https://doi.org/10.1007/s11227-020-03226-w

Clustering-Based Force-Directed Algorithms for 3D Graph Visualization

Jiawei Lu, Yain-Whar Si[1]
Department of Computer and Information Science, University of Macau
{gerrylow2019@gmail.com, fstasp@umac.mo}

Abstract

Force directed algorithm is one of the most commonly used methods for visualization of 2D graphs. These algorithms can be applied to a plethora of applications such as data visualization, social network analysis, crypto-currency transactions, and wireless sensor networks. Due to their effectiveness in visualization of topological data, various force directed algorithms for 2D graphs were proposed in recent years. Although force directed algorithms for 2D graphs were extensively investigated in research community, the algorithms for 3D graph visualization were rarely reported literature. In this paper, we propose four novel Clustering-based Force-directed (CFD) algorithms for visualization of 3D graphs. By using clustering algorithms, we divide a large graph into many smaller graphs so that they can be effectively processed by Force directed algorithms. In addition, weights are also introduced to further enhance the calculation for clusters. The proposed CFD algorithms are tested on 3 datasets with varying number of nodes. The experimental results show that proposed algorithms can significantly reduce edge crossings in visualization of large 3D graphs. The results also reveal that CFD algorithms can also reduce Kamada and Kawai (KK) energy and standardized variance of edge lengths in 3D graph visualization.

Keywords: 3D graphs; visualization; clustering algorithm; force-directed algorithm.

1. Introduction

With the advancement of Internet and social media, development of efficient methods for analyzing large scale networks become increasingly important in data analytics and visualization community. In recent years, various types of algorithms related to visualization of social networks were proposed in literature [1]. Among these algorithms, force-directed placement (FDP) algorithms have been widely used in visualization of networks [2]. In additions FDP algorithms are also used in plethora of applications such as data visualization, social network analysis, crypto-currency transactions, and wireless sensor networks. FDP algorithms evolve from the idea of "spring embedding" force-directed placement in which nodes can repel or attract other nodes that are connected to them by edges, effectively acting as springs [3]. Some of the well-known FDP algorithms include Fruchterman and Reingold (FR) [4] (which is based on the work of Eades [5]) and Kamada and Kawai (KK) [6]. However, due to the high computational cost, majority of these algorithms do not perform well when dealing with graphs that have more than 2000 vertices.

Although force directed algorithms for 2D graphs were extensively investigated in research community, force directed algorithms for 3D graph visualization were rarely reported literature. 3D visualization can convey more information when it is compared to 2D visualization since it has additional depth information. To the best of authors' knowledge, there are only a few FDP algorithms designed for drawing 3D graphs. Among them, original FR algorithm was extended for 3D drawing in

[1] Corresponding author. Email: fstasp@umac.mo, Address: Department of Computer and Information Science, University of Macau, Avenida da Universidade, Taipa, Macau, China. Tel: (853) 88224454

software applications such as Pajek [7] and Cytoscape [8]. In our preliminary testing using Pajek, the 3D version of FR stalled when processing the input topology containing more than 200 vertices. In our testing with various distances between 1-hop neighbors, the algorithm often fails to place some of the nodes to reasonable positions. From the output, we also observed that some of the nodes were interlaced with each other, while some of the neighboring nodes were placed too far away from each other. In addition, layouts that Pajek produced often have some nodes that are too close to each other and. Besides, in Cytoscape, layouts are often too dense in the center where many nodes reside. Therefore, the quality of layout degrades significantly when the graph contains thousands of nodes.

To alleviate these problems, a set of four novel Clustering-based Force-directed (CFD) algorithms for drawing 3D graphs is proposed in this paper.

1. CFD-FB which is based on weighted FR and BigCLAM [9] clustering algorithm.
2. CFD-FM which is based on weighted FR and MCL [10] clustering algorithm.
3. CFD-KB which is based on weighted KK and BigCLAM clustering algorithm.
4. CFD-KM which is based on weighted KK and MCL clustering algorithm.

Inspired by the principle of divide and conquer, the proposed approach aims to divide a large graph into several smaller graphs so that they can be manipulated by a FDP. To achieve this objective, two clustering algorithms are adopted to extend the force directed algorithms. Although any clustering algorithm can be adopted for this purpose, algorithms that can handle thousands of nodes efficiently are selected in our approach. To this end, BigCLAM and MCL clustering algorithms are integrated with 3D versions of FR and KK algorithms in the proposed CFD.

By using clustering algorithms, each cluster from the input topology is transformed into individual small graphs (or layouts). Next, we design weighted 3D versions of FR and KK algorithms (weighted FDP) for stitching these graphs back together to form a complete layout of the given input. Specifically, weighted FDP algorithms treats clusters as nodes with varying sizes. In addition, the edges between two clusters are treated as an edge labelled with weight. By using weighted FDP algorithms, an initial layout about clusters can be calculated before each small graph can be placed at the positions of their corresponding clusters. To further improve the quality of the final output, the original (unweighted) versions of FDPs are again deployed in fine-tuning step.

The CFD algorithms proposed in this paper aim to reduce edge crossings significantly. By reducing edge crossings, a cleaner view of the layout can be achieved. The proposed algorithms also reduce KK energy and standardized variance of edge lengths. By reducing standardized variance, more uniform edge lengths can be obtained. Our experiment results show that CFD-FB achieves the best performance among the four algorithms proposed. Compared with other CFD algorithms, CFD-FB is also less computational expensive and achieves the best results when handling large graphs.

This paper is organized as follows. In section 2, we briefly review the related work. In section 3, we introduce the attributes that are used to evaluate the quality of the 3D layouts. In section 4, we describe the CFD algorithms for 3D graph visualization. In section 5, we evaluate the performance of CFD algorithms. We conclude the paper in section 6 with future work.

2. Related work

Fruchterman and Reingold (FR) [4] and Kamada and Kawai (KK) [6] algorithms are commonly used for generating 2D layouts of unstructured, unpartitioned, and unweighted graphs. Both FR and KK

initialize a layout randomly from a graph. FR algorithm is based on two principles: nodes repulse each other, and edges attract the nodes they connected. For each iteration in this algorithm, the displacement of each node is calculated based on its position with all other nodes and the nodes it connects by edges. KK algorithm is based on a spring model in which nodes are mutually connected by springs. The KK energy of each spring is calculated from the difference of its desirable length and its actual length. The goal of the algorithm is to minimize KK energy by moving nodes to stable positions one at a time. The algorithm terminates after the squared sum of partial derivatives of KK energy of every node is under a preset threshold. To apply FR and KK for 3D drawing and weighted graphs, several modifications are introduced recently. These modifications include gravitational force introduced to FR by Frick et al. [11] and social-gravity force introduced by Bannister et al. [12].

However, computation cost (complexity) of KK algorithm for generating a layout for a large graph is extremely high. KK algorithm often fails to produce any output when processing graphs containing thousands of nodes. As for the FR algorithm, the computational cost is lower than KK algorithm. However, the graphs generated by the FR algorithm usually contain large number of tangled edges.

In [13], Davison and Harel proposed an algorithm based on the paradigm of simulated annealing for drawing 2D layouts. The algorithm incorporates four different criteria into a cost function. The four criteria considered are: (1) distributing nodes evenly, (2) making edge-lengths uniform, (3) minimizing edge-crossings, and (4) keeping nodes from coming too close to edges. Some of those criteria are incompatible with each other. Thus, weights for different criteria in the cost function are often needed to find a trade-off among them.

Jacomy et al. developed ForceAtlas2 [14] for Gephi by combining existing techniques. ForceAtlas2 is a modification of FR and it also utilizes gravity in its force calculation. ForceAtlas2 calculates attraction force using original method but uses node degree in addition for the calculation of repulsive force. The formula for repulsive force calculation is similar to the edge repulsion proposed by Noack [15]. ForceAtlas2 [14] was used in social networks visualization. However, ForceAtlas2 was originally designed for generating 2D layouts whereas in our approach, the proposed CFD algorithms were designed for producing 3D layouts.

In [16], Christ Walshaw proposed a multilevel algorithm which is similar to the idea of divide-and-conquer approach. Christ Walshaw used FR as the FDP algorithm to produce 3D layouts. However, the algorithm proposed in [16] is for drawing layouts of structured graphs. Yifan Hu also proposed an algorithm [17] for structured graphs. However, the experiments about the comparison of drawings of the graphs in [17] were only performed in 2D. In contrast to their approaches, in this paper, the quality of 3D output layouts generated by the proposed CFD algorithm is evaluated in terms of standardized variance, KK energy, and total number of edge crossings.

Lin and Yen [18] proposed a force-directed graph drawing method based on edge-edge repulsion. The proposed method was evaluated for both 2D and 3D layouts in [18]. In the proposed method, instead of using repulsion force, edges are modelled as charged springs. The repulsion force between two adjacent edges becomes stronger when the edges become closer. The method proposed by Lin and Yen was designed to preserve the original properties of a high degree of symmetry and uniform edge length. The algorithm was also designed to prevent zero angular resolution. The layouts generated by their approach usually have larger average angular resolution. In contrast to their approach, the CFD algorithms proposed in this work are designed to minimize edge crossings and KK energy of the resulted graphs.

In [19], Arleo et al. proposed a distributed multilevel force-directed algorithm called Multi-GiLA. The algorithm includes three main phases: coarsening, placement, and single-level layout. The proposed algorithm was designed for execution on a distributed computer network. The coarsening phase in Multi-GiLA is similar to the clustering phase of our proposed CFD algorithms. However, instead of

using a clustering algorithm to generate only one higher-level graph, in which each node represents a cluster, the coarsening phase produces a stack of graphs (multi-level).

Suh et al. [20] proposed a method that leverages persistent homology features of an undirected graph. In their approach, users can interactively control the layout to emphasize user-selected aspects of the graph. In contrast to their approach, the CFD algorithms proposed in this paper do not provide mechanisms for interacting with the generated layout.

Recently, S.H. Hong et al. [21] proposed a multi-level graph drawing algorithm based on Infomap clustering [22] which computes the clusters by translating a graph into a map. Next, the algorithm performs initial placement based on three variations: circle, barycenter, and zero placement. In the refinement stage, multi-level force directed algorithms such as FR [4] and FRG [23] are used for generating the final layout.

In addition to these force-directed approaches, various computation methods are also used for generating graphs. For example, in [24], Wang et al. proposed a deep learning-based approach for graph drawing. In [25], Robert Gove proposed a random sampling force-calculation algorithm. First, the algorithm uses Random Vertex Sampling (RVS) to derive good initial positions. Next, it uses Barnes-Hut simulation [26] to refine the layout. The algorithm proposed by Robert Gove is based on divide and conquer principle. In contrast to the approach proposed by Robert Gove, our proposed algorithms use clustering as a pre-processing step.

Apart from the work of S.H. Hong et al. [21], there is no reported work on augmenting FDP algorithm with clustering algorithms for 3D graphs. The most similar one being the work of Christ Walshaw's [16], in which a multilevel algorithm and FR are used to produce 3D layouts. In our work, we propose a set of novel CFD algorithms based on FR and KK. One of the main objectives of our proposed methods is to reduce edge crossings in 3D layouts. In addition, the multilevel algorithm proposed in [16] is designed for drawing structured graphs whereas the CFD algorithms in this paper are designed for drawing unstructured graphs.

3. Preliminaries

In this section, we describe the criteria for evaluating the layouts generated from CFD algorithms. In our work, the number of edge crossings, standardized variance of edge lengths, and Kamada–Kawai energy (KK energy) are used to assess the quality of a layout.

Edge Crossing: When a 3D layout is projected to a computer screen, it resembles to a 2.5D layout (a 2D layout with additional depth information). Edge crossing happens when two edges from two pairs of nodes intersect with each other in the 2D view. Thus, like a 2D layout, edge crossings from a certain viewing angle could induce visually unaesthetic outputs. Edges seldom cross with each other in a 3D space in the graph layout. However it could be less useful to detect all edge crossings after the layout is projected to a 2D display, since depth information can be used to illustrate the distance between nodes and viewpoint, as shown Figure 1. Therefore, in this paper, we only detect close enough edges. If the shortest distance between two edges is smaller than a threshold, these two edges are counted..

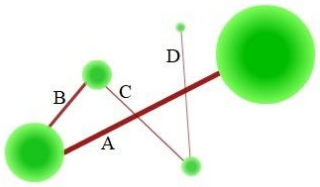


Figure 1: Two pairs of edges (A and C, A and D) are less likely to interfere with each other if they are separated a certain distance from each other in depth dimension

The shortest distance d between two edges G_1 and G_2 (line segments) can be defined as follows:

$$d(G_1, G_2) = \min_{P \in G_1, Q \in G_2} d(P, Q) \quad (1)$$

where point P and Q are on G_1, G_2 respectively and the distance between P and Q is minimal. In this paper, the numerical method for calculating the shortest distance is not used due to the computation cost. Instead, a mathematical method for calculating the shortest distance between two line segments [27] is adopted since it can provide an accurate solution with less computation cost.

Standardized variance of edge lengths: For a graph layout, the variance of edge lengths can be used to measure the variability of distances among neighboring nodes. High variance may indicate that the neighboring nodes are misplaced, or they are either too close or too far away from each other. The standardization is done by dividing variance by the squared average length of edges since using longer edges in a layout can increase the variance. The variance can be calculated as follows:

$$Var(X) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2, \mu = \frac{1}{n}\sum_{i=1}^{n} x_i \quad (2)$$

where x_i is the length of the i^{th} edge. Based on the equation (2), standard variance of edge lengths can be calculated as follows:

$$std\_Var(X) = Var(X)/\mu^2 \quad (3)$$

The distance between each 2 nodes: Neighboring nodes from a graph should be placed closer to each other in order to achieve an even distribution of nodes. The ideal distance between any 2 nodes can be calculated based on the number of nodes in the closest path connecting these nodes [6]. That is, the more nodes on the path, the longer the distance for those 2 nodes. Besides, according to [4], all nodes should not be placed too close to each other. Kamada and Kawai [6] proposed an energy function to evaluate the degree of imbalance of a graph layout. The energy of a graph layout E is the sum of energy of all nodes in that graph.

$$E = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\frac{1}{2}k_{ij}\left(\left|p_i - p_j\right| - l_{ij}\right)^2 \quad (4)$$

$$l_{ij} = L \times d_{ij}, L = L_0 / \max_{i<j} d_{ij}$$

$$k_{ij} = K/d_{ij}^2$$

where L_0 is the length of a side of screen and K is a constant. The energy function is based on a spring model. l_{ij} is the original length of the spring that connects node i and j. The original length is also the desirable distance between these two nodes in the graph layout. $\left|p_i - p_j\right|$ is the actual length of the spring connected by node i and j. k_{ij} is the strength of the spring between node i and j. d_{ij}^2 is the length of the shortest path between node i and j. The length of the shortest path between two nodes is 1 if these two nodes are neighboring nodes. If two nodes are not directly connected with each other, the length of the shortest path is $1 + \min_{numNodes} numNodes$, where $numNodes$ is the number of nodes on the path that connect these nodes. Since multiple paths may exist, the path with the fewest number of nodes is chosen. In this paper, KK energy function from equation (4) is adopted as one of the criteria

for evaluating the quality of the output graph since KK energy reflects how evenly the nodes are distributed. Since the length of the shortest path is involved, the shortest path problem needs to be solved before the energy function could be computed. In this paper, Floyd–Warshall algorithm [28] is used to find the shortest paths in a weighted graph. Since we do not consider weights in the final layout, the weight for each edge is set to 1 in the algorithm.

