Dynamic Disk Connectivity*

Alexander Kauer[1] and Wolfgang Mulzer[1]

1 **Institut für Informatik, Freie Universität Berlin, Berlin, Germany**
[akauer, mulzer]@inf.fu-berlin.de

Abstract

Let $0 < \varphi \leq 1$ be a parameter. We present a data structure for maintaining the connected components of the intersection graph of a set of n disks with radii in $[\varphi, 1]$. The data structure allows inserting or deleting a disk in $\mathcal{O}(\frac{1}{\varphi} 2^{\alpha(n)} \log^{10} n)$ amortized expected time and querying two disks for connectivity in $\mathcal{O}(\log n)$ amortized time, where $\alpha(n)$ is the inverse Ackermann function. It requires $\mathcal{O}(\frac{1}{\varphi} n \log^3 n)$ expected space.

1 Introduction

Determining the connectivity in graphs is a fundamental algorithmic problem. The dynamic version where edges can be inserted or deleted is reasonably well understood [3–5, 9, 11], with data structures that support updates and queries for the connectivity of two vertices in polylogarithmic time. However, the case of vertex updates seems significantly harder, as a single update can have a larger impact. Chan et al. [2, Theorem 1] presented a data structure allowing vertex updates in $\widetilde{\mathcal{O}}(m^{2/3})$ amortized time and queries in $\widetilde{\mathcal{O}}(m^{1/3})$ time, where m is the number of possible edges of the graph that need to be known in advance.

A special case is dynamic connectivity of geometric intersection graphs. The vertices of such graphs are geometric objects of a certain type and two vertices share an edge if and only if the respective objects intersect. Connectivity queries now model reachability queries in sensor/IoT networks, road networks, and similar geometrically defined networks. Due to the restricted nature of the possible graphs, faster solutions may now be within reach. On the other hand, we must perform additional work to find the edges affected by an update.

Chan et al. [2, Theorem 5] also gave a general method for various geometric objects with slightly sub-linear update times and sub-linear query times. For disks with radii in $[\varphi, 1]$, $0 < \varphi \leq 1$, Kaplan et al. [7] described a faster data structure with $\mathcal{O}((\frac{1}{\varphi})^2 2^{\alpha(n)} \log^{10} n)$ amortized expected update time and $\mathcal{O}(\frac{\log n}{\log \log n})$ worst case query time; see Seiferth's thesis for details [10, Theorem 3.11]. Here, we show how to improve the dependence on $\frac{1}{\varphi}$ to linear.

2 Basic Composition of the Data Structure

The data structure by Kaplan et al. [7] relies on a dynamic graph connectivity structure for edge updates combined with appropriate rebuilding for changing vertex counts.

▶ **Theorem 2.1** (Holm et al. [5, Theorem 3]). *Let G be a graph with n vertices. There is a data structure such that inserting or deleting an edge in G take amortized time $\mathcal{O}(\log^2 n)$ and a connectivity query takes worst case time $\mathcal{O}(\frac{\log n}{\log \log n})$. This data structure requires $\mathcal{O}(m + n \log n)$ space, where m is the largest edge count at any given time.*

To avoid too many edge updates in the data structure of Holm et al. and to allow faster retrieval of intersecting disks, the intersection graph is not maintained explicitly, but is

* Partially supported by ERC STG 757609, GIF grant 1367/2015, and DFG grant MU 3501/2-2.

35th European Workshop on Computational Geometry, Utrecht, The Netherlands, March 18–20, 2019.
This is an extended abstract of a presentation given at EuroCG'19. 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.

represented by a grid with cell diameter φ. All disks with center in a single cell intersect, so we can merge them for connectivity. We use the cells as vertices and insert an edge between two of them if they contain intersecting disks. Every cell's disks can intersect only disks of $\mathcal{O}((\frac{1}{\varphi})^2)$ other cells, called the *neighborhood*. This limits the number of edge updates.

When inserting or deleting a disk, we must determine for which cells in the neighborhood we need to insert or delete edges. For this, we maintain for each pair of neighboring cells a *maximal bichromatic matching* (MBM) of intersecting disks between the cells using a data structure by Kaplan et al. requiring $\mathcal{O}(2^{\alpha(n)} \log^{10} n)$ expected amortized time for updates and $\mathcal{O}(n \log^3 n)$ expected space [7, Lemma 9.9 in the full version]. Altogether, this results in Kaplan et al.'s main result for dynamic disk connectivity:

