3} \\ &= |V|/\frac{3}{1 - 2\sum_{i=1}^{r} 1/x_i} \end{aligned}$$

This finishes the proof of statement (a).

We now prove statement (b). Let S be the set of all edges added to the graph G in the q iterations of the while-loop. From Lemma 6 we know that the total length $\ell(S)$ of all edges in S is at most $2q \cdot \ell(MWPM(V))$. In line 12 of the NODE-REDUCTION-ALGORITHM, the matching M is computed by the EVEN-COMPONENT-ALGORITHM on a subset of the edges in $NN(V) \cup S$. Lemma 4 therefore implies $\ell(M) \leq \ell(NN(V)) + \ell(S)$. Now Lemma 3 together with the above bound on $l(S)$ yields

$$\begin{aligned}\ell(M) &\leq \ell(NN(V)) + \ell(S)\\ &\leq 2 \cdot \ell(MWPM(V)) + 2q \cdot \ell(MWPM(V)) \;=\; 2(q+1) \cdot \ell(MWPM(V)).\end{aligned}$$

Finally, we prove statement (c). Let H be the graph obtained from $NN(V)$ by adding the edges of S and $MWPM(V)$. All connected components in H have even cardinality as by definition H contains a perfect matching. This implies that each connected component of H contains an even number of odd connected components from $NN(V) \cup S$. As W contains exactly one point from each odd connected component of $NN(V) \cup S$, this implies that each connected component of H contains an even number of vertices of W. Within each connected component of H we can therefore pair all vertices from W and connect each pair by a path. By taking the symmetric difference of all these paths we get a set of edge disjoint paths in H such that each vertex of W is an endpoint of exactly one such path. The triangle inequality implies that an edge connecting the two endpoints of such a path is at most as long as the path. If we replace each path by the edge connecting its two endpoints we get a perfect matching for W. Therefore, we know that the total length of all these edge disjoint paths in H is an upper bound for $\ell(MWPM(W))$. As the total length of all these edge disjoint paths is bounded by the total length of all edges in H, we get by using Lemma 3:

$$\begin{aligned}\ell(MWPM(W)) &\leq \ell(E(H))\\ &\leq \ell(NN(V)) + \ell(S) + \ell(MWPM(V))\\ &\leq (2q+3) \cdot \ell(MWPM(V)).\end{aligned}$$

(Part of this argument would easily follow from the theory of T-joins which we avoid to introduce here.) □

Lemma 10 *For constant r the* NODE-REDUCTION-ALGORITHM *(Algorithm 2) on input V with $|V| = n$ has runtime $O(n \log n)$.*

Proof In line 1 of the NODE-REDUCTION-ALGORITHM the nearest neighbor graph can be computed in $O(n \log n)$ as was proved by Shamos and Hoey [14]. As the nearest neighbor graph has a linear number of edges we can use depth first search to compute its connected components and their parity in $O(n)$. The minimum spanning tree T for V in line 2 of the algorithm can be computed in $O(n \log n)$ by using the algorithm of Shamos and Hoey [14]. To compute the set S in the subroutine EDGES-FROM-TREE, simply run through all edges of T and store for each odd connected component the shortest edge leaving that component. We have r calls to the subroutine EDGES-FROM-TREE. As T has $n - 1$ edges we get a total bound of $O(n)$ for all calls to the subroutine Edges-From-Tree. Choosing a leaf node in a connected component which is a tree can be done in time proportional to the size of the connected component; thus the total runtime of lines 8– 11 is $O(n)$. Finally, by Lemma 5 the runtime of the Even-Component-Algorithm is linear in the size of the input graph. Therefore, line 12 requires $O(n)$ runtime. Summing up all these time complexities gives us a time complexity $O(n \log n)$. □

5 Iterating the Node-Reduction-Algorithm

The Node-Reduction-Algorithm (Algorithm 2) on input $V \subseteq \mathbb{R}^2$ returns a set $W \subseteq V$ and a perfect matching on $V \setminus W$. The idea now is to iterate the Node-Reduction-Algorithm on the set W of unmatched vertices. By Lemma 9(a) we know that after each iteration the set W shrinks by at least a constant factor. Therefore, after a logarithmic number of iterations the set W will be empty. However, we can do a bit better by stopping as soon as the set W is small enough to compute a Euclidean minimum weight perfect matching on W in $O(n \log n)$ time. We call the resulting algorithm the Iterated-Node-Reduction-Algorithm, see Algorithm 5. In line 3 of this algorithm we apply the Node-Reduction-Algorithm to the point set V_j. This gives us a matching we denote by M_j and a set of unmatched points which we denote by V_{j+1}.

