Combinatorial and Asymptotical Results on the Neighborhood Grid Data Structure

Martin Skrodzki[1], Ulrich Reitebuch[1], Konrad Polthier[1], and Shagnik Das[1]

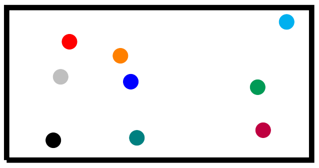
1 **Freie Universität Berlin**
`martin.skrodzki@fu-berlin.de`
`ulrich.reitebuch@fu-berlin.de`
`konrad.polthier@fu-berlin.de`
`shagnik@mi.fu-berlin.de`

Abstract

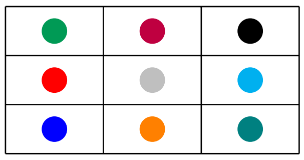
In 2009, Joselli et al. introduced the Neighborhood Grid data structure for fast computation of neighborhood estimates in point sets. Even though the data structure has been used in several applications and shown to be practically relevant, it is theoretically not yet well understood. The purpose of this paper is to give results on the complexity of building algorithms – both single-core and parallel – for the neighborhood grid. Furthermore, current investigations on related combinatorial questions are presented.

1 Introduction

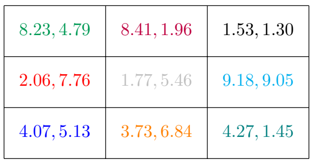
The neighborhood grid data structure can be used to compute estimates of neighborhoods in point sets. That is, for a given point p_i in a point set P, it provides a point p_j that is close to p_i but not necessarily its nearest neighbor in P. It has been introduced by Joselli et al. [3, 4]. In order to give a short introduction to the data structure, consider the example in Figure 1. It shows how points from a point set (Figure 1a) are placed in a grid (Figure 1b). The order in which the points are given is random, thus their initial placement in the grid is also. After the placement, only the coordinates of the points are considered in the grid (Figure 1c).



(a) Point set with nine points in $\mathbb{R}^2$.



(b) The points are initially placed randomly.



$8.23, 4.79$	$8.41, 1.96$	$1.53, 1.30$
$2.06, 7.76$	$1.77, 5.46$	$9.18, 9.05$
$4.07, 5.13$	$3.73, 6.84$	$4.27, 1.45$

(c) The point coordinates are considered.

Figure 1 First part of the neighborhood grid pipeline.

The grid as obtained in Figure 1c will now be sorted. Each row should grow in the first coordinates from left to right, each column should grow in the second coordinates from bottom to top. A corresponding sorted grid is given in Figure 2a. Note how it – in this example – recovers the combinatorial neighborhood relation from the points.

In order to use the grid to determine a neighborhood estimate for a given point p_i, find that point in the sorted grid. Then, consider a small neighborhood around that point in the grid, e.g. the one-ring around it. The size of this neighborhood should not depend on the number of inserted points such that this lookup runs in asymptotically constant time $\mathcal{O}(1)$. From that neighborhood in the grid, find the closest point to the considered point and output it as estimated nearest neighbor to p_i.

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

$2.06, 7.76$	$3.73, 6.84$	$9.18, 9.05$
$1.77, 5.46$	$4.07, 5.13$	$8.23, 4.79$
$1.53, 1.30$	$4.27, 1.45$	$8.41, 1.96$

(a) Coordinates are sorted to grow in their x-values in rows and in their y-values in columns.

$2.06, 7.76$	$3.73, 6.84$	$9.18, 9.05$
$1.77, 5.46$	$4.07, 5.13$	$8.23, 4.79$
$1.53, 1.30$	$4.27, 1.45$	$8.41, 1.96$

(b) Determining the neighbors of the upper left point by looking at its neighbors in the grid.

Figure 2 Second part of the neighborhood grid pipeline.