▶ **Theorem 2.2** (Data Structure for Disk Graphs [10, Theorem 3.11]). *Let $0 < \varphi \leq 1$, and let P be a set of disks with radius in $[\varphi, 1]$. There is a dynamic connectivity structure for the intersection graph of P such that the insertion or deletion of a disk takes amortized expected time $\mathcal{O}((\frac{1}{\varphi})^2 2^{\alpha(n)} \log^{10} n)$ and a connectivity query takes worst case time $\mathcal{O}(\frac{\log n}{\log \log n})$, where n is the maximum number of inserted disks at any time and $\alpha(n)$ is the inverse Ackermann function. The data structure requires $\mathcal{O}((\frac{1}{\varphi})^2 n \log^3 n)$ expected space.*

A limitation of the approach behind Theorem 2.2 is that all disks are handled identically, irrespective of their actual size. To address this, we will use a hierarchy of grids $(\mathcal{G}_i)_{0 \leq i \leq \lceil \log \frac{1}{\varphi} \rceil}$ instead of a single grid, where $\mathcal{G}_i$ has cells of diameter $\frac{1}{2^i}$, for $i = 0, \ldots, \lceil \log \frac{1}{\varphi} \rceil$. Each grid with cells of diameter d then stores all disks with radius in $[d, 2d)$.

We represent the hierarchy of grids with quadtrees [1], where every cell $\mathcal{G}_0$ constitutes the root of one quadtree. Each disks s is stored in the quadtree cell that contains s's center and whose diameter is comparable to the radius of s, as explained above. The quadtrees have height at most $\lceil \log \frac{1}{\varphi} \rceil$ and their nodes are created upon the first access.

As before, two intersecting disks must be contained in two neighboring cells. Each neighboring cell σ of a given cell τ is either a child of a larger neighboring cell of τ, a child of τ itself, or a cell in $\mathcal{G}_0$. See Figure 1 for an example. A cell has of at most $13^2 \in \mathcal{O}(1)$ neighbors on its own level and on each level above, whereas in the levels below the number grows exponentially. Altogether, the size of a neighborhood is at most

$$\mathcal{O}(1) \cdot \sum_{i=0}^{\lceil \log \frac{1}{\varphi} \rceil} (2^i)^2 \in \mathcal{O}\Big(\big(\tfrac{1}{\varphi}\big)^2\Big). \tag{1}$$

▶ **Lemma 2.3.** *The data structure based on quadtrees has the same asymptotic update, query, and space bounds as the one from Theorem 2.2.*

Proof. When inserting or deleting a disk with radius r we need to obtain its cell $\sigma \in \mathcal{G}_i$ with $\frac{1}{2^i} \leq r < \frac{1}{2^{i-1}}$ and its neighborhood. We already observed that all neighboring cells are either a child of another neighboring cell, a child of σ, or in $\mathcal{G}_0$. Thus, these can be obtained via recursing down one or multiple quadtrees, requiring only constant time per neighbor. For each of the $\mathcal{O}((\frac{1}{\varphi})^2)$ neighboring cells we then need to update the MBM and may need to update the connectivity structure, as before. ◀

3 Different Handling of a Disk's Boundary and Inner Area

3.1 Limit the Insertions into MBMs

The quadratic dependency on $\frac{1}{\varphi}$ during updates in Lemma 2.3 can be approached using two observations: First, when a cell's disks are contained completely in an inserted disk, we do

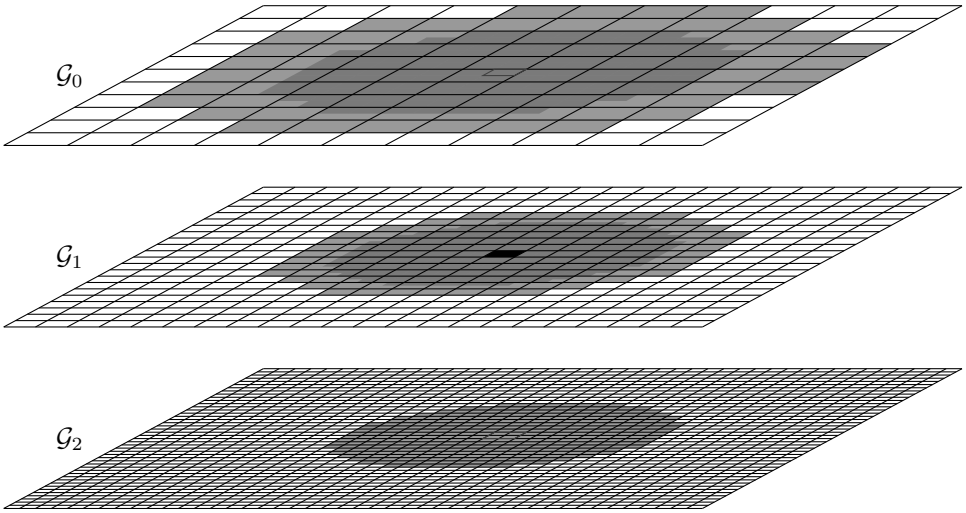

Figure 1 The neighborhood of the black colored cell in $\mathcal{G}_1$. The area of the neighboring cells in one level beneath is colored in a darker shade. When representing the grids via quadtrees every neighboring cell is either a child of another neighboring cell, a child of the black cell, or in $\mathcal{G}_0$.

not need to update the MBM. Instead, we can directly insert an edge into the underlying connectivity structure.