4. Clustering-based force-directed algorithm for 3D graph visualization

In this section, we describe the proposed CFD algorithms in detail. These CFD algorithms are implemented based on two well-known FDP algorithms and clustering algorithms.

4.1 Overview of the CFD algorithm

Clustering-based force-directed (CFD) algorithms are designed to accept a topology defined in a graph description format such as graph modeling language (GML) [29]. The main objective of CFD algorithms is to generate a 3D graph layout from the given topology. The overview of CFD algorithms is depicted in Figure 2. The input to the overall process is a file containing a graph topology. The output of the CFD algorithm is a layout that contains information about the positions (coordinates) of every node. CFD algorithms are able to produce outputs in KiNG (.kin) [30] format.

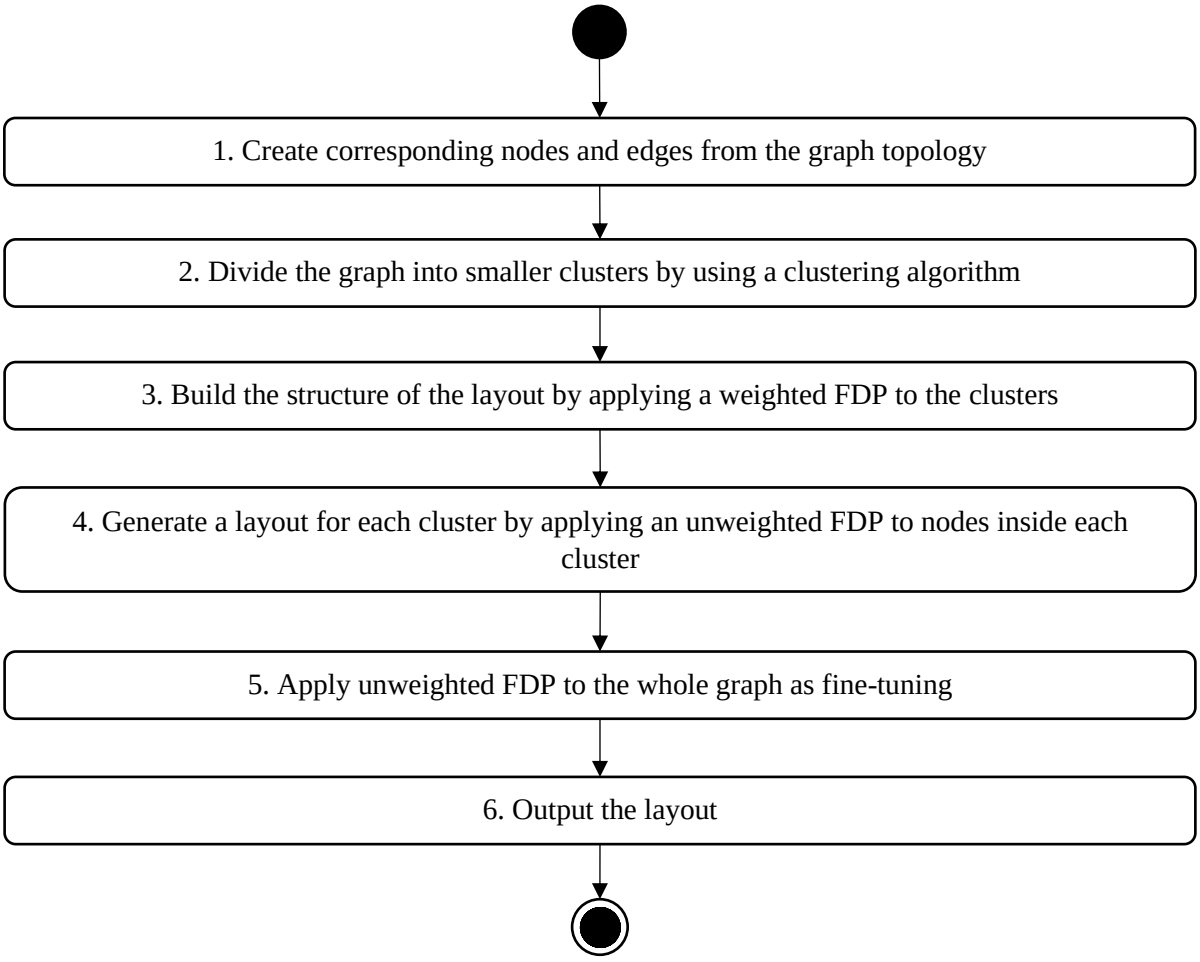


Figure 2: Overview of clustering-based force-directed (CFD) algorithms

The steps of a CFD algorithm are described as follows:

1. Nodes and edges are created based on the input graph topology.
2. A clustering algorithm is applied to the input graph topology. The purpose of applying the clustering algorithm to the graph topology is to divide the graph into a number of subgraphs. Dividing the input graph into multiple subgraphs reduces the computational cost of force-directed algorithms in later stages of the algorithm. The clustering algorithms considered in our approach are BigCLAM and MCL clustering algorithms.
3. After the clustering operation, a weighted FDP algorithm is applied to the entire graph in this step. In this research, we developed weighted version of FDP algorithms by considering the *size* of the nodes and *weight* of the edges. These weighted version of FDP algorithms are 3D version of weighted FR and KK algorithms. In these algorithms, each subgraph (cluster) is treated as a node and the size of the node is calculated based on the number of nodes inside that subgraph. Thus, a subgraph with more nodes will be pushed farther away from other subgraphs. Next, the edges between two subgraphs are merged to become a single edge and its weight is calculated based on the number of edges between the subgraphs. Therefore, any increase in the number of edges between two subgraphs can make them closer.
4. In this step, an unweighted FDP algorithm is applied to each subgraph to generate a 3D layout. An unweighted FDP is the original version of 3D force directed algorithm that does not consider the size of the nodes and weights of the edges. The unweighted FDP algorithms used in our approach are 3D version of FR and KK algorithms. After these subgraphs are processed, each output is then put back to their corresponding places.
5. In this step, an unweighted FDP algorithm is again applied to the entire graph for fine-tuning.
6. After the fine-tuning, the final 3D graph is output in a graph layout definition file.

Algorithm 1 is the pseudo-code of the proposed CFD algorithm. In this paper, four CFD algorithms (CFD-FB, CFD-FM, CFD-KB, CFD-KM) are implemented based on Algorithm 1.

Algorithm 1: Clustering-based force-directed algorithm

CFD
Read a file to parse its graph topology and create the corresponding nodes and edges
Grouping ← clustering // run the clustering algorithm
// grouping stores lists of node indices; each list represents a cluster
// gNodes is a list of nodes, each representing a cluster (the size of the node is the number of nodes inside that cluster, the coordinates of the node are the coordinates of the center of that cluster); gEdges is a list of edges, each representing an edge between two clusters (the weight of the edge is the number of edges between those two clusters)
Execute weighted FDP for clusters with gNodes and gEdges as argument for each group in grouping
Execute original FDP for nodes in each cluster
Place the nodes in each cluster according to the center of that cluster
Execute original FDP for the whole graph as fine-tuning
Output evaluation results and layout

4.2 Clustering algorithms

A clustering algorithm can divide a large graph into several clusters. After the clustering, the number of nodes in each cluster is much smaller compared to the original graph. Several clustering algorithms were proposed in recent years. These algorithms include BigCLAM clustering algorithm [9], Markov Clustering (MCL) [10] and Restricted Neighborhood Search Clustering (RNSC) [31]. In this paper, BigCLAM and MCL are selected for clustering the input graphs.

4.3 Weighted FDP for 3D graphs

In this section, we introduce two weighted force-directed placement algorithms for 3D graphs.

4.3.1 Weighted Fruchterman and Reingold algorithm for 3D graphs

Fruchterman and Reingold algorithm [4] is a well-known FDP algorithm. According to [4], FR algorithm generates the layout of the graphs by considering two principles for graph drawing: nodes connected by an edge should be drawn near each other, and nodes should not be drawn too close to each other. FR algorithm uses repulsive force to push away nodes that are too close from each other, and attractive force to pull neighboring nodes closer. It also uses the concept of temperature to limit the maximum distance a node can move during each iteration. Repulsive forces are applied for all neighboring nodes. An attractive force is applied to every two nodes. Attractive force $f_a(d)$ and repulsive force $f_r(d)$ can be calculated as follows:

$$f_a(d) = \frac{d^2}{k} \tag{5}$$

$$f_r(d) = -\frac{k^2}{d} \tag{6}$$

where d is the distance between the two nodes being calculated and k is the optimal distance between those two nodes. The distance between two neighboring nodes is optimal when the repulsive force and the attractive force relating to those two nodes cancel each other. That means:

$$f_a(d) = -f_r(d) \tag{7}$$

If (7) holds, then $d = k$.

Thus, k can be used to represent the optimal distance desired for the graph layout. In [4], k is set as $k = C\sqrt{\frac{area}{|n|}}$, where *area* is the extent of the screen, $|n|$ is the number of nodes, and C is a constant determined by experiments. According to [4], C is set to 1. Note that for 3D drawings, k can be set as $k = C\sqrt[3]{\frac{volume}{|n|}}$.

The FR algorithm iteratively calculates the repulsive force and the attractive force of every two nodes that are connected by an edge. Next, the calculated forces are applied to the corresponding nodes and the maximum displacement is limited by the "temperature" value at each iteration. The algorithm terminates after a specified number of iterations. Temperature is a variable used for constraining the maximum displacement for a node during each iteration. Instead of using a cooling function, "gravity" is introduced in our experiments. The gravity is a force that pulls all nodes toward the center during each iteration. The purpose of gravity is to keep the nodes as compact as possible. By using gravity, maintaining a drawing space boundary is also no longer required. Therefore, the optimal unit length

(k) of an edge is no longer determined by screen length and the number of nodes. k is set to 10 in our experiments. Besides, cooling schedule is also no longer required since gravity is a constant. Therefore, temperature is set to $k/5$ throughout the iterations.

When the gravity is not used, the nodes in a graph layout can be sparse. On the other hand, if a high gravity value is used, it can decrease the quality of the graph since excessive gravity can overwhelm the repulsive force between nodes and can lead to the collapse of the graph. Therefore, it is crucial to choose the right gravity for the graph. Gravity can be computed by using the following equation:

$$f_g(n) = \gamma G(\xi - P[n]) \tag{8}$$

where ξ is the centroid of the points; G is a gravitational constant and γ is a constant. In this paper, γ is set as 0.1. $P[n]$ is the coordinate vector of node n. G is set to 1.0.

Pseudo-code of 3D version of FR algorithm with gravity is given in Algorithm 2. In this paper, ξ (the centroid of the points) is fixed at the center of the drawing space instead of being calculated each iteration.

Algorithm 2: 3D FR algorithm with gravity

```
FR
    let k be the optimal distance for all neighboring nodes
    k ← 10 // with gravity, k is set as 10, and drawing space boundary implementation is not needed
    let nodes be the list of all nodes
    let edges be the list of all edges
    let temperature be the maximum displace of any node and initialized to k/5
    let numIterations be the number of iterations for the FR algorithm and initialized to 100
    let γ_t be a gravitational scaling parameter
    let gravity be a gravitational constant and initialized as 1.0
    let γ be a constant and set as 0.1
    for iteration from 1 to numIterations
        // calculate repulsive force
        for i from 1 to |nodes|
            nodes.i.disp ← 0
            for j from 1 to |n|
                if i != j
                    Δ ← nodes.i.pos – nodes.j.pos
                    // ignore repulsive force calculation for two nodes that have a
distance greater than 2 * k (the double of optimal distance for neighboring nodes) since the force between them is
ignorable
                    if Δ <= 2 * k
                        nodes.i.disp ← nodes.i.disp + (Δ/|Δ|) f_r(|Δ|)
        // calculate attractive force
        for edge in edges
            Δ ← edges.source.pos – edges.target.pos
            temp_disp ← (Δ/|Δ|) f_a(|Δ|)
            edges.source.disp ←edges.source.disp - temp_disp
            edges.target.disp ←edges.target.disp + temp_disp
        // calculate gravity
        let gf be the value of the gravitational force
        for node in nodes
            gf ← gravity * γ * k * node.pos
            node.disp ← node.disp - gf
        // apply displacements
        for node in nodes
            node.pos ← node.pos + (node.disp/|node.disp|) min(node.isp, temperature)
```

In this paper, we further extend the 3D version of FR algorithm given in Algorithm 2 by using weights. In the weighted version, the repulsive force and attractive force between nodes are defined differently. We further extend the 3D version of FR algorithm by using size of the node sizes and weights of the edges. First, clusters are created after a graph is partitioned by a clustering algorithm. The resulting clusters are then replaced by nodes. Since clusters can have different sizes depending on the number of nodes inside each cluster, each node after replacement is labelled with the corresponding size. Second, the number of edges between any 2 clusters can be calculated by counting the total number of edges connecting the nodes from the first cluster to the second cluster. These connecting edges between two clusters are replaced with a single edge. The weight of the edge equals to the total number of connecting edges being replaced. However, it is possible that there are no connecting edges between two clusters.

According to the above calculation, we can observe that bigger nodes require more space in the layout. Therefore, the repulsive force associated with this node should be increased. The weight of an edge can also affect the attractive force between the corresponding two nodes. The higher the weight, the stronger the attractive force. The repulsive and attractive forces are defined in weighted FR for 3D as follows:

$$f_a(d) = \frac{d^2}{k} w \tag{9}$$

$$f_r(d) = -\frac{k^2}{d}\left(\frac{\sqrt{s_1}+\sqrt{s_2}}{2}\right) \tag{10}$$

where d is the distance between the two nodes being calculated, w is the weight of the edge between those 2 nodes, and s_1, s_2 are the sizes of these two nodes.

4.3.2 Weighted Kamada and Kawai algorithm for 3D graphs

In this paper, we further extend the 3D version of Kamada and Kawai (KK) algorithm [6] using weights. KK is a force directed algorithm based on a spring model. The degree of imbalance E in KK algorithm can be calculated as follows:

$$E = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\frac{1}{2}k_{ij}\left(\left|p_i - p_j\right| - l_{ij}\right)^2 \tag{11}$$

$$l_{ij} = L \times d_{ij},\ L = L_0 / \max_{i<j} d_{ij}$$

$$k_{ij} = K / d_{ij}^2$$

where L is the optimal length of a single edge; K is a constant, d_{ij} is the graph distance between node i and j, and l_{ij} is the optimal distance of node i and j. d_{ij} can be calculated by Floyd–Warshall algorithm [28] for finding the shortest paths.

In [6], L is calculated as $L = L_0 / \max_{i<j} d_{ij}$, where L_0 is the length of a side of the screen. In the experiments, neither a constant drawing space nor the constrained drawing space is used. The main objective is to keep the optimal unit length of an edge invariable throughout the experiment. Therefore, similar to FR, the optimal unit length of an edge is set as 10.