Algorithm 5 Iterated-Node-Reduction-Algorithm.

Input: a set $V \subseteq \mathbb{R}^2$ of even cardinality, $\varepsilon > 0$, $r \in \mathbb{N}$, $x_1 < x_2 < \ldots < x_r$ with $x_i \in \mathbb{R}$
Output: a perfect matching M for V
1: $V_1 := V$, $j := 1$
2: **while** $|V_j| > |V|^{2/3-\varepsilon}$ **do**
3: $\quad V_{j+1}, M_j \leftarrow$ Node-Reduction-Algorithm$(V_j, r, x_1, x_2, \ldots, x_r)$
4: $\quad j := j + 1$
5: **end while**
6: $M := MWPM(V_j) \cup M_1 \cup M_2 \cup \ldots \cup M_{j-1}$

Clearly, the Iterated-Node-Reduction-Algorithm returns a perfect matching on the input set V. The next lemma states the runtime of the Iterated-Node-Reduction-Algorithm.

Lemma 11 *The* Iterated-Node-Reduction-Algorithm *(Algorithm 5) on input V with $|V| = n$, fixed $r \in \mathbb{N}$ and $x_1 > 2$ has runtime $O(n \log n)$.*

Proof We have $|V_1| = |V|$ and because of $x_1 > 2$, Lemma 9(a) implies $|V_j| < |V|/2^{j-1}$ for $j \geq 2$. By Lemma 10 the runtime of the Node-Reduction-Algorithm on the set V_j is $O(|V_j| \log |V_j|)$. The total runtime for lines 2–5 of the Iterated-Node-Reduction-Algorithm is therefore bounded by

$$O\left(\sum_{j=1}^{\infty} |V_j| \log |V_j|\right) = O\left(\log |V| \cdot \sum_{j=1}^{\infty} \frac{|V|}{2^{j-1}}\right) = O(|V| \log |V|).$$

In line 6 of the ITERATED-NODE-REDUCTION-ALGORITHM we can use the algorithm of Varadarajan [19] to compute a Euclidean minimum weight perfect matching on V_j. For a point set of size s Varadarajan's algorithm has runtime $O(s^{\frac{3}{2}} \log^5(s))$. As the set V_j in line 6 of the ITERATED-NODE-REDUCTION-ALGORITHM has size at most $|V|^{2/3-\varepsilon}$ we get a runtime of

$$O\left((|V|^{2/3-\varepsilon})^{\frac{3}{2}} \log^5(|V|^{2/3-\varepsilon})\right) = O\left(\frac{|V|}{|V|^{\frac{3\epsilon}{2}}} \log^5(|V|^{2/3-\varepsilon})\right) = O(|V|).$$

In total the ITERATED-NODE-REDUCTION-ALGORITHM has runtime $O(n \log n)$. □

We now analyze the approximation ratio of the ITERATED-NODE-REDUCTION-ALGORITHM. For this we will have to choose appropriate values for $x_1, \ldots, x_r$. Our analysis will yield the best approximation ratio if we choose the x_i in such a way that

$$\frac{\log 3}{\log x_1} = \frac{\log 5}{\log x_2} = \cdots = \frac{\log(2r+3)}{\log x_{r+1}} \text{ with } x_{r+1} := \frac{3}{1 - 2\sum_{i=1}^{r} 1/x_i} \tag{2}$$

One can easily solve these equations numerically by a brute force search for all possible values of x_1 in the range between 4 and 6 and accuracy 10^{-9}. This way we get as a solution for $r = 3$:

$$x_1 \approx 4.34480819 \quad x_2 \approx 8.60221014 \quad x_3 \approx 13.48967391 \quad x_4 \approx 18.87735817$$

For $r = 1000$ we obtain:

$$x_1 \approx 5.92564165 \quad x_2 \approx 13.553044874 \quad \ldots \quad x_{1001} \approx 222506.653295 \tag{3}$$

To prove a bound on the approximation ratio of the ITERATED-NODE-REDUCTION-ALGORITHM, we will bound the length of the matching $MWPM(V_j)$ computed in line 6 of the algorithm and the total length of all matchings M_j computed in all iterations of the algorithm. We start with a bound on $\ell(MWPM(V_j))$.