In this report, we present new results on the neighborhood grid data structure. A more extensive version of our results can be found in a corresponding paper on ArXiv [6]. Malheiros and Walter [2] investigated several iterative building strategies for the data structure. Despite the evidences of practical relevance, as given in the publication cited above, neither Joselli nor Malheiros investigated the asymptotic building times of the grid or answered the question for a time-optimal building algorithm. Therefore, this paper contains:

- a polynomial-time algorithm to build a neighborhood grid (Theorem 2.2),
- a proof of asymptotic time-optimality of the presented algorithm (Section 3.1),
- a comparison with the parallel building algorithm of Malheiros and Walter (Section 3.2).

The mentioned ArXiv paper contains – apart from more examples and proofs – several combinatorial results on the number of possible sorted placements, a complete list of unique sorted placements for $n \in \{1, 2, 3\}$, and a proof of non-existence of unique sorted placements for $n \geq 4$. So far, the following question remains unanswered:

- For a given $n \in \mathbb{N}$, $n \geq 4$, what is a point set with the least or largest number of stable states?

For the case of the largest number we give a conjecture.

2 The Neighborhood Grid

In this first section, we present the neighborhood data structure, fix corresponding notation, and prove a first theorem on a polynomial-time building algorithm.

2.1 Definition of the Data Structure

Given a set of points $P = \{p_1, \ldots, p_N \mid p_i \in \mathbb{R}^d\}$. In the following we will assume that $N = n^2$ for some $n \in \mathbb{N}$ and $d = 2$. Therefore, each point is given by $p_i = (p_{i1}, p_{i2}) \in \mathbb{R}^2$, where p_{i1} will be referred to as x- and p_{i2} as y-value. Furthermore, we assume that $p_i \neq p_j$ for all $i \neq j$. Finally, we can restrict w.l.o.g. to $\{p_{ik} \mid i \in [N]\} = [N]$ for $k \in \{1, 2\}$, which will be important in Section 3.1. Consider [6] for the general case without these restrictions.

The points will be placed in an $n \times n$ matrix, where each cell of the matrix contains a point p_i. That is, we consider a matrix $M \in (\mathbb{R}^2)^{n \times n}$ which then has the form

$$M = \begin{array}{|c|c|c|} \hline (a_{1n}, b_{1n}) & \ldots & (a_{nn}, b_{nn}) \\ \hline \vdots & \iddots & \vdots \\ \hline (a_{11}, b_{11}) & \ldots & (a_{n1}, b_{n1}) \\ \hline \end{array}. \tag{1}$$

Ultimately, we want to order the points in the matrix such that the following state is reached.

▶ **Definition 2.1.** The matrix M as given in (1) is said to be in a **stable state** if and only if the following two conditions are satisfied for any $i, j \in [n]$, $i \neq j$.

1. For all $k \in [n]$ it is: $i < j \Rightarrow a_{ki} < a_{kj} \vee (a_{ki} = a_{kj} \wedge b_{ki} < b_{kj})$.
2. For all $\ell \in [n]$ it is: $i < j \Rightarrow b_{i\ell} < b_{j\ell} \vee (b_{i\ell} = b_{j\ell} \wedge a_{i\ell} < a_{j\ell})$.

In other words, a matrix M is in a stable state, if the points in each row of M are ordered lexicographically according to the first and then the second coordinate. Similarly, all columns of M have to be ordered lexicographically according to the second and then the first coordinate. An illustration of Definition 2.1 is given in Figure 3. We call a stable state **unique**, if there exists no other stable state for the same point set P.

$(95, 13)$	$(95, 65)$	
	$(26, 61)$	$(13, 69)$
$(55, 42)$	$(60, 49)$	

$(06, 69)$	$(26, 61)$	$(86, 89)$
$(02, 55)$	$(80, 34)$	$(86, 41)$
$(05, 19)$	$(47, 11)$	$(95, 13)$

Figure 3 On the left a partially filled matrix with a violation of Definition 2.1 marked red. On the right a 3×3 matrix M in stable state.

