Dynamic Algorithms in Unit Disk Graphs

DALGO Lab

Pohang University of Science and Technology

November 10, 2023

Abstract

This research focuses on developing dynamic algorithms in unit disk graph for efficiently solving graph-based problems. First, we will present an efficient algorithm for the vertex cover problem. In our algorithm, updates require O(1) time, and we guarantee $2^{O(\sqrt{k})}$ time for queries.

1 Introduction

The time complexity of the vertex cover problem for planar graphs is known as $2^{O(\sqrt{k})}n^{O(1)}$ [1]. In our approach, we maintain the number of vertices in the kernel at $O(k)$, which ensures a time complexity of $2^{O(\sqrt{k})}$.

We employ a grid wherein each cell's diagonal is of unit length, thereby ensuring that all vertices in a single cell constitute a clique. When analyzing a given cell, 5×5 grid centered on that cell will be used. the term "cell around" and "neighborhood" refer to the adjacent 5×5 grid of cells surrounding a given cell. Cells are named with distinct names based on V_C, which is number of vertices in the cell C.

1. if V_C is greater than or equal to $k+2$, $saturated\,cell$
2. if V_C is greater than 2 and less than or equal to $k+1$, $friendly\,cell$
3. if $V_C = 1$, $lonely\,cell$
4. if $V_C = 0$, $empty\,cell$

2 Algorithm

In this section, we will explain our data structure and the algorithms for updates and queries.

2.1 Data structure

In order to dynamically maintain the kernel, we will preserve this data.

1. for each cell C, all the vertices in this cell and V_C
2. Number of $saturated\,cell$, N_S
3. Number of $friendly\,cell$, N_f
4. Set of $edges$ between lonely cells, E_l

2.2 Insertion

In this section, we will show how to maintain the kernel during the vertex insertion process.

First, V_C of the cell which contains the new vertex V_{new} is increased by 1. Determine cells that have become lonely cells from isolated lonely cells and include them in the kernel. If V_C is greater than $k+2$, we do nothing with the data structure. If V_C is equal to $k+2$, this means this cell is *friendly cell* before the insertion, but now it is *saturated cell*. So decrease N_f by 1 and increase N_s by 1. Also, delete this cell from the kernel. If V_C is greater than 3 and less than $k+1$, include V_{new} in the kernel. If V_C is equal to 2, increase N_f by 1 and include V_{new} and one other vertex in the kernel. Also, vertices in lonely cells within the surrounding 5×5 grid are also included in the kernel. Next, now this cell is no longer a lonely cell, the edges between lonely cells around this cell are removed from E_l. If V_C is equal to 1, check the surrounding 5×5 grid and calculating the edge between lonely cells to include these edges in E_l. Next, if at least one cell is *friendly cell*, include V_{new} in the kernel. If all the surrounding cells are *lonely cells* and *empty cells*, determine whether this vertex is isolated. if the vertex is isolated, do nothing. Else if the vertex is not isolated, include V_{new} in the kernel.

2.3 Deletion

In this section, we will show how to maintain the kernel during the vertex deletion process.

First, V_C of the cell which contains the deleted vertex V_{del} is decreased by 1 and delete V_{del} from the kernel. Determine the lonely cell that has become isolated by deleting the vertex and remove it from the kernel. If V_C is greater than or equal to $k+2$, do nothing. If V_C is equal to $k+1$, this means this cell is *saturated cell* before the deletion, but now it is *friendly cell*. So decrease N_s by 1 and increase N_f by 1. And this cell and all lonely cells around are added to the kernel. If V_C is greater than 1 and less than $k+1$, do nothing. If V_C is equal to 1, decrease N_f by 1. Check the surrounding 5×5 grid and calculate the edge between this cell and *lonely cells* to update E_l. If at least one cell is *friendly cell*, the remaining one vertex is still kept in the kernel as before. If all the surrounding cells are *lonely cells* and *empty cells*, determine whether this vertex is isolated. If the vertex is isolated, remove one remaining vertex from the kernel. Else if the vertex is not isolated, the vertex is still kept in the kernel as before. If V_C is equal to 0, determines the isolation of surrounding cells and if isolated, removes them from the kernel.