Lemma 12 *For input $V \subseteq \mathbb{R}^2$, $\varepsilon > 0$, $r \in \mathbb{N}$ and $2 < x_1 < x_2 < \ldots < x_r$ that satisfy condition (2) the length of the matching $MWPM(V_j)$ computed in line 6 of the* ITERATED-NODE-REDUCTION-ALGORITHM *can be bounded by*

$$\ell(MWPM(V_j)) \leq (2r+3) \cdot \left(|V|^{1/3+\varepsilon}\right)^{\frac{\log 3}{\log x_1}} \cdot \ell(MWPM(V)).$$

Proof To avoid confusion we denote by j_0 the index j after finishing the while-loop and we use the variable j to denote one of the possible values of the index while running the while-loop. Thus we want to prove a bound on $\ell(MWPM(V_{j_0}))$.

As long as the set V_j has cardinality larger than $|V|^{2/3-\varepsilon}$ the ITERATED-NODE-REDUCTION-ALGORITHM makes in line 3 a call to the NODE-REDUCTION-ALGORITHM.

In each call the NODE-REDUCTION-ALGORITHM will execute between 0 and r calls to the subroutine EDGES-FROM-TREE. For $q = 0, \dots, r$ let a_q be the number of calls of the NODE-REDUCTION-ALGORITHM that make exactly q calls to the subroutine EDGES-FROM-TREE. Lemma 9(c) implies that if the NODE-REDUCTION-ALGORITHM executes q calls to the subroutine EDGES-FROM-TREE with input V_j, then

$$\ell(MWPM(V_{j+1})) \leq (2q+3) \cdot \ell(MWPM(V_j)). \tag{4}$$

Inequality (4) yields the following bound for the length of $MWPM(V_{j_0})$ computed in line 6 of the ITERATED-NODE-REDUCTION-ALGORITHM:

$$\ell(MWPM(V_{j_0})) \;\leq\; \prod_{q=0}^{r} (2q+3)^{a_q} \cdot \ell(MWPM(V)). \tag{5}$$

It remains to get a bound on $\prod_{q=0}^{r}(2q+3)^{a_q}$. If the NODE-REDUCTION-ALGORITHM makes $q < r$ calls to the subroutine EDGES-FROM-TREE then from Lemma 9(a) we know that

$$|V_{j+1}| \leq |V_j|/x_{q+1}. \tag{6}$$

If the NODE-REDUCTION-ALGORITHM makes r calls to the subroutine EDGES-FROM-TREE then from Lemma 9(a) and the definition of x_{r+1} in (2) we know that

$$|V_{j+1}| \leq |V_j|/x_{r+1}. \tag{7}$$

By the condition of the while-loop in lines 2–5 of the ITERATED-NODE-REDUCTION-ALGORITHM we get by using inequalities (6) and (7) that for the set V_{j_0} in line 6 of the ITERATED-NODE-REDUCTION-ALGORITHM we have

$$|V_{j_0}| \leq |V|^{2/3-\varepsilon} \quad \text{and} \quad |V_{j_0}| \leq \frac{|V|}{\prod_{q=0}^{r} \left(x_{q+1}\right)^{a_q}}$$

Similarly we get for the previous iteration $j_0 - 1$ as the while-loop has not yet ended and by inequalities (6) and (7) and the fact that x_{r+1} is the largest among all x_{q+1} for $q \in \{0, \dots, r\}$:

$$|V_{j_0-1}| > |V|^{2/3-\varepsilon} \quad \text{and} \quad |V_{j_0-1}| \leq \frac{|V| \cdot x_{r+1}}{\prod_{q=0}^{r} \left(x_{q+1}\right)^{a_q}}$$

which gives

$$|V|^{2/3-\varepsilon} < \frac{|V| \cdot x_{r+1}}{\prod_{q=0}^{r} \left(x_{q+1}\right)^{a_q}} \quad \text{or equivalently} \quad \prod_{q=0}^{r} \left(x_{q+1}\right)^{a_q} < |V|^{1/3+\varepsilon} \cdot x_{r+1}. \tag{8}$$