In this research, we developed a 3D KK algorithm by extending the formulation given in [6, 32]. For the 3D KK algorithm, the coordinate variable of p_i is (x_i, y_i, z_i). Thus:

$$E = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\frac{1}{2}k_{ij}\left(\left(x_i - x_j\right)^2 + \left(y_i - y_j\right)^2 + \left(z_i - z_j\right)^2 + l_{ij}^2 - 2l_{ij}\sqrt{\left(x_i - x_j\right)^2 + \left(y_i - y_j\right)^2 + \left(z_i - z_j\right)^2}\right) \quad (12)$$

For a node m ($1 \le m \le n$, n is the total number of nodes in a graph), the condition for a local minimum is

$$\frac{\partial E}{\partial x_m} = \frac{\partial E}{\partial y_m} = \frac{\partial E}{\partial z_m} = 0 \quad (13)$$

The followings are the partial derivatives of E by x_m, y_m and z_m respectively.

$$\frac{\partial E}{\partial x_m} = \sum_{i\neq m} k_{mi}\left((x_m - x_i) - l_{mi}\frac{(x_m - x_i)}{\sqrt{(x_m-x_i)^2+(y_m-y_i)^2+(z_m-z_i)^2}}\right) \quad (14)$$

$$\frac{\partial E}{\partial y_m} = \sum_{i\neq m} k_{mi}\left((y_m - y_i) - l_{mi}\frac{(y_m - y_i)}{\sqrt{(x_m-x_i)^2+(y_m-y_i)^2+(z_m-z_i)^2}}\right) \quad (15)$$

$$\frac{\partial E}{\partial z_m} = \sum_{i\neq m} k_{mi}\left((z_m - z_i) - l_{mi}\frac{(z_m - z_i)}{\sqrt{(x_m-x_i)^2+(y_m-y_i)^2+(z_m-z_i)^2}}\right) \quad (16)$$

Δ_m is defined as

$$\Delta_m = \sqrt{\left(\frac{\partial E}{\partial x_m}\right)^2 + \left(\frac{\partial E}{\partial y_m}\right)^2 + \left(\frac{\partial E}{\partial z_m}\right)^2} \quad (17)$$

The following system of equations are derived by using Newton-Raphson method [33] for nonlinear systems of equations.

$$\begin{cases} \frac{\partial^2 E}{\partial x_m^2}\left(x_m^{(t)}, y_m^{(t)}, z_m^{(t)}\right)\delta x + \frac{\partial^2 E}{\partial x_m y_m}\left(x_m^{(t)}, y_m^{(t)}, z_m^{(t)}\right)\delta y + \frac{\partial^2 E}{\partial x_m z_m}\left(x_m^{(t)}, y_m^{(t)}, z_m^{(t)}\right)\delta z = -\frac{\partial E}{\partial x_m}\left(x_m^{(t)}, y_m^{(t)}, z_m^{(t)}\right) \\ \frac{\partial^2 E}{\partial y_m x_m}\left(x_m^{(t)}, y_m^{(t)}, z_m^{(t)}\right)\delta x + \frac{\partial^2 E}{\partial y_m^2}\left(x_m^{(t)}, y_m^{(t)}, z_m^{(t)}\right)\delta y + \frac{\partial^2 E}{\partial y_m z_m}\left(x_m^{(t)}, y_m^{(t)}, z_m^{(t)}\right)\delta z = -\frac{\partial E}{\partial y_m}\left(x_m^{(t)}, y_m^{(t)}, z_m^{(t)}\right) \\ \frac{\partial^2 E}{\partial z_m x_m}\left(x_m^{(t)}, y_m^{(t)}, z_m^{(t)}\right)\delta x + \frac{\partial^2 E}{\partial z_m y_m}\left(x_m^{(t)}, y_m^{(t)}, z_m^{(t)}\right)\delta y + \frac{\partial^2 E}{\partial z_m^2}\left(x_m^{(t)}, y_m^{(t)}, z_m^{(t)}\right)\delta z = -\frac{\partial E}{\partial z_m}\left(x_m^{(t)}, y_m^{(t)}, z_m^{(t)}\right) \end{cases} \quad (18)$$

where:

$$\frac{\partial^2 E}{\partial x_m^2} = \sum_{i\neq m} k_{mi}\left(1 - l_{mi}\frac{(y_m-y_i)^2+(z_m-z_i)^2}{((x_m-x_i)^2+(y_m-y_i)^2+(z_m-z_i)^2)^{3/2}}\right) \quad (19)$$

$$\frac{\partial^2 E}{\partial y_m^2} = \sum_{i\neq m} k_{mi}\left(1 - l_{mi}\frac{(x_m-x_i)^2+(z_m-z_i)^2}{((x_m-x_i)^2+(y_m-y_i)^2+(z_m-z_i)^2)^{3/2}}\right) \quad (20)$$

$$\frac{\partial^2 E}{\partial z_m^2} = \sum_{i\neq m} k_{mi}\left(1 - l_{mi}\frac{(x_m-x_i)^2+(y_m-y_i)^2}{((x_m-x_i)^2+(y_m-y_i)^2+(z_m-z_i)^2)^{3/2}}\right) \quad (21)$$

$$\frac{\partial^2 E}{\partial x_m \partial y_m} = \sum_{i\neq m} k_{mi} l_{mi}\frac{(x_m-x_i)(y_m-y_i)}{((x_m-x_i)^2+(y_m-y_i)^2+(z_m-z_i)^2)^{3/2}} \quad (22)$$

$$\frac{\partial^2 E}{\partial x_m \partial z_m} = \sum_{i\neq m} k_{mi} l_{mi}\frac{(x_m-x_i)(z_m-z_i)}{((x_m-x_i)^2+(y_m-y_i)^2+(z_m-z_i)^2)^{3/2}} \quad (23)$$

$$\frac{\partial^2 E}{\partial y_m \partial x_m} = \sum_{i\neq m} k_{mi} l_{mi}\frac{(x_m-x_i)(y_m-y_i)}{((x_m-x_i)^2+(y_m-y_i)^2+(z_m-z_i)^2)^{3/2}} \quad (24)$$

$$\frac{\partial^2 E}{\partial y_m \partial z_m} = \sum_{i \neq m} k_{mi} l_{mi} \frac{(y_m - y_i)(z_m - z_i)}{((x_m - x_i)^2 + (y_m - y_i)^2 + (z_m - z_i)^2)^{3/2}} \quad (25)$$

$$\frac{\partial^2 E}{\partial z_m \partial x_m} = \sum_{i \neq m} k_{mi} l_{mi} \frac{(x_m - x_i)(z_m - z_i)}{((x_m - x_i)^2 + (y_m - y_i)^2 + (z_m - z_i)^2)^{3/2}} \quad (26)$$

$$\frac{\partial^2 E}{\partial z_m \partial y_m} = \sum_{i \neq m} k_{mi} l_{mi} \frac{(y_m - y_i)(z_m - z_i)}{((x_m - x_i)^2 + (y_m - y_i)^2 + (z_m - z_i)^2)^{3/2}} \quad (27)$$

Then the unknown δx, δy and δz can be computed from the system of equations. In the implementation, Gaussian elimination is adopted. After that, displacement (δx, δy, and δz) will be applied to the node and new displacement will be computed again using Newton-Raphson method. The iteration of the computation of δx, δy and δz terminates when Δ_m at $\left(x_m^{(t)}, y_m^{(t)}, z_m^{(t)}\right)$ is less than the threshold ε. KK algorithm terminates when there is no Δ_m for any node m for which $\Delta_m > \varepsilon$.

Next, we further extend the 3D KK by using weights. Similar to weighted FR for 3D, we perform clustering as well as nodes and edges replacement. Since resulting clusters can have different sizes, the ideal distance among nodes need to be adjusted with respect to the sizes of the clusters. Therefore, we modify l_{ij} from equation (11) as follows:

$$l_{ij} = L \times d_{ij} \left(\frac{\sqrt{s_i} + \sqrt{s_j}}{2}\right) \quad (28)$$

where d_{ij} is the graph distance between node i and j; and s_i, s_j are the sizes of node i and j. In this paper, we adopt Floyd–Warshall algorithm [28] for finding shortest paths in a weighted graph. The algorithm can also be used for computing graph distances. w_{st} is the weight of the edge connecting node s and t. The computation of d_{ij} with Floyd–Warshall algorithm based on w_{st} is described in Algorithm 3.

Algorithm 3: Distance calculation based on Floyd–Warshall algorithm

```
floyd_warshall
    let dist be an |n| × |n| array of minimum distances initialized to the number of nodes
    for each edge (s, t)
        dist[s][t] ← 1/w_st // w_st is the weight of the edge connecting node s and t
        dist[t][s] ← 1/w_st
    for each node n
        dist[n][n] ← 0
    for k from 1 to |n|
        for i from 1 to |n|
            for j from 1 to |n|
                if dist[i][j] > dist[i][k] + dist[k][j]
                    dist[i][j] ← dist[i][k] + dist[k][j]
    return dist
```

In the original version of 3D KK, $dist[s][t]$ and $dist[t][s]$ for each edge are initialized as 1.

5. Experiments

In the experiments, we compare the performance and efficiency of four CFD algorithms for 3D graph visualization proposed in the paper with the 3D versions of FR and KK algorithms. The descriptions

of the algorithms evaluated in the experiments are listed as follows. Note that all the tested algorithms were implemented for 3D graph visualization. For the purpose of simplification, we will omit the word "3D" from the name of the algorithms in the experiment results.

1. FR algorithm.
2. KK algorithm.
3. CFD-FB which is based on weighted FR with BigCLAM clustering algorithm.
4. CFD-FM which is based on weighted FR with MCL clustering algorithm.
5. CFD-KB which is based on weighted KK with BigCLAM clustering algorithm.
6. CFD-KM which is based on weighted KK with MCL clustering algorithm.

The experiments were conducted on a Windows 10 computer with 8 Gigabytes of RAM and an Intel Core i7-4770 CPU with a base clock speed of 3.40 GHz. Eclipse Neon Release (4.6.0) is adopted as the IDE for the experiments. The gravitational constant is set to 1.0 in the experiments. Note that the initialization of the CFD algorithms includes clustering and random positioning of nodes whereas the initialization of an FDP algorithm only includes random positioning of nodes. No cooling schedule is used in the experiments. Experiment settings for FR, CFD-FB, and CFD-FM are listed in Table 1 and settings for KK, CFD-KB and CFD-KM are listed in Table 2. Each combination given in the settings for each algorithm is executed once in the experiments.

Table 1: Experiment settings for FR, CFD-FB and CFD-FM

Algorithm	Time limit (minutes)			Number of Iterations			Community FDP		Percentage of Iterations for Fine-tuning FR		
FR	10	30	60	100	300	500	-	-	-	-	-
CFD-FB	10	30	60	100	300	500	true	false	100	50	10
CFD-FM	10	30	60	100	300	500	true	false	100	50	10

Table 2: Experiment settings for KK, CFD-KB and CFD-KM

Algorithm	Time limit (minutes)			Threshold			Community FDP		Fine-tuning threshold multiplier		
KK	10	30	60	125	50	20	-	-	-	-	-
CFD-KB	10	30	60	125	50	20	true	false	5.0	3.0	1.0
CFD-KM	10	30	60	125	50	20	true	false	5.0	3.0	1.0

CFD algorithms with different settings may produce layouts of different quality for the same graph.

- "Community FDP" from above tables is a flag used to control step 4 in Figure 2. If "Community FDP" is set to true, step 4 will be executed. Otherwise step 4 is skipped in the entire processing. The main objective of this flag is to evaluate the effect of performing step 4 (generating a layout for each cluster by applying an unweighted FDP to nodes inside each cluster) on the overall performance of CFD algorithms.
- "Fine-tuning FR" column in Table 1 is the percentage of iterations for the fine-tuning step in FR algorithm. For instance, if "Number of Iterations" is set to 300 and "Fine-tuning FR" is set to 50%, fine-tuning step in FR will be executed for 150 iterations. Fine-tuning FR is only applicable to CFD-FB and CFD-FM.

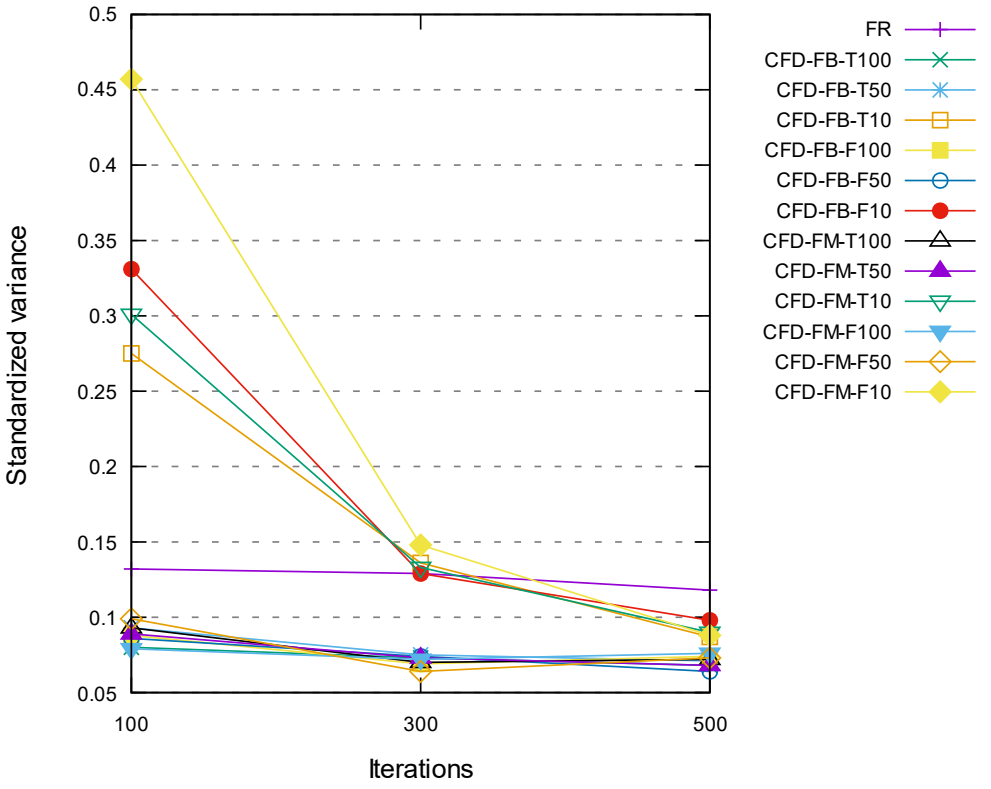
- "Fine-tuning threshold multiplier" in Table 2 is the multiplier of threshold of fine-tuning KK. For instance, if "Threshold" is set to 50 and "Fine-tuning threshold multiplier" is set to 3.0, the threshold that is used in KK when CFD is in fine-tuning step will be 150. Fine-tuning threshold multiplier is only applicable to CFD-KB and CFD-KM.