▶ **Definition 3.1.** Let s be a disk and $\sigma \in \mathcal{G}_i$ a cell, for some $i \geq 0$. Then, σ is *fully contained* in s if and only if σ intersects s and cannot contain disks intersecting the boundary of s.

Second, we bound the number of cells that still require an MBM after considering the fully contained cells.

▶ **Lemma 3.2.** *Inserting or deleting a disk s of radius r into the data structure of Lemma 2.3 requires checking $\mathcal{O}(\frac{r}{\varphi})$ cells that may contain disks intersecting the boundary of s. Those can be found in $\mathcal{O}(\frac{r}{\varphi})$ time.*

These are exactly all cells within some distance of the updated disk's boundary, where the distance depends on the disk's radius and the cell diameter. They must be either a cell of $\mathcal{G}_0$ or a child of a cell of that type, allowing the retrieval in $\mathcal{O}(\frac{r}{\varphi})$ time by a simple top-down traversal. See Figure 2. Another look at the cells during the recursive retrieval yields the following corollary.

▶ **Corollary 3.3.** *Inserting or deleting a disk s of radius r into the data structure of Lemma 2.3 requires checking $\mathcal{O}((\frac{r}{\varphi})^2)$ cells fully contained in s. Those can be found in $\mathcal{O}((\frac{r}{\varphi})^2)$ time.*

Among all paths in the quadtrees there are $\mathcal{O}(\frac{r}{\varphi})$ topmost cells fully contained in s. These can be found in $\mathcal{O}(\frac{r}{\varphi})$ time. Their interiors are pairwise disjoint and their union is exactly the union of all cells fully contained in s.

Using Corollary 3.3, we can save some time during updates: we do not update the MBM to inner cells, but insert an edge directly into the underlying edge connectivity structure.

▶ **Lemma 3.4.** *There is a data structure for dynamic disk connectivity with expected amortized update time $\mathcal{O}((\frac{1}{\varphi})^2 \log^2 n + \frac{1}{\varphi} 2^{\alpha(n)} \log^{10} n)$ and worst case time $\mathcal{O}(\frac{\log n}{\log \log n})$ for connectivity queries while requiring $\mathcal{O}((\frac{1}{\varphi}) n \log^3 n + (\frac{1}{\varphi})^2 n)$ expected space.*

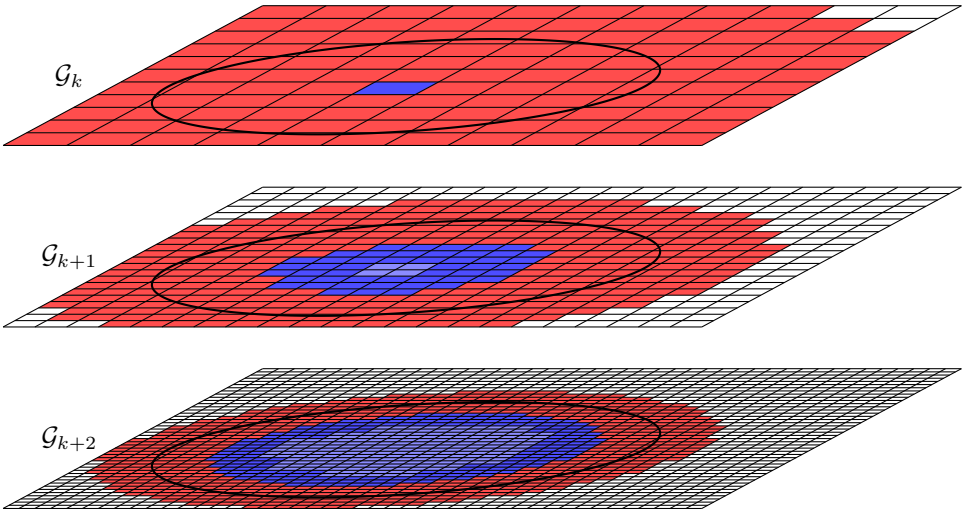

Figure 2 The types of cells which require checking when updating a disk in Lemma 2.3: fully contained (topmost) fully contained may contain disks intersecting the boundary

Proof. We augment the data structure of Lemma 2.3. In addition to an MBM, we save a counter for each neighboring pair of cells of different size. The counter describes how many of the disks of the larger cell fully contain the smaller cell.