We now get by using (2) and (8):

$$\begin{aligned}\prod_{q=0}^{r}(2q+3)^{a_q} &= \prod_{q=0}^{r}\left(x_{q+1}\right)^{\frac{\log(2q+3)}{\log x_{q+1}}a_q}\\ &= \prod_{q=0}^{r}\left(\left(x_{q+1}\right)^{a_q}\right)^{\frac{\log 3}{\log x_1}}\\ &< \left(x_{r+1}\right)^{\frac{\log 3}{\log x_1}}\left(|V|^{1/3+\varepsilon}\right)^{\frac{\log 3}{\log x_1}}\\ &= \left(x_{r+1}\right)^{\frac{\log(2r+3))}{\log x_{r+1}}}\left(|V|^{1/3+\varepsilon}\right)^{\frac{\log 3}{\log x_1}}\\ &= (2r+3)\left(|V|^{1/3+\varepsilon}\right)^{\frac{\log 3}{\log x_1}} \qquad (9)\end{aligned}$$

By plugging this into inequality (5) we get:

$$\ell(MWPM(V_j)) \ < \ (2r+3)\cdot\left(|V|^{1/3+\varepsilon}\right)^{\frac{\log 3}{\log x_1}}\cdot \ell(MWPM(V)).$$

□

Lemma 13 *For input V and ε, let t denote the number of iterations made by the* ITERATED-NODE-REDUCTION-ALGORITHM. *Then we have*

$$\sum_{j=1}^{t}\ell(M_j) \ \leq\ 2\cdot z_t\cdot\prod_{j=1}^{t-1}y_j\cdot\ell(MWPM(V))$$

where M_j is the matching computed in line 3 of the algorithm and z_j and y_j are defined as follows: if the NODE-REDUCTION-ALGORITHM *makes q calls to* EDGES-FROM-TREE *in iteration j of the* ITERATED-NODE-REDUCTION-ALGORITHM *then $y_j = 2q+3$ and $z_j = 2q+2$.*

Proof By Lemma 9(b) we have $\ell(M_j) \leq z_j \cdot \ell(MWPM(V_j))$ for all $j = 1,\ldots,t$. Moreover, by Lemma 9(c) we have $\ell(MWPM(V_j)) \leq y_{j-1}\cdot\ell(MWPM(V_{j-1}))$ and therefore we get for all $j = 1,\ldots,t$:

$$\ell(M_j) \ \leq\ z_j\cdot\prod_{k=1}^{j-1}y_k\cdot\ell(MWPM(V)). \qquad (10)$$

We now prove the statement of the lemma by induction on t. For $t = 1$ we have by Lemma 9(b): $\ell(M_1) \leq z_1\cdot\ell(MWPM(V))$. Now let us assume $t > 1$ and that the statement holds for $t-1$. Using (10) we get

$$\sum_{j=1}^{t}\ell(M_j) \ =\ \ell(M_t)+\sum_{j=1}^{t-1}\ell(M_j)$$

$$\begin{aligned}
&\leq \ell(M_t) + 2 \cdot z_{t-1} \cdot \prod_{k=1}^{t-2} y_k \cdot \ell(MWPM(V)) \\
&\leq z_t \cdot \prod_{k=1}^{t-1} y_k \cdot \ell(MWPM(V)) + 2 \cdot z_{t-1} \cdot \prod_{k=1}^{t-2} y_k \cdot \ell(MWPM(V)) \\
&= (z_t \cdot y_{t-1} + 2 \cdot z_{t-1}) \cdot \prod_{k=1}^{t-2} y_k \cdot \ell(MWPM(V)).
\end{aligned}$$

Now we have $z_t \cdot y_{t-1} + 2 \cdot z_{t-1} \leq z_t \cdot y_{t-1} + 2 \cdot y_{t-1} \leq 2 \cdot z_t \cdot y_{t-1}$ which proves the lemma. □

Combining Lemmas 12 and 13, we can now state the approximation ratio for the ITERATED-NODE-REDUCTION-ALGORITHM.

Lemma 14 *For a set $V \subseteq \mathbb{R}^2$ of even cardinality, $\varepsilon > 0$, fixed $r \in \mathbb{N}$, and $2 < x_1 < x_2 < \ldots < x_r$ that satisfy condition (2) the* ITERATED-NODE-REDUCTION-ALGORITHM *has approximation ratio $O\left(|V|^{(1/3+\varepsilon)\cdot\frac{\log 3}{\log x_1}}\right)$.*

Proof Similar to the proof of Lemma 12, let a_q denote the number of iterations in the ITERATED-NODE-REDUCTION-ALGORITHM in which the NODE-REDUCTION-ALGORITHM makes exactly q calls to the subroutine EDGES-FROM-TREE. We denote the number of iterations of the ITERATED-NODE-REDUCTION-ALGORITHM by t and use the definition of z_j and y_j from Lemma 13. By Lemma 13 we have

$$\sum_{j=1}^{t} \ell(M_j) \leq 2 \cdot z_t \cdot \prod_{j=1}^{t-1} y_j \cdot \ell(MWPM(V)) \leq (4r+4) \cdot \prod_{j=0}^{r} (2q+3)^{a_q} \cdot \ell(MWPM(V)).$$