5.1 Experiment results

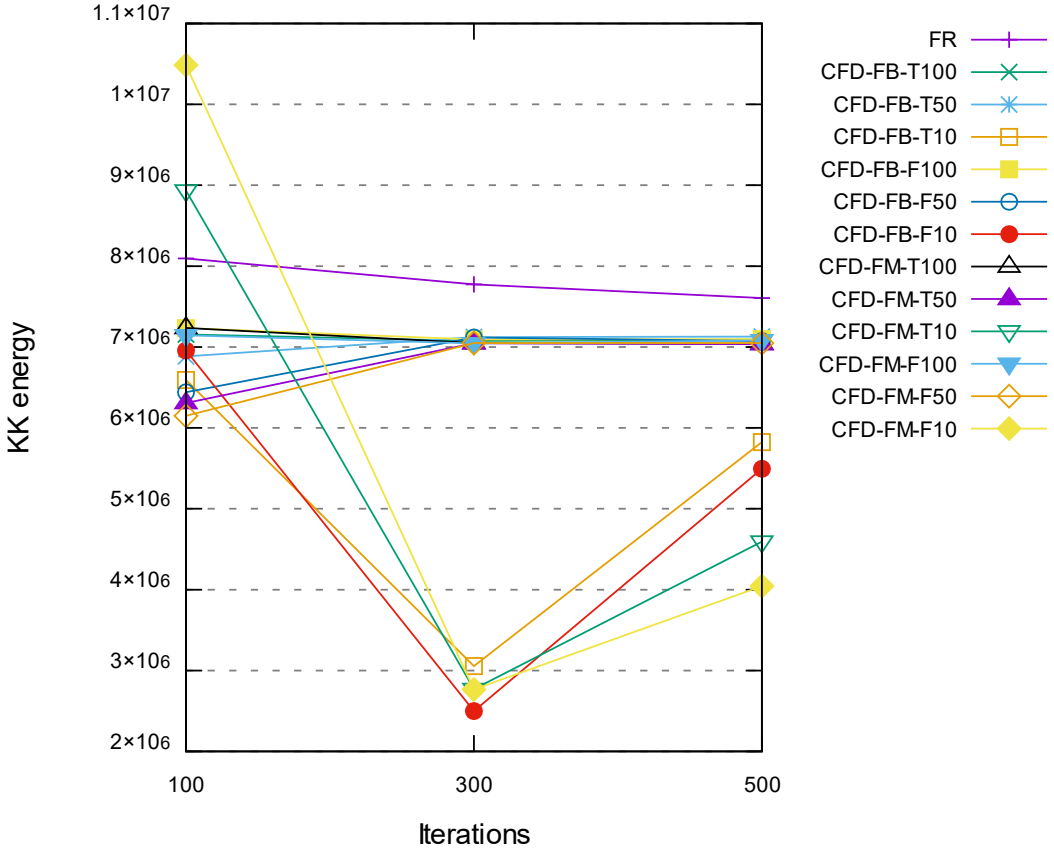
5.1.1 Layout quality comparison

In this section, a medium dataset ("PhD students in computer science"), a small dataset ("dolphins") and a large dataset ("Power grid") are used for evaluation.

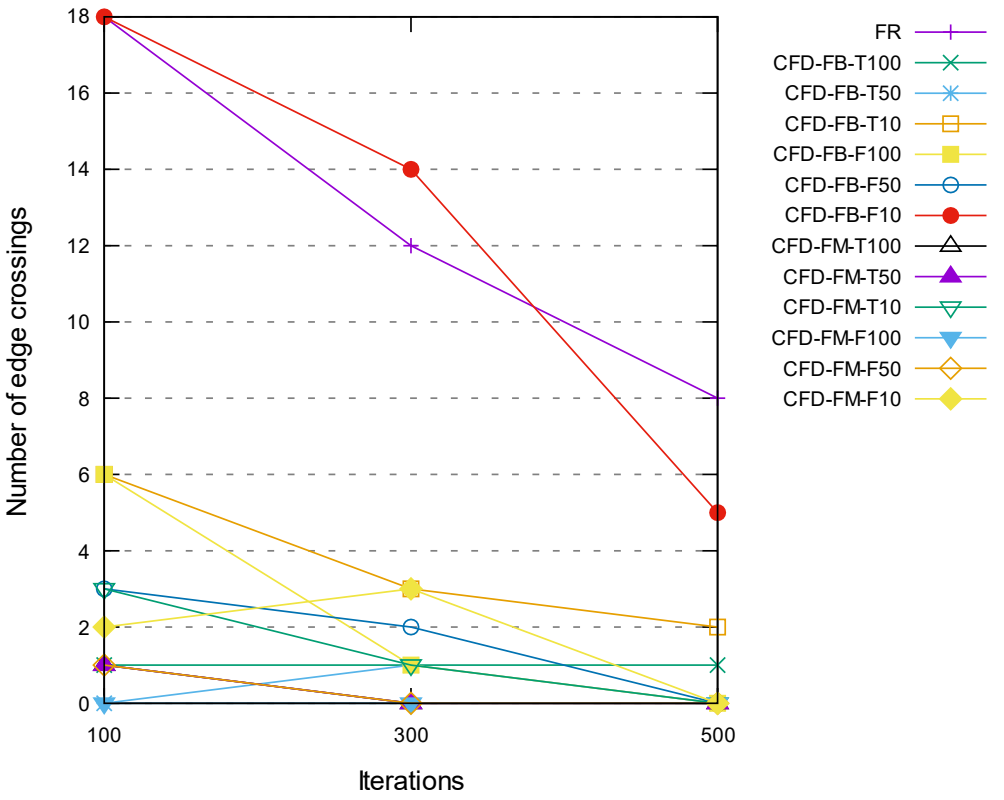
Dataset: PhD students in computer science (1025 nodes, 1043 edges) [34] : First, we compare the results of using FR alone and using CFD-FB and CFD-FM for 10 minutes execution time in Figure 3. T and F from the last field of the legends are used to indicate the application of a FDP in a specific test case. For example, "CFD-FB-T100" from Figure 3 denotes CFD-FB algorithm with FR applied to each cluster and 100% of iterations for fine-tuning FR. "CFD-FB-F50" denotes CFD-FB with no FR applied to any cluster and 50% of iterations for fine-tuning FR. The standardized variance, the KK energy and the number of edge crossings in Figure 3 show that the overall quality of the layouts was improved when using CFD-FB and CFD-FM with appropriate configuration (such as CFD-FB-T100 and CFD-FM-T100) compared with FR. Among them, the number of edge crossings in output layouts was reduced significantly (see Figure 3(c)). The total time measurement in Figure 3(d) shows that CFD-FM is not an efficient algorithm. CFD-FB is superior to other algorithms since it only incurs slightly more computational cost compared with FR alone.


Standardized variance
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
100
300
500
Iterations
FR
CFD-FB-T100
CFD-FB-T50
CFD-FB-T10
CFD-FB-F100
CFD-FB-F50
CFD-FB-F10
CFD-FM-T100
CFD-FM-T50
CFD-FM-T10
CFD-FM-F100
CFD-FM-F50
CFD-FM-F10

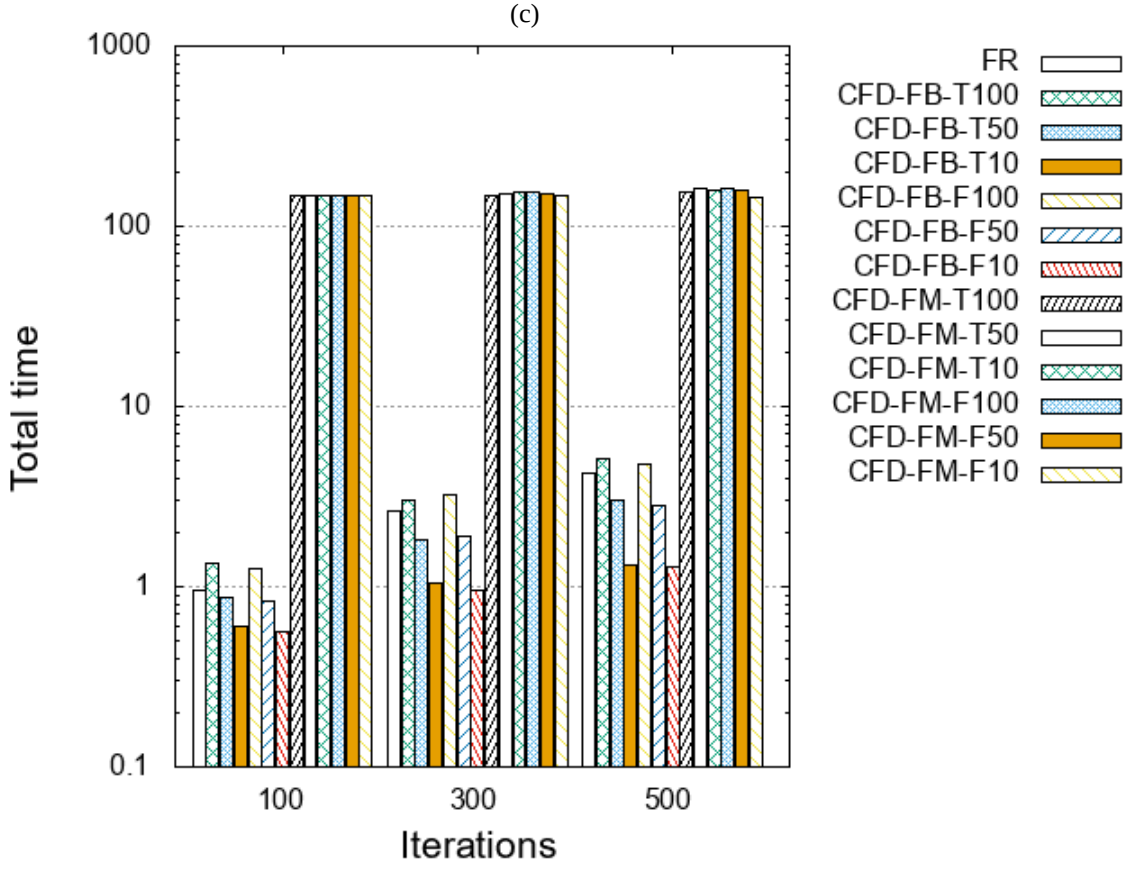
(a)


KK energy
1.1×10⁷
1×10⁷
9×10⁶
8×10⁶
7×10⁶
6×10⁶
5×10⁶
4×10⁶
3×10⁶
2×10⁶
100
300
500
Iterations
FR
CFD-FB-T100
CFD-FB-T50
CFD-FB-T10
CFD-FB-F100
CFD-FB-F50
CFD-FB-F10
CFD-FM-T100
CFD-FM-T50
CFD-FM-T10
CFD-FM-F100
CFD-FM-F50
CFD-FM-F10

(b)



(c)



(d)

Figure 3: Comparison of FR, CFD-FB and CFD-FM with a 10-minute time limit. (a) standardized variance (b) KK energy (c) edge crossings (d) total time usage.

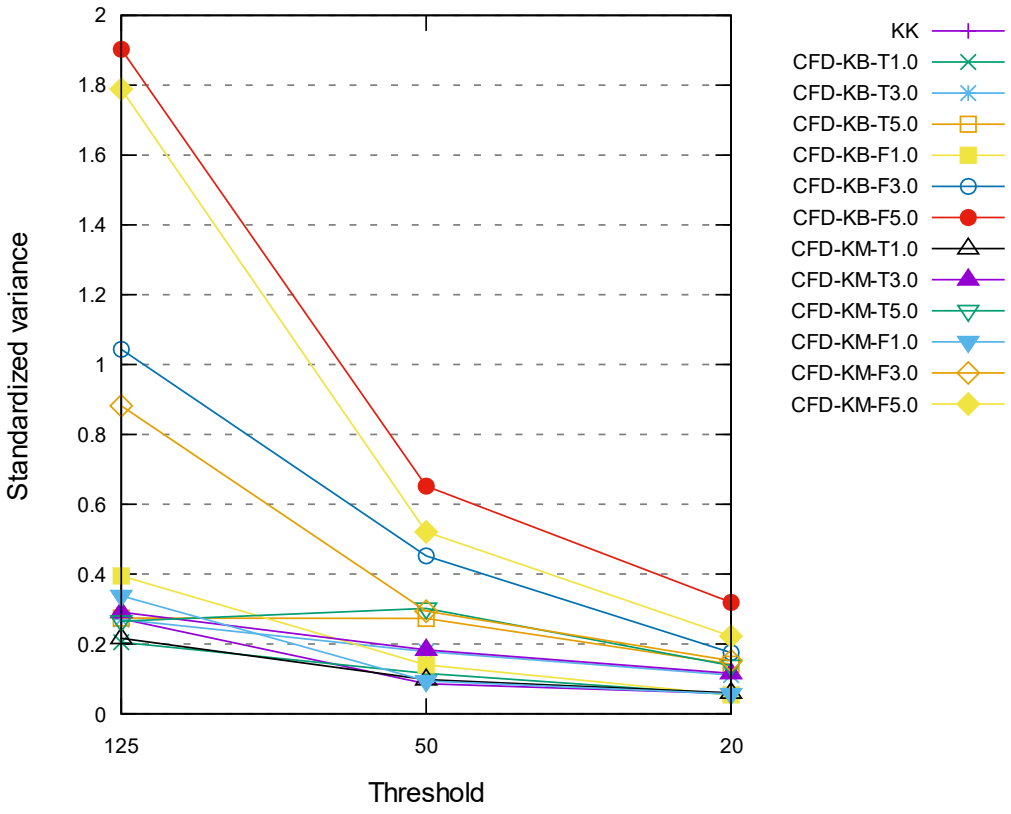
We perform similar experiments for KK, CFD-KB and CFD-KM for 10 minutes execution time and the experiment results are depicted in Figure 4. T and F from the last field of the legends are used to indicate the application of a FDP in a specific test case. For example, "CFD-KB-T1.0" stands for using CFD-KB with KK applied to each cluster and threshold multiplier is set to 1.0 for fine-tuning KK. "CFD-KB-F3.0" stands for using CFD-FB with no KK applied to any of the clusters and threshold multiplier is set to 3.0 for fine-tuning KK.

Figure 3(a) and Figure 4(a) also show that there is also a slight improvement on standardized variance on lower thresholds for KK. The significant improvement of KK energy in Figure 3(b) and Figure 4(b) is expected since KK algorithm utilizes the energy function to compute the displacement of nodes that can reduce the KK energy of a layout. However, Figure 3(d) and Figure 4(d) reveal that FR can effectively reduce the number of edge crossings than KK.

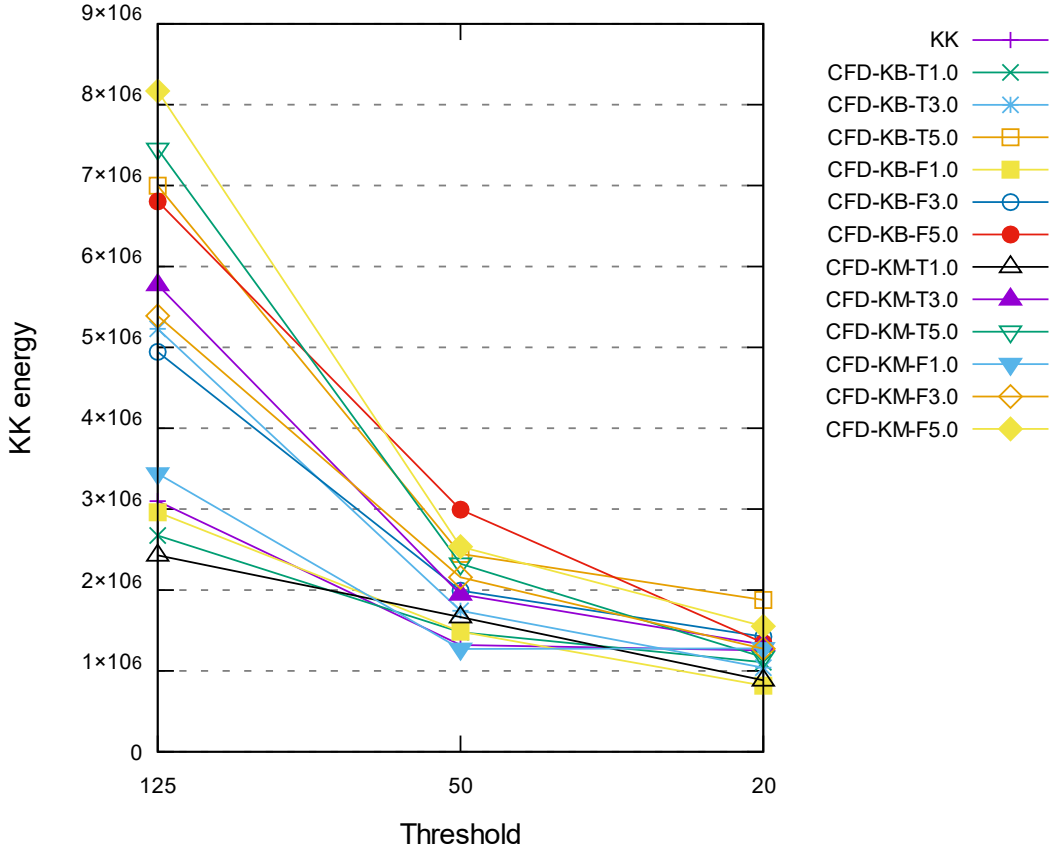
The overall quality of layouts generated by KK is similar to CFD-KB and CFD-KM in certain configuration settings such as CFD-KB-T1.0 and CFD-KM-T1.0. Figure 4(a) and (b) show that the differences in the values of the measurements for KK, CFD-KB-T1.0 and CFD-KM-T1.0 are less than 42%. The largest difference in KK energy can be observed in KK and CFD-KM-T1.0 when threshold value is set to 20.