2.2 Polynomial-Time building algorithm

Now the following question arises naturally: For any set of points P as specified above, is there a bijective placement $\pi : [n^2] \to [n] \times [n]$, $i \mapsto (k, \ell)$ such that the matrix $M_\pi(P)$ with

$$M = \begin{array}{|c|c|c|} \hline (p_{\pi^{-1}(n,1)1}, p_{\pi^{-1}(n,1)2}) & \dots & (p_{\pi^{-1}(n,n)1}, p_{\pi^{-1}(n,n)2}) \\ \hline \vdots & \iddots & \vdots \\ \hline (p_{\pi^{-1}(1,1)1}, p_{\pi^{-1}(1,1)2}) & \dots & (p_{\pi^{-1}(1,n)1}, p_{\pi^{-1}(1,n)2}) \\ \hline \end{array}. \tag{2}$$

is in a stable state? In other words, given n^2 points, can these be written into an $n \times n$ matrix such that it is in a stable state as defined in Definition 2.1.

▶ **Theorem 2.2.** *For every set of points $P = \{p_1, \dots, p_N \mid p_i \in \mathbb{R}^2\}$ there is a bijective placement π such that $M_\pi(P)$ is in a stable state. A placement π can be found in $\mathcal{O}(N \log(N))$.*

Proof. Consider the points $p_1, \dots, p_N$ as a sequence. Sort this sequence according to the first condition given in Definition 2.1. Obtain a sequence

$$(q_{11}, q_{12}), (q_{21}, q_{22}), \dots, (q_{N1}, q_{N2}),$$

where for $i, j \in [n]$, $i < j$ we have $q_{i1} < q_{j1}$ or $(q_{i1} = q_{j1} \wedge q_{i2} < q_{j2})$. Now split this sequence into n blocks as follows:

$$\underbrace{(q_{11}, q_{12}), \dots, (q_{n1}, q_{n2}),}_{=:Q_1} \underbrace{(q_{(n+1)1}, q_{(n+1)2}), \dots, (q_{(2n)1}, q_{(2n)2}),}_{=:Q_2}$$
$$\dots \underbrace{(q_{(n^2-n+1)1}, q_{(n^2-n+1)2}), \dots, (q_{N1}, q_{N2}),}_{=:Q_n}.$$

Now consider each sequence Q_i and sort it according to the second condition given in Definition 2.1. Obtain a sequence

$$R_k := (r_{11}, r_{12}), (r_{21}, r_{22}), \ldots, (r_{n1}, r_{n2}),\ k \in [n],$$

where for $i < j$ we have $r_{i2} < r_{j2}$ or $(r_{i2} = r_{j2} \wedge r_{i1} < r_{j1})$. That is, the points in the sequence R_k are sorted according to the second condition of Definition 2.1. Furthermore, for $i < j$, any point from R_i satisfies the first condition of Definition 2.1 when compared to any point from R_j, since the R_k derive from the Q_k. Therefore, placing the sequence R_k into the kth column of the matrix M results in a stable state.

Concerning the runtime, in the first step, N points were sorted, which takes $\mathcal{O}(N \log(N))$. In the second step, n sets of n points each were sorted, which takes

$$n \cdot \mathcal{O}(n \log(n)) = \mathcal{O}(n^2 \log(\sqrt{N})) = \mathcal{O}(N \log(N)),$$

as $N = n^2$. Hence, the stable state was computed in $\mathcal{O}(N \log(N))$. ◀

Theorem 2.2 imposes an upper bound on the runtime of any time-optimal comparison-based algorithm that creates a stable state of a matrix M. The next question is then: What is a lower bound?

3 Optimality of the Algorithm

3.1 A Lower Bound

To prove a lower bound, consider a comparison-based algorithm $\mathcal{A}$. The input to $\mathcal{A}$ is a point set P, the output is a stable placement π. Each query of $\mathcal{A}$ establishes $p_{ik} < p_{jk}$ for $i, j \in [N]$, $k \in \{1, 2\}$ and can be seen as a node of a decision-tree. The leafs of this tree correspond to placements of which some are stable for the given point set. A time-optimal algorithm builds this tree such that it is of depth $\log((n^2)!)$.