We already know from inequality (9) in the proof of Lemma 12 that this results in the upper bound $\sum_{j=1}^{t} \ell(M_j) = O\left(|V|^{(1/3+\varepsilon)\cdot\frac{\log 3}{\log x_1}} \cdot \ell(MWPM(V))\right)$. Together with Lemma 12 this implies that the length of the perfect matching returned by the ITERATED-NODE-REDUCTION-ALGORITHM is bounded by $O(|V|^{(1/3+\varepsilon)\cdot\frac{\log 3}{\log x_1}} \cdot \ell(MWPM(V)))$. □

We can now prove our main result on approximating Euclidean minimum weight perfect matchings for point sets in $\mathbb{R}^2$.

Theorem 1 *For n points in $\mathbb{R}^2$ there exists a deterministic $O(n^{0.206})$-approximation algorithm for the Euclidean minimum weight perfect matching problem with runtime $O(n \log n)$.*

Proof We claim that our algorithm ITERATED-NODE-REDUCTION-ALGORITHM has the desired properties if we set r to 1000 and choose $2 < x_1 < x_2 < \ldots < x_{1000}$ that satisfy condition (2). From Lemma 11 we know that the runtime of this algorithm is $O(n \log n)$. By Lemma 14 its approximation ratio is $O\left(|V|^{(1/3+\varepsilon)\cdot\frac{\log 3}{\log x_1}}\right)$. Using the numeric solution (3) for x_1 we get $\frac{1}{3}\frac{\log 3}{\log x_1} < 0.20582$. By choosing ε sufficiently small we get an approximation ratio of $O(n^{0.206})$. □

6 Extension to Higher Dimensions

We now want to extend our result for the 2-dimensional case to higher dimensions. For a fixed dimension $d > 2$ we can use essentially the same approach as in two dimensions, but need to adjust two things. First, for $d > 2$ no $O(n \log n)$ algorithm is known that computes a Euclidean minimum spanning tree in $\mathbb{R}^d$. Instead we will use Vaidya's [18] $O(n \log n)$ algorithm to compute a 3^r-nearest neighbor graph in $\mathbb{R}^d$ for fixed d and fixed r. We call this algorithm Edges-From-Nearest-Neighbor (Algorithm 6).

Algorithm 6 Edges-From-Nearest-Neighbor.

```
Input: a graph G on vertex set V, r ∈ ℕ and a 3^r-nearest neighbor graph NN' on V
Output: a subset of the edges of NN'
S := ∅
Choose a bijection f : E(NN') → {1, ..., |E(NN')|} s.t. f(e_1) < f(e_2) implies l(e_1) ≤ l(e_2)
for each odd connected component of G with size less than 3^r + 1 do
    add to S an edge from NN' with minimum f-value that leaves this component
end for
return the set S
```

We have to do the following changes to the Node-Reduction-Algorithm (Algorithm 2): In line 2, we calculate a 3^r-nearest-neighbor graph NN' of V instead of a minimum spanning tree T of V. In line 6, we call Edges-From-Nearest-Neighbor instead of Edges-From-Tree. For this modified algorithm, which we call the modified Node-Reduction-Algorithm, we get the following statement, which closely mimics Lemma 9.

Lemma 15 *For a point set $V \subseteq \mathbb{R}^d$, $r \in \mathbb{N}$, and $x_1, x_2, \ldots, x_r$ satisfying condition (2), let W be the point set and M be the matching returned by the modified* Node-Reduction-Algorithm. *If the subroutine* Edges-From-Nearest-Neighbor *is called q times within the modified* Node-Reduction-Algorithm *then we have:*

(a) $|W| \leq \begin{cases} |V|/x_{q+1} & \text{if } q < r \\ |V|/\frac{3}{1-2\sum_{i=1}^{r}\frac{1}{x_i}} & \text{if } q = r \end{cases}$

(b) $\ell(M) \leq (2q+2) \cdot \ell(MWPM(V))$

(c) $\ell(MWPM(W)) \leq (2q+3) \cdot \ell(MWPM(V))$

Proof We first prove statement (a). For $q < r$ the proof is the same as in Lemma 9. Case $q = r$. While in Section 4 we assumed the r-well structured sets to be in $\mathbb{R}^2$, the definition clearly extends to higher dimensions if instead of using Edges-From-Tree we use EDGES-FROM-NEAREST-NEIGHBOR. The method used in the proof of Lemma 7 for creating well structured sets does not depend on the dimension but only depends on which components get joined together at which round. All possible combinations can be translated to a line and thus we get a modified version of Lemma 7 with $V \subseteq \mathbb{R}^d$ and $V' \subseteq \mathbb{R}^d$. Thus Corollary 8 also holds in higher dimension and we may assume that the point set V is r-well structured.