The only significant difference among KK, CFD-KB and CFD-KM was the computational time. By using CFD-KB, the computational time was reduced. Even so, it is still too computationally expensive compared to CFD-FB. Thus, KK is not an efficient algorithm. Besides, it is possible that KK algorithm can be trapped when the algorithm is trying to reduce the Δ_m of a node m so that it is less than the threshold ε (see section 4.3.2). We can observe that CFD-FB produced the best overall result.

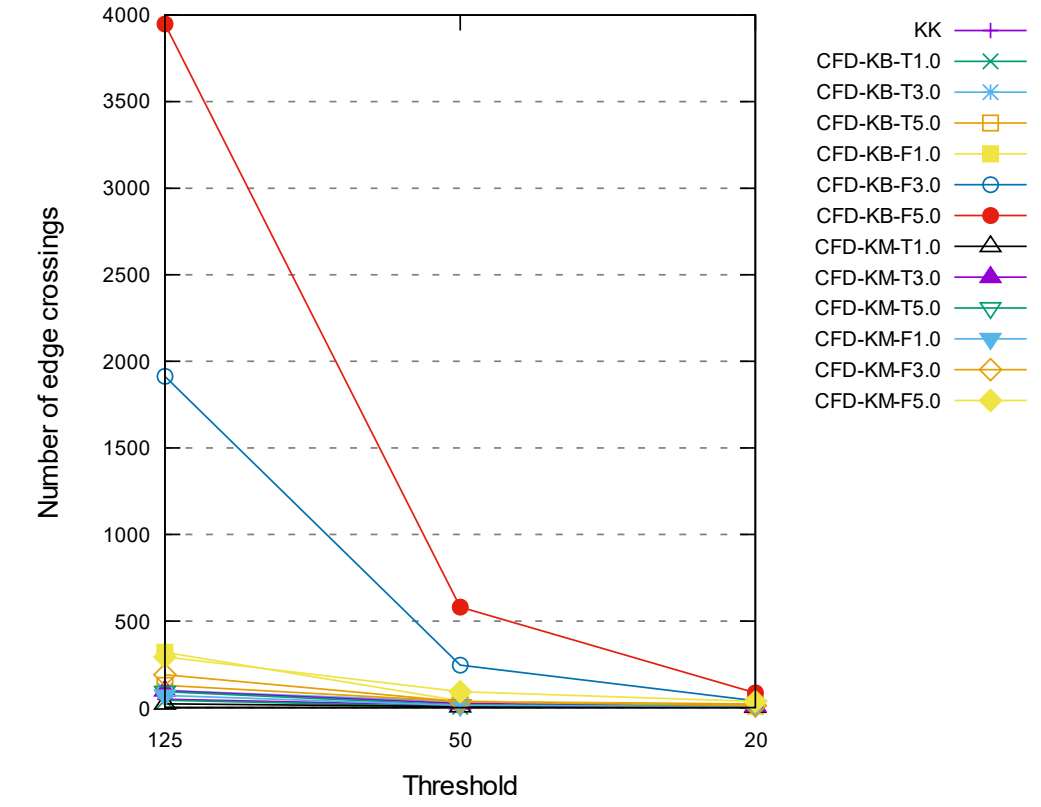
Experiment results for 30 minutes time limit are depicted in Figure 5 and Figure 6. Experiment results for 60 minutes time limit are shown in Figure 7 and Figure 8. These results are similar to the results from 10-minutes time limit.


Standardized variance
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
125
50
20
Threshold
KK
CFD-KB-T1.0
CFD-KB-T3.0
CFD-KB-T5.0
CFD-KB-F1.0
CFD-KB-F3.0
CFD-KB-F5.0
CFD-KM-T1.0
CFD-KM-T3.0
CFD-KM-T5.0
CFD-KM-F1.0
CFD-KM-F3.0
CFD-KM-F5.0

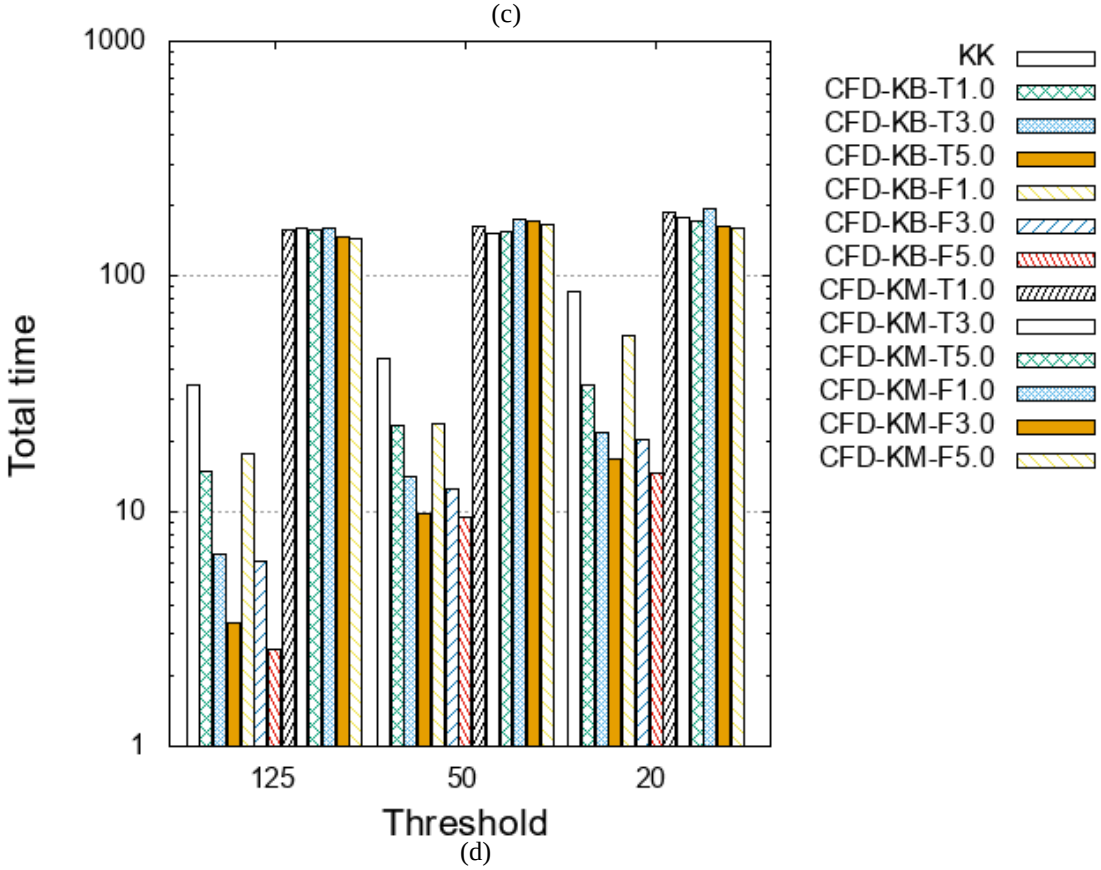
(a)


KK energy
9×10⁶
8×10⁶
7×10⁶
6×10⁶
5×10⁶
4×10⁶
3×10⁶
2×10⁶
1×10⁶
0
125
50
20
Threshold
KK
CFD-KB-T1.0
CFD-KB-T3.0
CFD-KB-T5.0
CFD-KB-F1.0
CFD-KB-F3.0
CFD-KB-F5.0
CFD-KM-T1.0
CFD-KM-T3.0
CFD-KM-T5.0
CFD-KM-F1.0
CFD-KM-F3.0
CFD-KM-F5.0

(b)

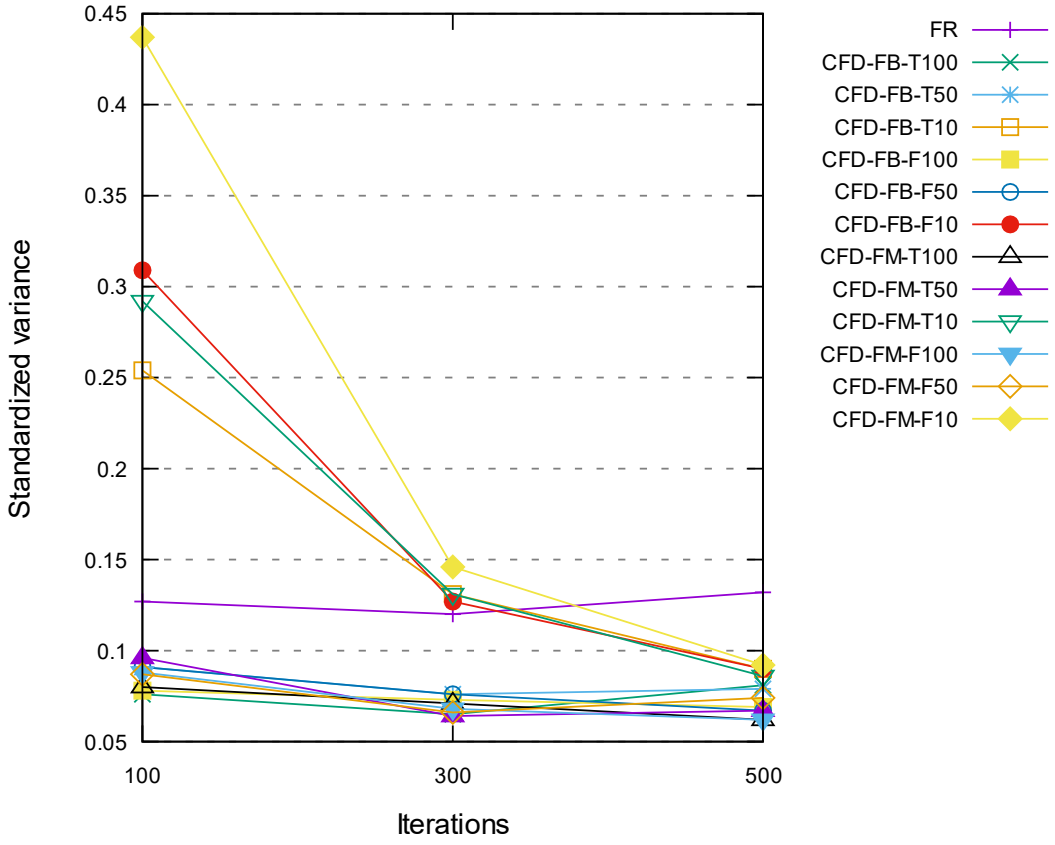


(c)

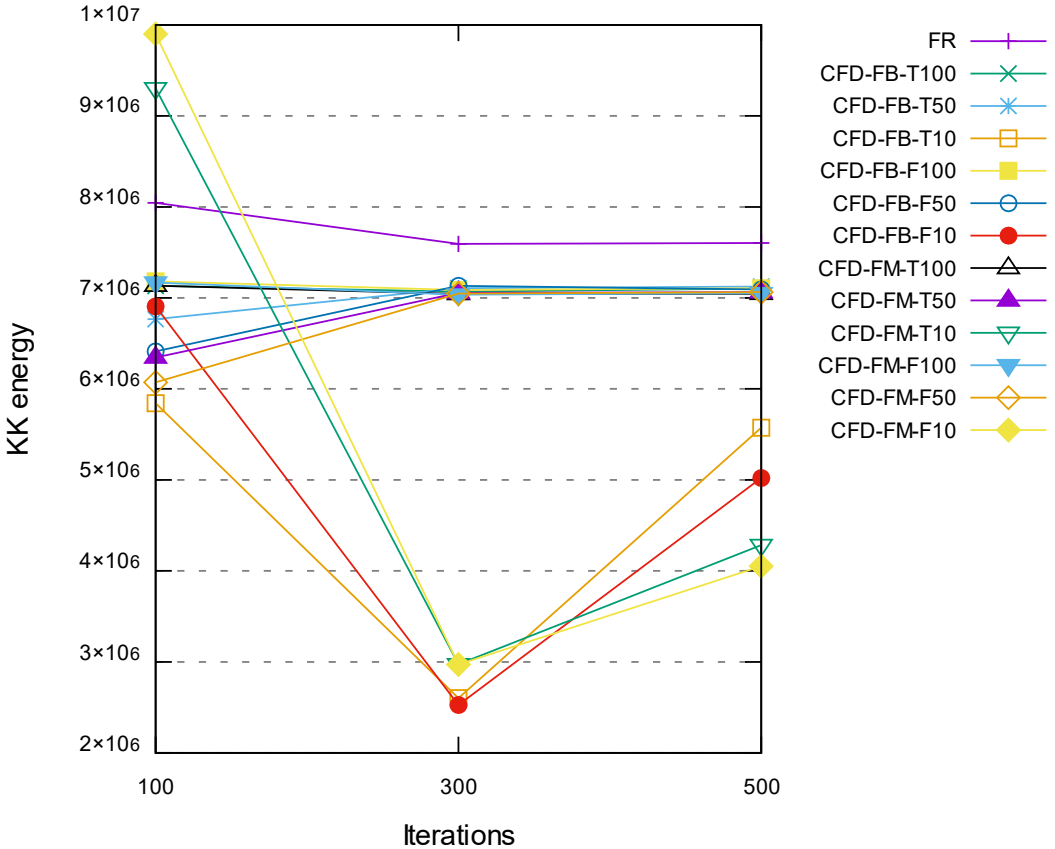


(d)

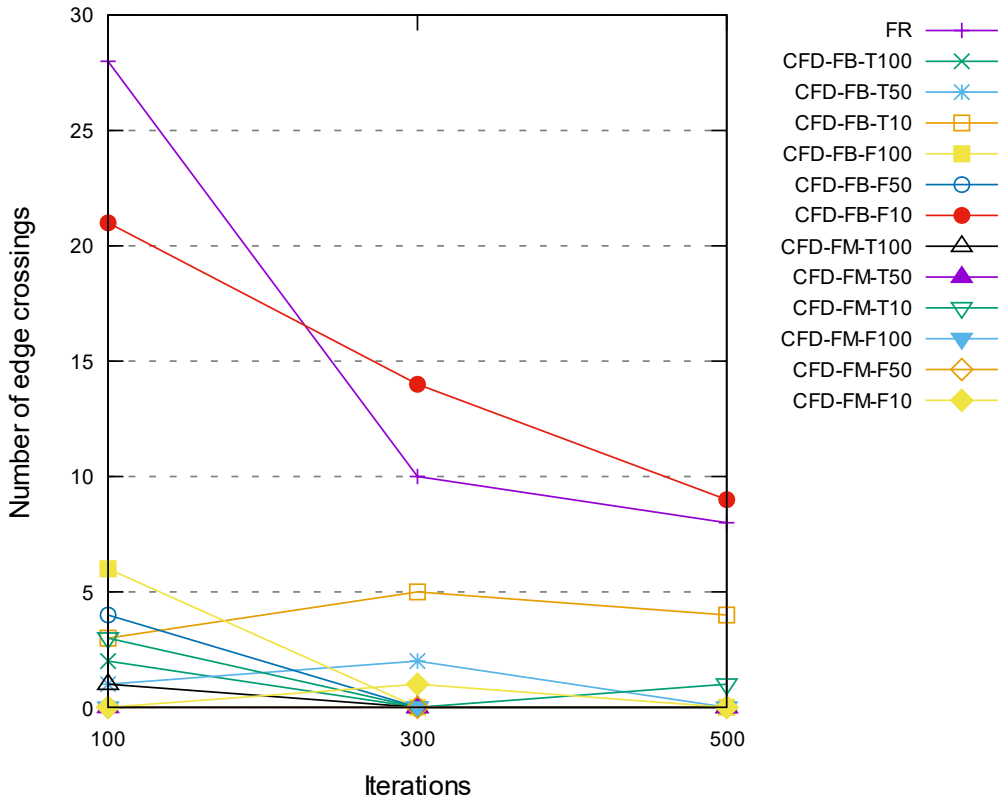
Figure 4: Comparison of KK, CFD-KB and CFD-KM with 10-minute time limit. (a) standardized variance (b) KK energy (c) edge crossings (d) total time usage.