If we fix a placement π, we can say w.l.o.g. that π fixes the x-coordinates in the matrix such that they satisfy Definition 2.1. When counting the number of point sets for which π is stable, we can now pair the already placed x-values with y-values as follows: When setting up the y-values for the first column, one can pick n of the possible $N = n^2$ values, which then admit to a unique order in the column. Therefore, for the y-values in the first column, there are $\binom{n^2}{n}$ possibilities. For the second column, there are $\binom{n^2-n}{n}$ possibilities, until there is $\binom{n^2-(n-1)n}{n} = 1$ possibility for the last column. Overall, there are

$$\prod_{k=0}^{(n-1)} \binom{n^2 - kn}{n} = \frac{n^2!}{(n^2-n)!n!} \cdot \frac{(n^2-n)!}{(n^2-2n)!n!} \cdot \ldots \cdot \frac{(2n)!}{n!n!} \cdot \frac{n!}{n!} = \frac{(n^2)!}{(n!)^n}$$

possibilities to put y-values into the matrix and obtain a stable state from them utilizing the fixed π. That is, a placement π is always stable for exactly $\frac{(n^2)!}{(n!)^n}$ point sets.

Thus, when building its decision-tree, the algorithm $\mathcal{A}$ cannot stop at a subtree with more than $\frac{(n^2)!}{(n!)^n}$ leafs, as one of them will surely not be stable under the currently considered placement. That is, the tree has to be traversed to depth at least

$$\log((n^2)!) - \log\left(\frac{(n^2)!}{(n!)^n}\right) = \log\left(\frac{(n^2)! \cdot (n!)^n}{(n^2)!}\right) = \log((n!)^n) = n \cdot \log(n!) = \mathcal{O}(n^2 \cdot \log(n)).$$

Therefore, each decision-based algorithm building a stable state needs to perform at least $\Omega(n^2 \cdot \log(n))$ operations. Together with Theorem 2.2 this proves the following:

▶ **Theorem 3.1.** *The algorithm outlined in Theorem 2.2 is asymptotically time-optimal.*

3.2 Comparison to Malheiros and Walter

In the previous section, we have seen that a decision-based algorithm running on a single core has optimal asymptotic runtime $\mathcal{O}(n^2 \cdot \log(n))$. However, both Joselli et al. [3, 4] and Malheiros and Walter [2] utilize a parallelized version of odd-even sort. Assuming $n^2/2$ processors given, they alternately perform one step of the odd-even sort algorithm on rows and columns. By exchanging two points that violate Definition 2.1, they claim to converge to a stable state. Even though they do not prove this claim, it can easily be established when plugging the matrix M from Equation (1) into the energy

$$E(M) = \sum_{i,j=1}^{n} i \cdot a_{ij} + j \cdot b_{ij}, \tag{3}$$

which grows for each exchange, but is bounded from above. Thus, the procedure converges to a stable state.

As a point can only move by one row or column in each step, consider the element $(1, 1)$ that has to be placed in the lower left corner given our restrictions. In case it starts in the upper right corner, the algorithm needs to perform $2n - 2 = \mathcal{O}(n)$ steps to move the element to its designated position. Therefore, this parallel algorithm has a lower bound of $\omega(n)$. There are even examples for elements that cycle through the grid, consider [6] for an example.

Note that the algorithm presented in Theorem 2.2 needs to sort the given points. When utilizing $n^2/2$ processors, sorting can be performed in $\log(n^2)$ time, see [1]. Therefore, the presented algorithm can be parallelized to run in $\mathcal{O}(\log(n^2))$. However, this is of rather theoretical relevance, as the constants in [1] are comparably large.

Compare this to building a KdTree in parallel. In each step i, we have to sort i sets of $n^2/2^i$ points in the dimension with largest spread, which takes $\log(n^2) - i$ time for each of the $\log(n^2)$ levels of the tree, resulting in a total building time of $\mathcal{O}(\log^2(n))$. Therefore, the neighborhood grid can be build slightly faster, but only gives estimated answers, while the KdTree provides exact neighbor relations.