This implies that each odd connected component in each graph G_i up to $i = r - 1$ contains at most 3^r points. Thus, the 3^r-nearest neighbor graph contains for each odd connected component at least one edge leaving it. On r-well structured sets the original and the modified NODE-REDUCTION-ALGORITHM therefore produce the same result on V. Then by using the Lemma 9 for the original NODE-REDUCTION-ALGORITHM we get that $|W| < |V|/\frac{3}{1-2\sum_{i=1}^{r}\frac{1}{x_i}}$.

Statements (b) and (c) follow from the fact that the proof of Lemma 6 also holds for the set S returned by EDGES-FROM-NEAREST-NEIGHBOR, meaning that $\ell(S) \leq 2\ell(MWPM(V))$. This is the only bound that is used for set S in Lemma 9 for proving the statements (b) and (c), thus the statement also holds for this modified version. □

We also have to change the ITERATED-NODE-REDUCTION-ALGORITHM (Algorithm 5). In line 2, instead of the threshold $V^{2/3-\varepsilon}$ we use the threshold $V^{1/3-\varepsilon}$. This is because the fastest known exact algorithm to compute a Euclidean minimum weight perfect matching in $\mathbb{R}^d$ is the $O(n^3)$ implementation of Edmonds' algorithm due to Gabow [5] and Lawler [9]. Using these two changes we get:

Theorem 2 *For any fixed dimension d there exists a deterministic $O(n^{0.412})$-approximation algorithm for the Euclidean minimum weight perfect matching problem in $\mathbb{R}^d$ with runtime $O(n \log n)$.*

Proof We get a higher dimensional analog of Lemma 12 by plugging Lemma 15 instead of Lemma 9 into the proof of Lemma 12. In addition we replace $|V|^{1/3+\varepsilon}$ by $|V|^{2/3+\varepsilon}$. If we apply all these changes then the approximation ratio in Lemma 14 changes to $O\left(|V|^{\frac{2}{3}\frac{\log 3}{\log x_1}}\right)$. Now setting $r = 1000$ and choosing the numeric value (3) for x_1 we get $\frac{2}{3}\frac{\log 3}{\log x_1} < 0.41163$. This implies the claimed approximation ratio $O(n^{0.412})$. The runtime analysis is the same as that from Lemma 11 due to Vaidya's $O(n \log n)$ algorithm [18] for computing *NN*'. □

Vaidya [18] states the runtime of his nearest-neighbor algorithm as $O((cd)^d n \log n)$, making the dependence on the dimension d explicit. Since computing *NN*' is the most time-consuming step in our approach, the overall runtime of our algorithm has exactly the same dependence on d as Vaidya's algorithm.

Remark Theorem 2 also holds if instead of the Euclidean metric, we have some other L_p, $p = 1, 2, \ldots, \infty$ metric. The runtime stays the same, as we can do all the computational steps, including the computationally expensive steps like finding the 3^r-nearest neighbor graph with the same runtime according to [18]. The approximation ratio also stays the same, as when finding different bounds, we only use the triangle inequality which also holds in L_p metric spaces.

Fig. 3 The recursively constructed point sets V_i shown for $i = 0, 1, 2$

7 A Lower Bound Example

In this section we provide a lower bound example for the ITERATED-NODE-REDUCTION-ALGORITHM, showing that its approximation ratio cannot be better than $O(n^{0.106})$.

Lemma 16 *The approximation ratio of the* ITERATED-NODE-REDUCTION-ALGORITHM *cannot be better than* $\Omega(n^{0.106})$.

Proof We recursively define point sets V_i for $i = 0, 1, 2, \ldots$ as follows. All points in V_i lie on a horizontal line. The set V_0 contains two points at distance 1. For $i \geq 1$ the point set V_i is obtained by placing seven copies of V_{i-1} next to each other on a horizontal line at distance 13^{i-1}. Figure 3 shows the point sets V_0, V_1, and V_2.