Standardized variance
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
100
300
500
Iterations
FR
CFD-FB-T100
CFD-FB-T50
CFD-FB-T10
CFD-FB-F100
CFD-FB-F50
CFD-FB-F10
CFD-FM-T100
CFD-FM-T50
CFD-FM-T10
CFD-FM-F100
CFD-FM-F50
CFD-FM-F10

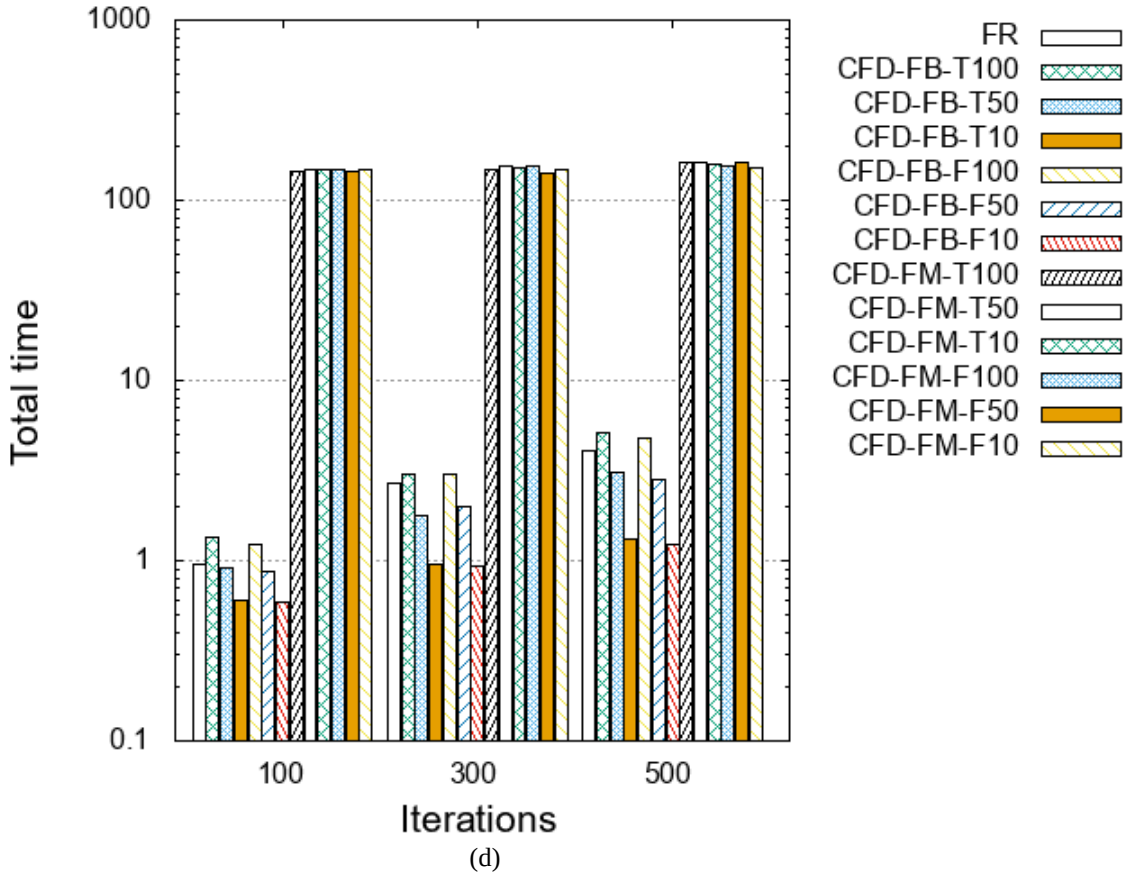
(a)


KK energy
1×10⁷
9×10⁶
8×10⁶
7×10⁶
6×10⁶
5×10⁶
4×10⁶
3×10⁶
2×10⁶
100
300
500
Iterations
FR
CFD-FB-T100
CFD-FB-T50
CFD-FB-T10
CFD-FB-F100
CFD-FB-F50
CFD-FB-F10
CFD-FM-T100
CFD-FM-T50
CFD-FM-T10
CFD-FM-F100
CFD-FM-F50
CFD-FM-F10

(b)

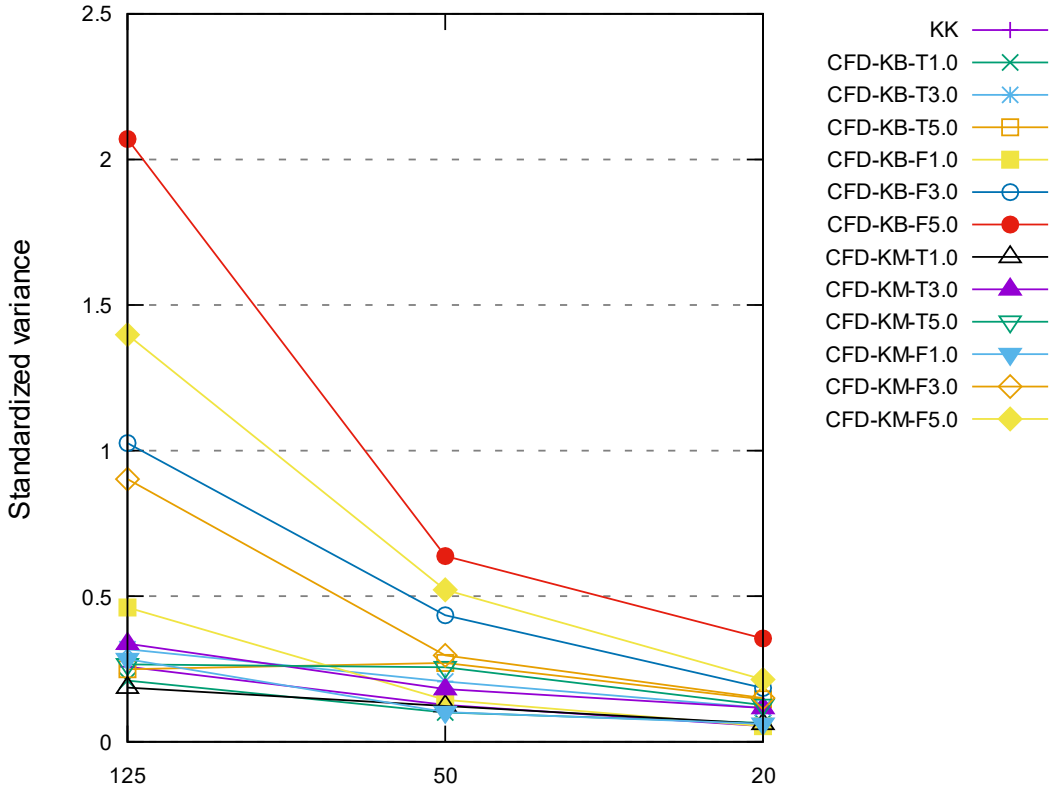


(c)

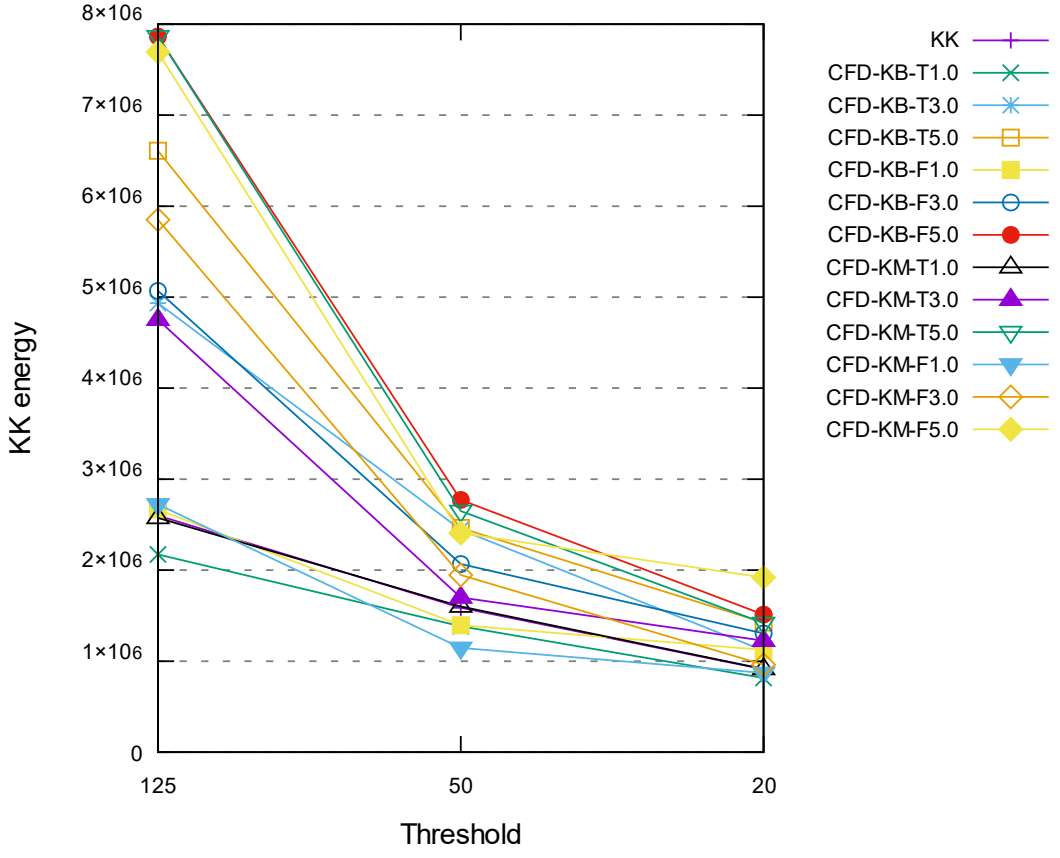


(d)

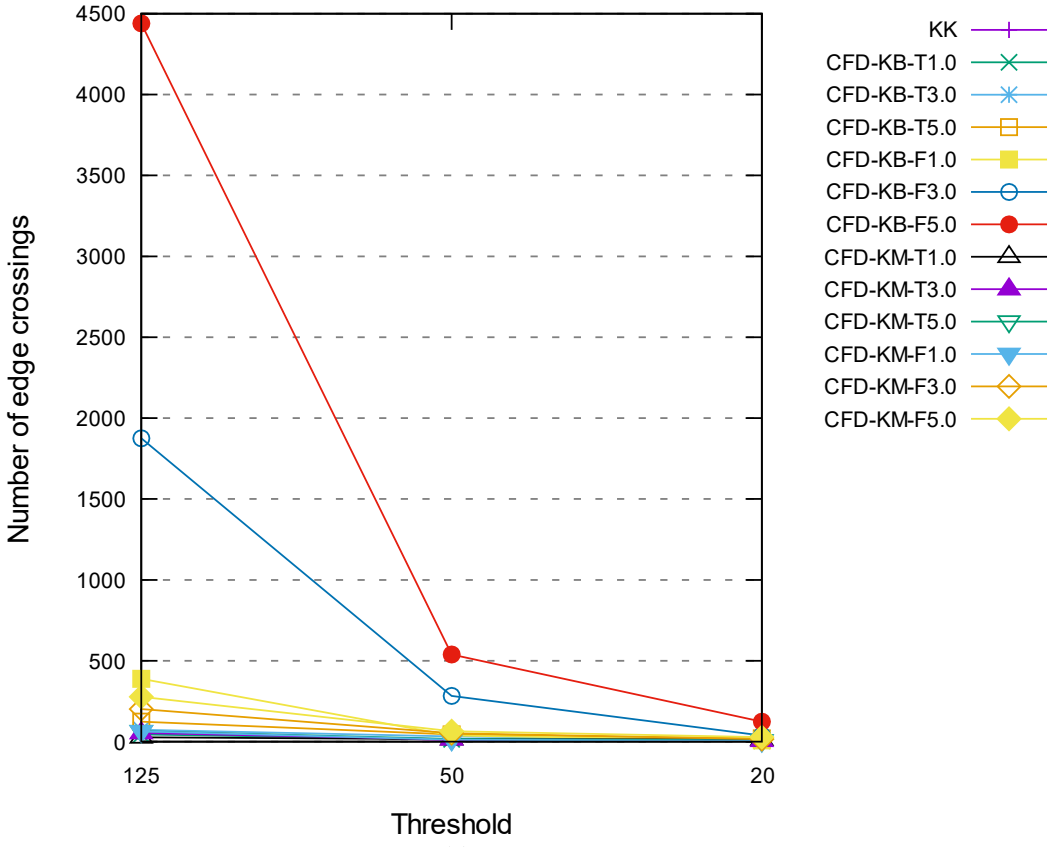
Figure 5: Comparison of FR, CFD-FB and CFD-FM with 30-minute time limit. (a) standardized variance (b) KK energy (c) edge crossings (d) total time usage.