2.4 Query

For the vertex cover query, algorithm operates as follows.

If N_s is greater than or equal to 1, k vertex cover is impossible. If N_f is greater than k , k vertex cover is impossible. if $|E_l|$ is greater than $20k$, k vertex cover is impossible. Else, vertex cover algorithm is performed on the kernel that we maintain.

3 Correctness

We need to make sure that the vertex cover of the kernel is same as the vertex cover of the original graph. We also prove the bounds on the number of vertices in the kernel and analyze its time complexity.

3.1 Correctness of the algorithm

First, presence of a *saturated cell* implies we need to select at least $k+1$ vertices for vertex cover to address a clique consisting of $k+2$ or more vertices. Therefore, k vertex cover is impossible. Next, if the number of $friendly\,cell$ is greater than $k+1$, k vertex cover is impossible because at least one vertex in $friendly\,cell$ is selected for vertex cover, Last, if $|E_l|$ is greater than $20k$, more than k vertices are needed to cover this all edges because one *lonely cells* can cover up to 20 *lonely cells* around.

Except for the above cases, we have four kinds of edges: edges within the same cell, between $friendly\,cells$, between $friendly\,cells$ and *lonely cells*, between *lonely cells*. Because all the vertices in $friendly\,cells$ are included in the kernel, all edges between vertices in the same cell is covered. Similary, edges between $friendly\,cells$ are all included in the kernel. Edges between $friendly\,cells$ and *lonely cells* are covered by including lonely cells which have friendly cells as neighborhood in the kernel. Edges between lonely cells are covered because isolated vertex in *lonely cells* were removed from the kernel and the rest were included in the kernel.

3.2 Number of vertices in kernel

By summing up the number of the vertices in $friendly\,cells$ and *lonely cells*, we can get total number of vertices in the kernel.

For each $friendly\,cell$ C, $V_C - 1$ vertices must be included in the vertex cover. For k vertex cover, the size of vertex cover is less than or equal to k. So for all $friendly\,cells$, $\sum_C V_C - N_f \leq k$. Using the fact that N_f is less than $k+1$, the number of vertices in all $friendly\,cells$ is less than $2k+1$.

Lonely cells can be separated into two types depending on the presence of $friendly\,cells$ around. The number of $friendly\,cells$ is less than or equal to k, and each $friendly\,cell$ can only cover constant number of *lonely cells*, so the number of *lonely cells* which have at leat one $friendly\,cell$ as neighborhood is $O(k)$. *Lonely cell* that are surrounded by only *lonely cells* or *empty cells* has constant number of edge because we remove isolated *lonely cell* from kernel. In this case, $O(k)$ edges can exist between lonely cell, so the number of vertices in *lonely cell* is $O(k)$.

Through the above two processes, both of $friendly\,cells$ and *lonely cells* in the kernel contain $O(k)$ vertices. Finally, total number of vertices in the kernel is $O(k)$.

3.3 Time complexity

In the update process, dealing with V_C, N_f and N_S takes $O(1)$ time. Determining cell isolation takes $O(1)$ time because we only need to consider constant number of lonely cells around it. Similary, finding the $friendly\,cells$ and *lonely cells* around the cell also takes constant time. Calculating the edge between *lonely cells* only requires checking constant number of surrounding cells. So each update takes $O(1)$ time.

In 3.2 we show that the total number of vertices in the kernel is $O(k)$. So $2^{O(\sqrt{k})}$ is guaranteed for each query.

References

[1] Erik D Demaine, Fedor V Fomin, Mohammadtaghi Hajiaghayi, and Dimitrios M Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and h-minor-free graphs. *Journal of the ACM (JACM)*, 52(6):866–893, 2005.