Immediately from the recursive definition of the sets V_i we get $|V_i| = 2 \cdot 7^i$. Each set V_i for $i \geq 1$ consists of 7^{i-1} groups of fourteen points placed at larger distance. As the distance between two neighboring points within each of these groups of fourteen points is exactly 1, we find a perfect matching of length 7 within each such group of fourteen points. Therefore we have $\ell(MWPM(V_i)) = 7^i = |V_i|/2$ for all $i \geq 0$.

If the ITERATED-NODE-REDUCTION-ALGORITHM is started on input set V_i for $i \geq 1$ then $NN(V_i)$ may contain exactly twelve edges within each group of fourteen consecutive points, as shown for V_1 in Fig. 4. Then the graph G_0 in the NODE-REDUCTION-ALGORITHM will contain $2 \cdot 7^{i-1} = |V_i|/7$ connected components each of size 7. If we choose $x_1 \approx 5.9256$ according to (3) then as G_0 has $|V_i|/7 < |V_i|/x_1$ many components, the routine EDGES-FROM-TREE is not called. Now the NODE-REDUCTION-ALGORITHM may choose within each group of fourteen consecutive points the two outermost points to include in the set W (see the example in Fig. 4). Thus, if the NODE-REDUCTION-ALGORITHM is started on the set V_i, then the set W after one iteration is a scaled version of the set V_{i-1} where the scaling factor is 13. Thus, the total length of a matching returned by the ITERATED-NODE-REDUCTION-ALGORITHM on the point set V_i is $\frac{2}{7}|V_i|$ plus the length of a matching returned by the ITERATED-NODE-REDUCTION-ALGORITHM on the point set V_{i-1} scaled by a factor of 13. The ITERATED-NODE-REDUCTION-ALGORITHM stops as soon as the remaining set of points has size less than $|V_i|^{2/3-\varepsilon}$. As in each iteration the size of the remaining point set decreases by a factor of 7, we will have at least k iterations where k satisfies $|V_i|/7^k = |V_i|^{2/3}$. This results in $k = \frac{1}{3}\log_7 |V_i|$. Therefore, we get as a lower

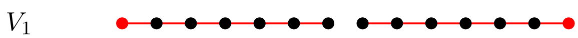

Fig. 4 A possible nearest neighbor graph for the point set V_1. The edges of the nearest neighbor graph are shown in red. The two points shown in red are the points that the NODE-REDUCTION-ALGORITHM may return as the set W

bound for the length of the matching returned by the ITERATED-NODE-REDUCTION-ALGORITHM on the point set V_i:

$$\frac{2}{7}\cdot|V_i|+\frac{2}{7}\cdot 13\cdot\frac{|V_i|}{7}+\ldots+\frac{2}{7}\cdot 13^{\lfloor k\rfloor}\cdot\frac{|V_i|}{7^{\lfloor k\rfloor}}=\frac{2}{7}\cdot|V_i|\cdot\sum_{j=0}^{\lfloor k\rfloor}\left(\frac{13}{7}\right)^j=\frac{1}{3}|V_i|\left(\left(\frac{13}{7}\right)^{\lfloor k\rfloor+1}-1\right).$$

Using $k=\frac{1}{3}\log_7|V_i|$ and assuming k to be sufficiently large we get

$$\frac{|V_i|}{3}\left(\left(\frac{13}{7}\right)^{\lfloor k\rfloor+1}-1\right)\geq\frac{|V_i|}{4}\left(\frac{13}{7}\right)^{\frac{1}{3}\log_7|V_i|}=\frac{|V_i|}{4}|V_i|^{\frac{1}{3}\frac{\log(13/7)}{\log 7}}>\frac{|V_i|^{1.106}}{4}$$

Thus we get an approximation ratio of $\Omega(n^{0.106})$. The total distance between the two outer most points in V_i is at most $|V_i|^{1.32}$. So the set V_i and its distances can be encoded with polynomial size. □

Note Added in Proof

The first deterministic $n/2$ approximation algorithm for the Euclidean minimum weight perfect matching problem with runtime $O(n\log n)$ was presented by Papadimitriou. It is described in [15]. The authors state that the runtime of the algorithm is $O(n^2)$ in the metric case. By applying the result of [14] one gets $O(n\log n)$ runtime in the Euclidean case. Reingold and Tarjan [13] showed that the greedy heuristic for the Euclidean minimum weight perfect matching problem has approximation ratio $n^{0.58496}$. They state a runtime of $O(n^2\log n)$. By applying the result of Bespamyatnikh [2] one gets $O(n\log n)$ runtime.