When updating a pair during insertion or deletion of a disk s, we now update either the counter or the MBM. If s fully contains the other cell we update the counter, otherwise the MBM. Both cells' content intersect if and only if the counter is non-zero and the smaller cell is non-empty or the MBM contains an edge. Depending on whether this condition changes during the update, the edge connectivity data structure must be updated as well.

When updating a disk s, we encounter $\mathcal{O}(\log \frac{1}{\varphi})$ neighboring cells until we reach the level where s must be inserted or deleted. For each of these, an update to the MBM is required. Afterwards, we recurse down according to Corollary 3.3 to retrieve all $\mathcal{O}((\frac{1}{\varphi})^2)$ fully contained cells and the remaining $\mathcal{O}(\frac{1}{\varphi})$ cells described in Lemma 3.2 and update the MBMs, counters, and connectivity structure accordingly. ◀

3.2 Query for Replacements Instead

We were able to reduce the cost of the quadratic part, but didn't eliminate it. As we can have $\mathcal{O}((\frac{1}{\varphi})^2)$ edge changes during an update we cannot fully handle them in the update. Thus, we move parts of the handling of fully contained cells into the query.

▶ **Lemma 3.5.** *The result of a query into the data structure of Lemma 3.4 does not change when we omit all edges introduced solely through fully contained cells (i.e. non-zero counters), except those involving the cells containing at least one of the query disks.*

See Figure 3 (a). By Lemma 3.5, we can ignore fully contained cells during updates, except maintaining for each cell which disks fully contain them as topmost. Instead of inserting the required edges during a query, we query for suitable representatives.

Fix a query for two disks s_1, s_2. Consider two other disks t_1, t_2, which contain s_1 with the radius of t_1 not smaller than the radius of t_2. If there is a path in the intersection graph between s_1 and s_2, then there is a path from t_1 to s_2, even when s_1 is removed. Additionally, when s_2 is not contained in t_1, then the boundary of t_1 must intersect the boundary of

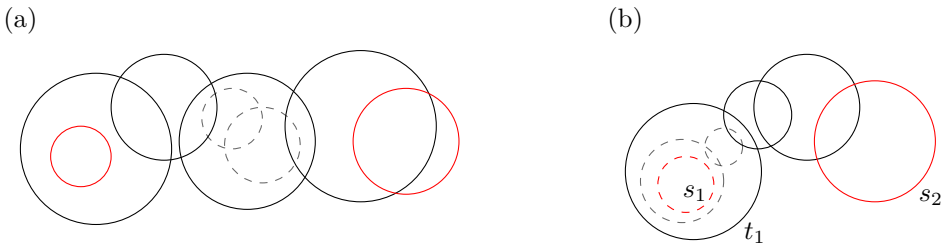

Figure 3 (a) A path between the two red disks uses the black disks as intermediates. Note that any non-red disk contained by a black disk is not required to form a path and can be safely ignored. (b) Removing the dashed disks and querying for t_1 instead of s_1 still leads to a valid path to s_2.

another disk of the path. Thus, all disks contained by t_1 (except s_2) can be removed without changing connectivity between t_1 and s_2, possibly including t_2; see Figure 3 (b). The removed disks include all those of Lemma 3.5 whose edges we exempted from the removal.

Still, for a representative this would require obtaining the largest disk fully containing a cell encountered on the path to a queried disk. Consider two different cells on the path down to s_1 and for each the largest disk which fully contains it but not its parent. The disk obtained at the larger cell must also fully contain the other chosen cell, but not the other way round. Thus, both disks intersect or the disk of the larger cell contains the other one. Therefore, we only need to retrieve the largest disk of the topmost cell of the path which was fully contained by some disk when replacing s_1 and s_2.

▶ **Lemma 3.6.** *Fix an instance of the data structure of Lemma 3.4 and two disks s_1, s_2. Let c_1, c_2 be the first cells on the paths to s_1, s_2 that are fully contained by a disk as topmost. Let s_1', s_2' be the largest such disks. A query for s_1, s_2 has the same result as a query for s_1', s_2' with all edges added solely through fully contained cells (i.e. non-zero counters) omitted.*

The dynamic nested rectangle intersection data structure by Kaplan et al. [6, Section 5] allows inserting or deleting nested or disjoint rectangles with a priority in amortized time $\mathcal{O}(\log^2 n)$ and retrieving the highest priority rectangle containing a query point in amortized time $\mathcal{O}(\log n)$, while requiring $\mathcal{O}(n \log n)$ space. We can use it to retrieve the representatives without a dependency on $\frac{1}{\varphi}$ in the query time by storing all fully contained topmost cells.