Standardized variance
2.5
2
1.5
1
0.5
0
125
50
20
Threshold
KK
CFD-KB-T1.0
CFD-KB-T3.0
CFD-KB-T5.0
CFD-KB-F1.0
CFD-KB-F3.0
CFD-KB-F5.0
CFD-KM-T1.0
CFD-KM-T3.0
CFD-KM-T5.0
CFD-KM-F1.0
CFD-KM-F3.0
CFD-KM-F5.0

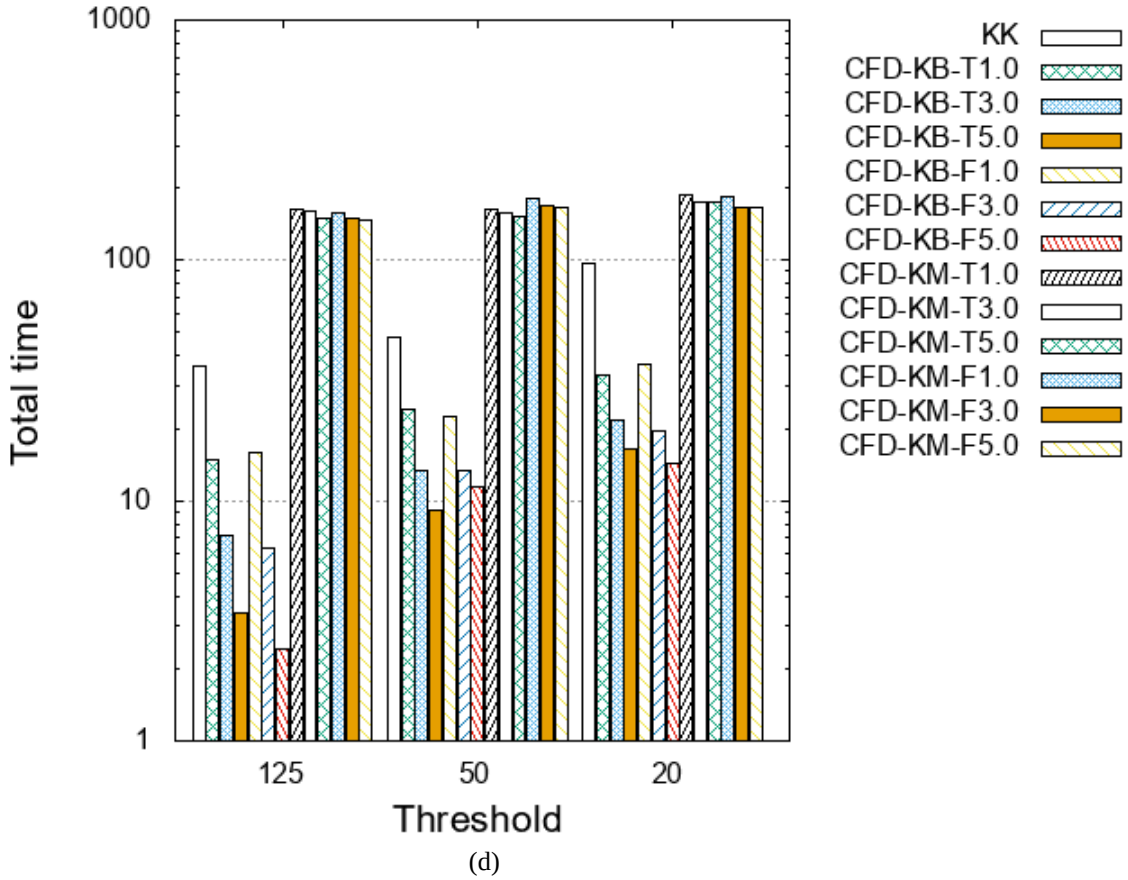
(a)


KK energy
8×10⁶
7×10⁶
6×10⁶
5×10⁶
4×10⁶
3×10⁶
2×10⁶
1×10⁶
0
125
50
20
Threshold
KK
CFD-KB-T1.0
CFD-KB-T3.0
CFD-KB-T5.0
CFD-KB-F1.0
CFD-KB-F3.0
CFD-KB-F5.0
CFD-KM-T1.0
CFD-KM-T3.0
CFD-KM-T5.0
CFD-KM-F1.0
CFD-KM-F3.0
CFD-KM-F5.0

(b)

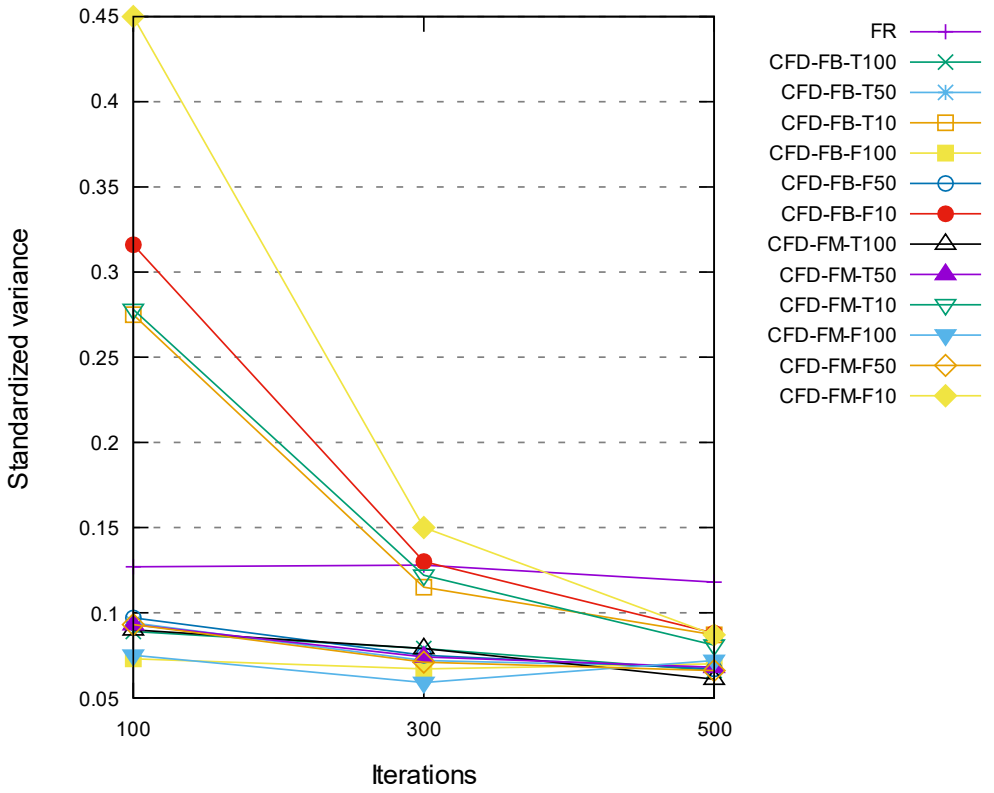


(c)

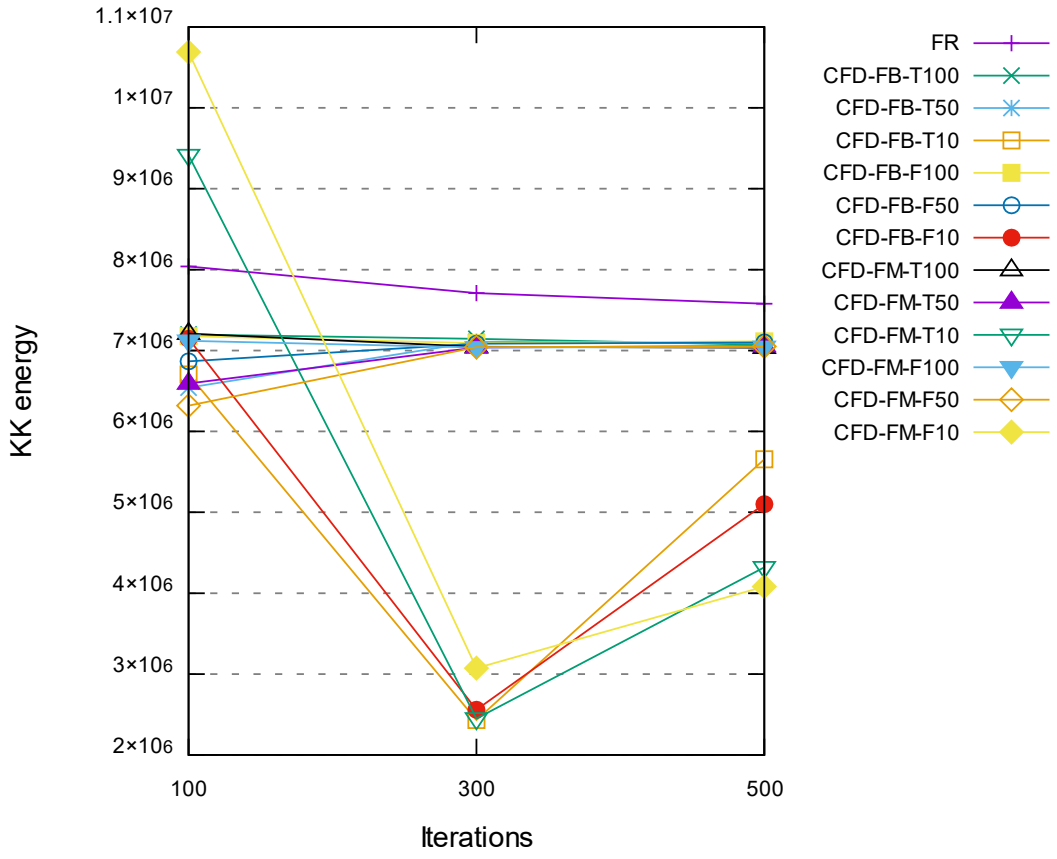


(d)

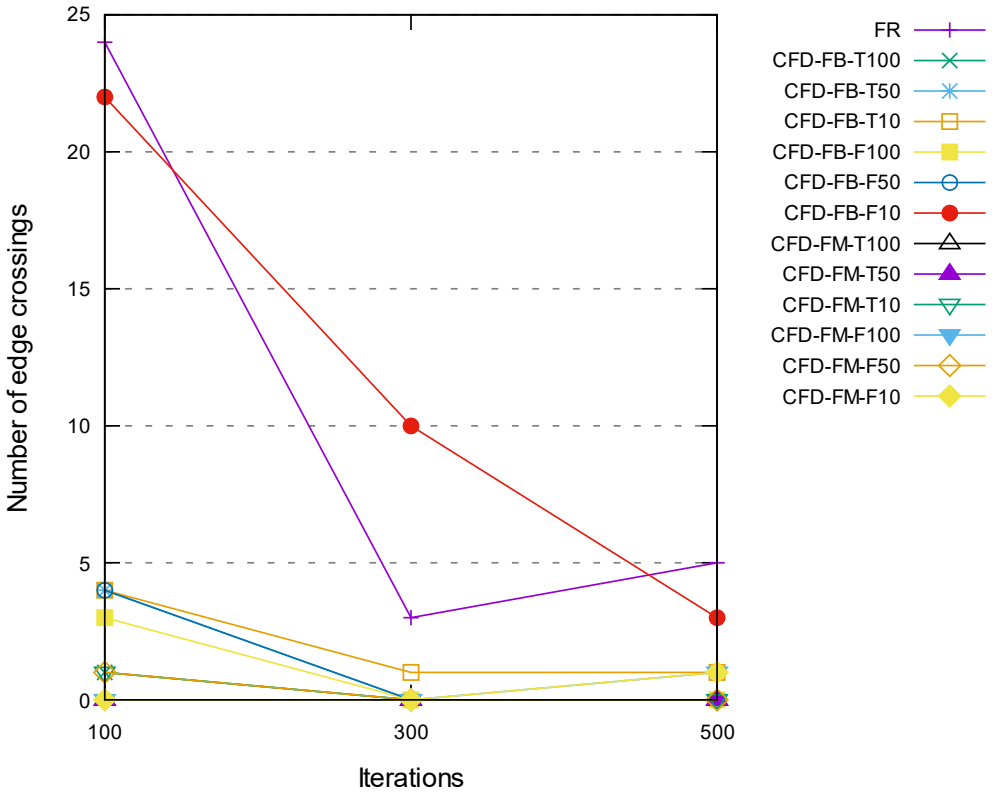
Figure 6: Comparison of KK, CFD-KB and CFD-KM with 30-minute time limit. (a) standardized variance (b) KK energy (c) edge crossings (d) total time usage.


Standardized variance
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
100
300
500
Iterations
FR
CFD-FB-T100
CFD-FB-T50
CFD-FB-T10
CFD-FB-F100
CFD-FB-F50
CFD-FB-F10
CFD-FM-T100
CFD-FM-T50
CFD-FM-T10
CFD-FM-F100
CFD-FM-F50
CFD-FM-F10

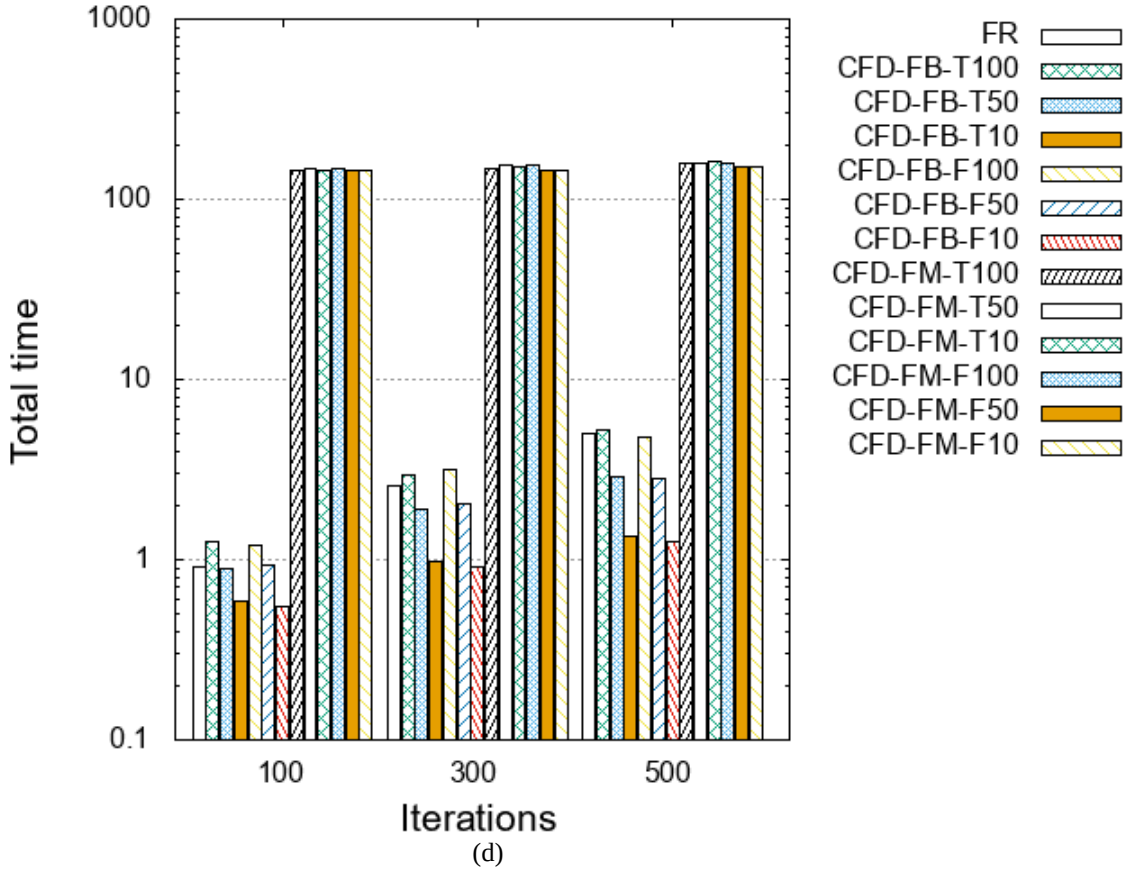
(a)


KK energy
1.1×10⁷
1×10⁷
9×10⁶
8×10⁶
7×10⁶
6×10⁶
5×10⁶
4×10⁶
3×10⁶
2×10⁶
100
300
500
Iterations
FR
CFD-FB-T100
CFD-FB-T50
CFD-FB-T10
CFD-FB-F100
CFD-FB-F50
CFD-FB-F10
CFD-FM-T100
CFD-FM-T50
CFD-FM-T10
CFD-FM-F100
CFD-FM-F50
CFD-FM-F10

(b)