Author Contributions All authors contributed equally to this work.

Funding Open Access funding enabled and organized by Projekt DEAL.

Data Availability No datasets were generated or analysed during the current study.

Declarations

Competing Interests The authors declare no competing interests.

References

1. Arora, S.: Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. J. ACM **45**(5), 753–782 (1998). https://doi.org/10.1145/290179.290180
2. Bespamyatnikh, S.N.: An optimal algorithm for closest-pair maintenance. Discrete Comput. Geom. **6**, 175–195 (1998). https://doi.org/10.1007/PL00009340
3. Edmonds, J.: Maximum matching and a polyhedron with 0,1-vertices. J. Res. Natl. Bur. Stand. B **69B**(1), 125–130 (1965)
4. Edmonds, J.: Paths, trees, and flowers. Can. J. Math. **17**, 449–467 (1965). https://doi.org/10.4153/CJM-1965-045-4
5. Gabow, H.: An efficient implementation of Edmonds' maximum matching algorithm. Technical report no. 31, Digital Systems Laboratory, Stanford University, (1972)
6. Grigoriadis, M.D., Kalantari, B.: A lower bound to the complexity of Euclidean and rectilinear matching algorithms. Inf. Process. Lett. **22**, 73–76 (1986). https://doi.org/10.1016/0020-0190(86)90143-2
7. Hierholzer, C.: Ueber die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren. Math. Ann. **6**(1), 30–32 (1873). https://doi.org/10.1007/BF01442866
8. Hougardy, S., Tammemaa, K.: Fast approximation algorithms for Euclidean minimum weight perfect matching. In: Bieńkowski, M., Englert, M. (eds.) Approximation and Online Algorithms. WAOA 2024, volume 15269 of Lecture Notes in Computer Science, pp. 76–88 (2025). https://doi.org/10.1007/978-3-031-81396-2_6
9. Lawler, E.L.: Combinatorial Optimization: Networks and Matroids. Holt, Rinehart, and Winston, New York (1976)
10. Preparata, F.P., Shamos, M.I.: Computational geometry. An introduction. Texts and monographs in computer science. Springer Verlag, (1985). https://doi.org/10.1007/978-1-4612-1098-6
11. Rao, S.B., Smith, W.D.: Approximating geometrical graphs via "spanners" and "banyans". In: Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing, pp. 540–550, (1998). https://doi.org/10.1145/276698.276868
12. Rao, S.B., Smith, W.D.: Improved approximation schemes for geometrical graphs via "spanners" and "banyans". unpublished extended version of [10], (1998). https://rangevoting.org/WarrenSmithPages/homepage/rao-smith-tsp.ps
13. Reingold, E.M., Tarjan, R.E.: On a greedy heuristic for complete matching. SIAM J. Comput. **10**(4), 676–681 (1981). https://doi.org/10.1137/0210050
14. Shamos, M.I., Hoey, D.: Closest-point problems. In: 16th Annual Symposium on Foundations of Computer Science (FOCS 1975), pp. 151–162. IEEE, (1975). https://doi.org/10.1109/SFCS.1975.8
15. Supowit, K.J., Plaisted, D.A., Reingold, E.M.: Heuristics for weighted perfect matching. In: STOC '80: Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing, pp. 398–419 (1980). https://doi.org/10.1145/800141.804689
16. Vaidya, P.M.: Approximate minimum weight matching on points in k-dimensional space. Algorithmica **4**, 569–583 (1989). https://doi.org/10.1007/BF01553909
17. Vaidya, P.M.: Geometry helps in matching. SIAM J. Comput. **18**(6), 1201–1225 (1989). https://doi.org/10.1137/0218080
18. Vaidya, P.M.: An $O(n \log n)$ algorithm for the all-nearest-neighbors problem. Discrete Comput. Geom. **4**, 101–115 (1989). https://doi.org/10.1007/BF02187718
19. Varadarajan, K.R.: A divide-and-conquer algorithm for min-cost perfect matching in the plane. In: Proceedings 39th Annual Symposium on Foundations of Computer Science, pp. 320–329. IEEE, (1998). https://doi.org/10.1109/SFCS.1998.743466
20. Varadarajan, K.R., Agarwal, P.K.: Approximation algorithms for bipartite and non-bipartite matching in the plane. In: Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '99, pp. 805–814 (1999)