▶ **Theorem 3.7.** *Let $0 < \varphi \leq 1$, and let P be a set of disks with radius in $[\varphi, 1]$. There is a dynamic connectivity structure for the intersection graph of P such that the insertion or deletion of a disk takes amortized expected time $\mathcal{O}(\frac{1}{\varphi} 2^{\alpha(n)} \log^{10} n)$ and a connectivity query takes amortized time $\mathcal{O}(\log n)$, where n is the maximum number of sites at any time and $\alpha(n)$ is the inverse Ackermann function. It requires $\mathcal{O}(\frac{1}{\varphi} n \log^3 n)$ expected space.*

Proof. We extend the data structure of Lemma 2.3 similar to Lemma 3.4. Instead of a counter, we maintain for each cell an AVL tree [8] of all disks which fully contain this cell as topmost, ordered by radius. Also, we maintain in each node the disks count in its subtree.

In addition, we maintain a rectangle intersection data structure as by Kaplan et al., such that each cell with a non-empty AVL tree and a non-zero disk count in its subtree is inserted into the data structure with its size as priority. Due to this and Corollary 3.3, we have at most $\mathcal{O}(\min(n^2, \frac{1}{\varphi} n))$ cells in the rectangular intersection data structure simultaneously.

During a disk update we have $\mathcal{O}(\frac{1}{\varphi})$ updates to the rectangle structure via changes to the topmost fully contained cells and along the path in the quadtrees $\mathcal{O}(\log \frac{1}{\varphi})$ updates via

changes to counters. As each of these updates requires $\mathcal{O}(\log(n^2)) = \mathcal{O}(\log n)$ amortized time and we maintain fewer MBMs, the overall update time is reduced by a factor of $\frac{1}{\varphi}$.

Queries are done via finding the representatives described in Lemma 3.6 with the help of the rectangle data structure and the AVL trees in $\mathcal{O}(\log(n^2) + \log n) = \mathcal{O}(\log n)$ amortized time and then querying the connectivity structure as before.

We need $\mathcal{O}(\frac{1}{\varphi} n \log(n^2))$ space for the rectangular intersection data structure and $\mathcal{O}(\frac{1}{\varphi} n)$ for the trees and counters, but maintain fewer MBMs. Thus, less expected space is required. ◀

References

1 Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark H. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, third edition, 2008.

2 Timothy M. Chan, Mihai Pătraşcu, and Liam Roditty. Dynamic connectivity: Connecting to networks and geometry. *SIAM J. Comput.*, 40(2):333–349, 2011.

3 David Eppstein, Giuseppe F. Italiano, Roberto Tamassia, Robert Endre Tarjan, Jeffery Westbrook, and Moti Yung. Maintenance of a minimum spanning forest in a dynamic plane graph. *J. Algorithms*, 13(1):33–54, 1992.

4 Monika Rauch Henzinger and Valerie King. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *J. ACM*, 46:502–516, 1999.

5 Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM*, 48(4):723–760, 2001.

6 Haim Kaplan, Eyal Molad, and Robert E. Tarjan. Dynamic rectangular intersection with priorities. In *Proc. 35th Annu. ACM Sympos. Theory Comput. (STOC)*, pages 639–648, 2003.

7 Haim Kaplan, Wolfgang Mulzer, Liam Roditty, Paul Seiferth, and Micha Sharir. Dynamic planar Voronoi diagrams for general distance functions and their algorithmic applications. In *Proc. 28th Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA)*, pages 2495–2504, 2017. Full version at `arXiv:1604.03654`.

8 Donald E. Knuth. *The Art of Computer Programming. Vol. 3. Sorting and Searching*. Addison-Wesley, 2nd edition, 1998.

9 Mihai Pătraşcu and Mikkel Thorup. Planning for fast connectivity updates. In *Proc. 48th Annu. IEEE Sympos. Found. Comput. Sci. (FOCS)*, pages 263–271, 2007.

10 Paul Seiferth. *Disk Intersection Graphs: Models, Data Structures, and Algorithms*. PhD thesis, Freie Universität Berlin, Germany, 2016.

11 Mikkel Thorup. Near-optimal fully-dynamic graph connectivity. In *Proc. 32nd Annu. ACM Sympos. Theory Comput. (STOC)*, pages 343–350, 2000.