4 Combinatorial Questions

4.1 Point Set with a unique Stable State

In the previous section it was shown that the running time of the algorithm outlined in Theorem 2.2 is asymptotically time-optimal. However, the question remains whether the stable state found by the algorithm for a given point set P is unique. By iterating over all possible point sets P with $n = 4$, we found that none of the two-dimensional point sets on 16 points has a unique stable state. Utilizing an inductive argument, we show that given any point set P with $n \geq 5$, there exists no unique stable state. See [6] for details and a complete enumeration of unique stable states in the case of $n \in \{1, 2, 3\}$. The fact that for $n \geq 4$ there is no point set with a single unique stable state raises the following question:

Open Question. Given $n \in \mathbb{N}$, $n \geq 4$, what is a point set P with the minimum number of stable states among all point sets with n^2 points?

4.2 Point Set with largest number of Stable States

We proceed by turning the question from the last section around. What is the maximal number of stable states a point set can obtain for some given $n \in \mathbb{N}$? In order to investigate this question, we first turn to a specific point set, for which we can count the number of stable states. Consider the *identity*: $\{(1,1),(2,2),\dots,(n^2,n^2)\}$. Counting the number of stable states for the identity is equivalent to placing only one number in each field of the $n \times n$ matrix, which then has to satisfy both conditions of Definition 2.1. But this is exactly the number of standard Young tableaux of shape $(n,\dots,n)$. See [5] for an introduction into the underlying combinatorics and [6] for the application of these to the given setup. The number of stable states of the identity is then given by

$$f^{(n,\dots,n)} = \frac{N!}{\prod_{i=1}^{n}\prod_{j=1}^{n}(2n-i-j+1)}. \tag{4}$$

The results for $n \in \{1,2,3\}$ and computational experiments lead us to state the following conjecture.

▶ **Conjecture 4.1.** *Given $n \in \mathbb{N}$, the number of stable states of any point set P on n^2 points is less or equal to $f^{(n,\dots,n)}$.*

5 Conclusion and Future Work

We have presented a polynomial-time algorithm to build a stable state for a given point set P. Furthermore, we have proven the parallel algorithm from [2–4] to converge to a stable state and provided a lower bound on its runtime. Finally, we have deduced two open combinatorial questions resulting from the investigations of the data structure. A question not addressed in this paper concerns the quality of neighborhood estimates obtained from the grid. Answering these is left as future work.

Acknowledgments. The first author acknowledges the support by the Einstein Center for Mathematics Berlin, the Berlin Mathematical School, and the German National Academic Foundation. Furthermore, this research was supported by the DFG Collaborative Research Center TRR 109, 'Discretization in Geometry and Dynamics'.

References

1 Miklós Ajtai, János Komlós, and Endre Szemerédi. Sorting in c logn parallel steps. *Combinatorica*, 3(1):1–19, 1983.

2 Marcelo de Gomensoro Malheiros and Marcelo Walter. Simple and efficient approximate nearest neighbor search using spatial sorting. In *Graphics, Patterns and Images (SIBGRAPI), 2015 28th SIBGRAPI Conference on*, pages 180–187. IEEE, 2015.

3 Mark Joselli, José Ricardo da S Junior, Esteban W Clua, Anselmo Montenegro, Marcos Lage, and Paulo Pagliosa. Neighborhood grid: A novel data structure for fluids animation with gpu computing. *Journal of Parallel and Distributed Computing*, 75:20–28, 2015.

4 Mark Joselli, Erick Baptista Passos, Marcelo Zamith, Esteban Clua, Anselmo Montenegro, and Bruno Feijó. A neighborhood grid data structure for massive 3d crowd simulation on gpu. In *2009 VIII Brazilian Symposium on Games and Digital Entertainment (SBGAMES)*, pages 121–131. IEEE, 2009.

5 Bruce E. Saga. *The Symmetric Group*. Springer, 2001.

6 M. Skrodzki, U. Reitebuch, and K. Polthier. Combinatorial and Asymptotical Results on the Neighborhood Grid. *ArXiv e-prints*, October 2017. arXiv:1710.03435.