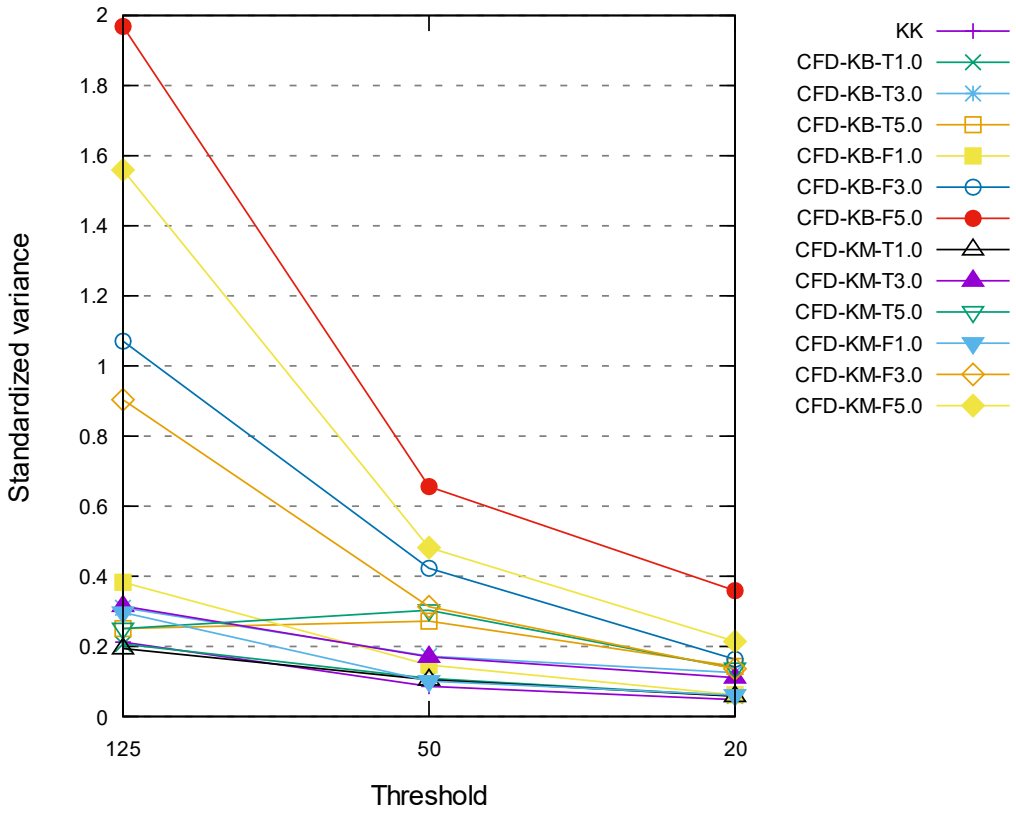


(c)

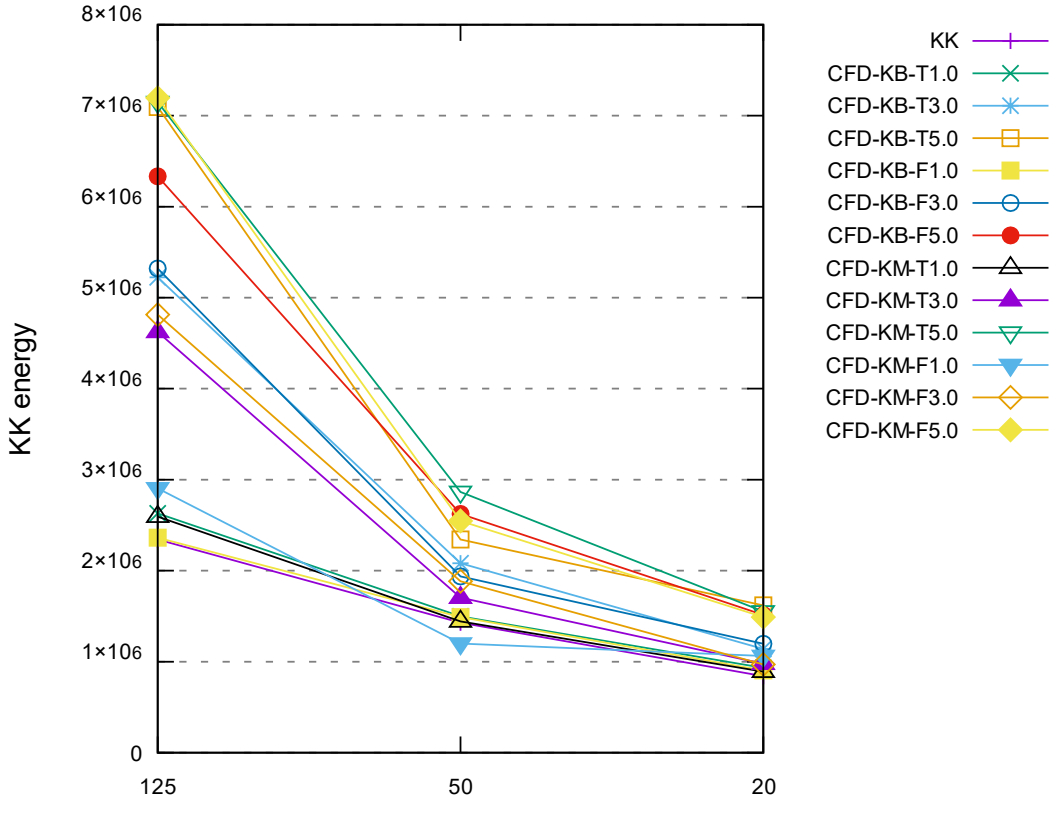


(d)

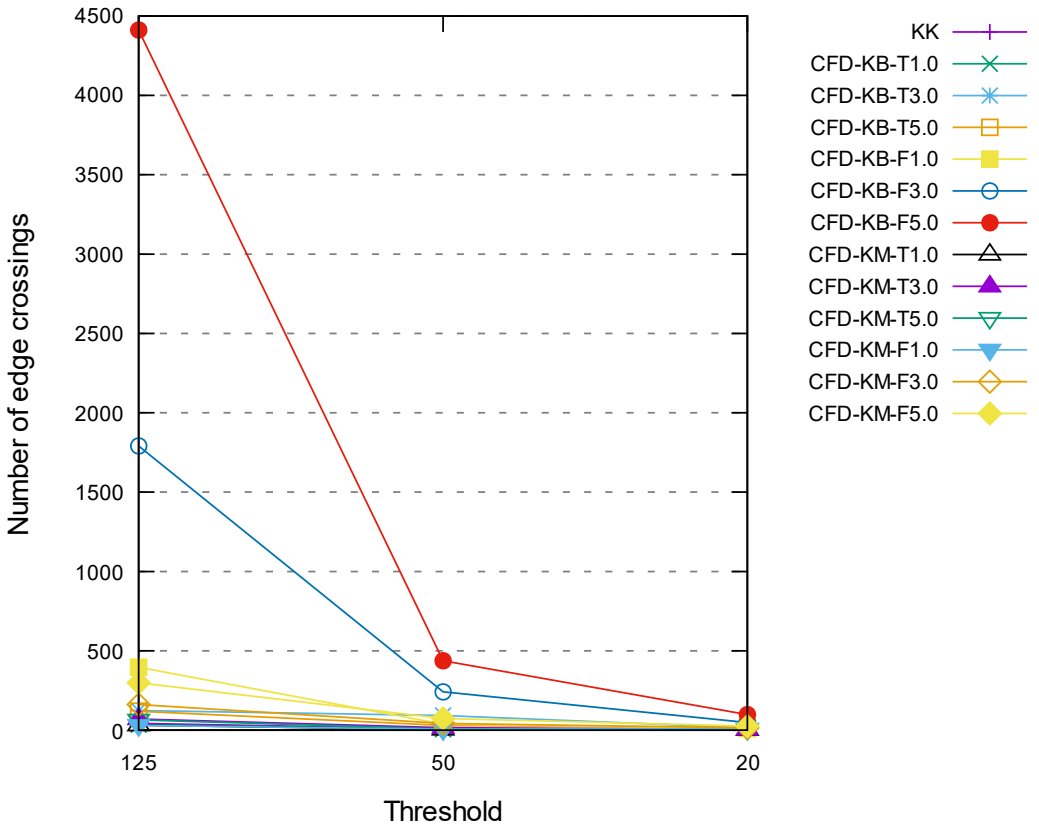
Figure 7: Comparison of FR, CFD-FB and CFD-FM with 60-minute time limit. (a) standardized variance (b) KK energy (c) edge crossings (d) total time usage.


Standardized variance
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
125
50
20
Threshold
KK
CFD-KB-T1.0
CFD-KB-T3.0
CFD-KB-T5.0
CFD-KB-F1.0
CFD-KB-F3.0
CFD-KB-F5.0
CFD-KM-T1.0
CFD-KM-T3.0
CFD-KM-T5.0
CFD-KM-F1.0
CFD-KM-F3.0
CFD-KM-F5.0

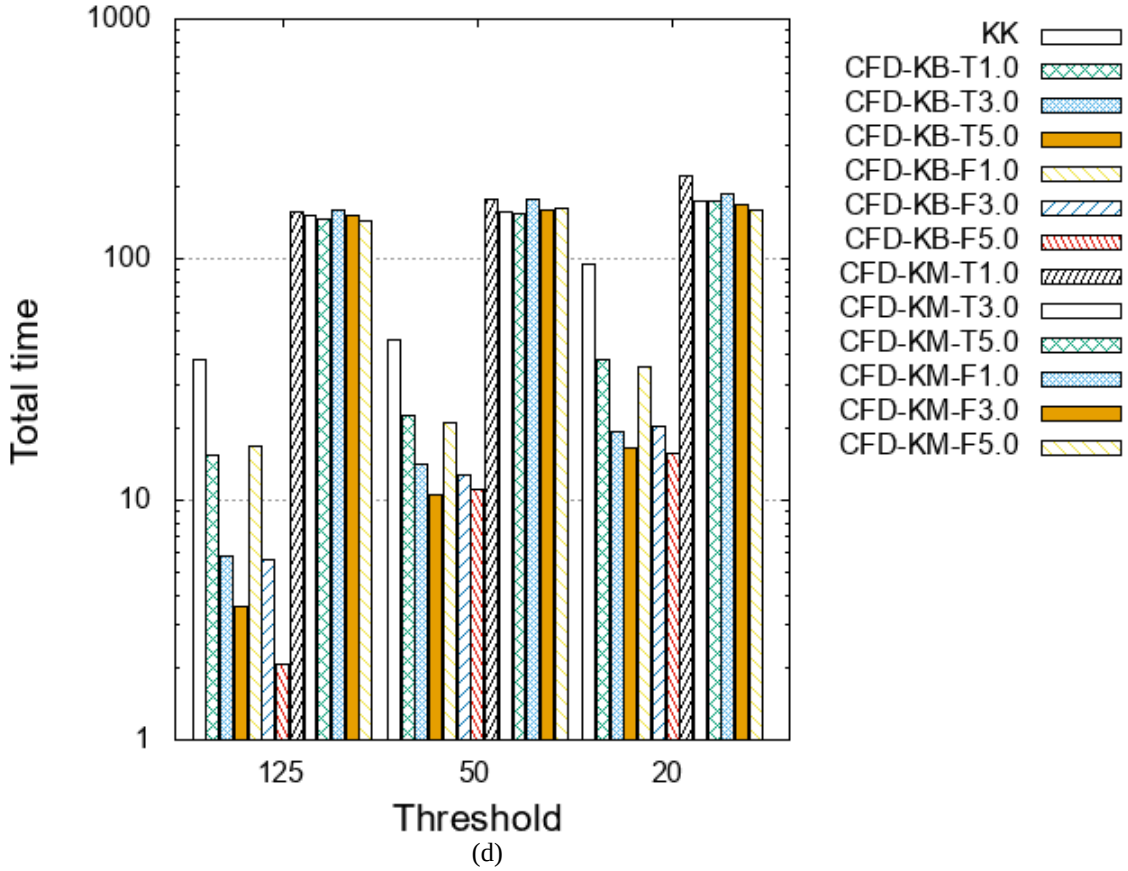
(a)


KK energy
8×10⁶
7×10⁶
6×10⁶
5×10⁶
4×10⁶
3×10⁶
2×10⁶
1×10⁶
0
125
50
20
Threshold
KK
CFD-KB-T1.0
CFD-KB-T3.0
CFD-KB-T5.0
CFD-KB-F1.0
CFD-KB-F3.0
CFD-KB-F5.0
CFD-KM-T1.0
CFD-KM-T3.0
CFD-KM-T5.0
CFD-KM-F1.0
CFD-KM-F3.0
CFD-KM-F5.0

(b)



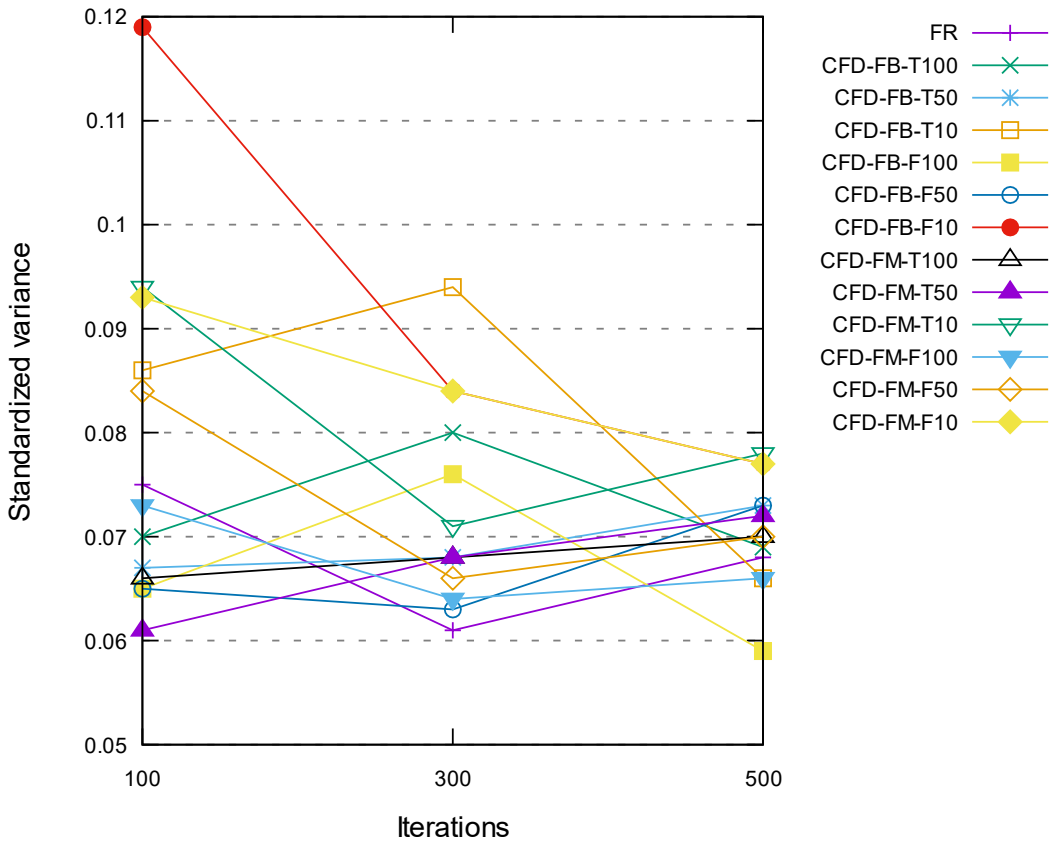
(c)



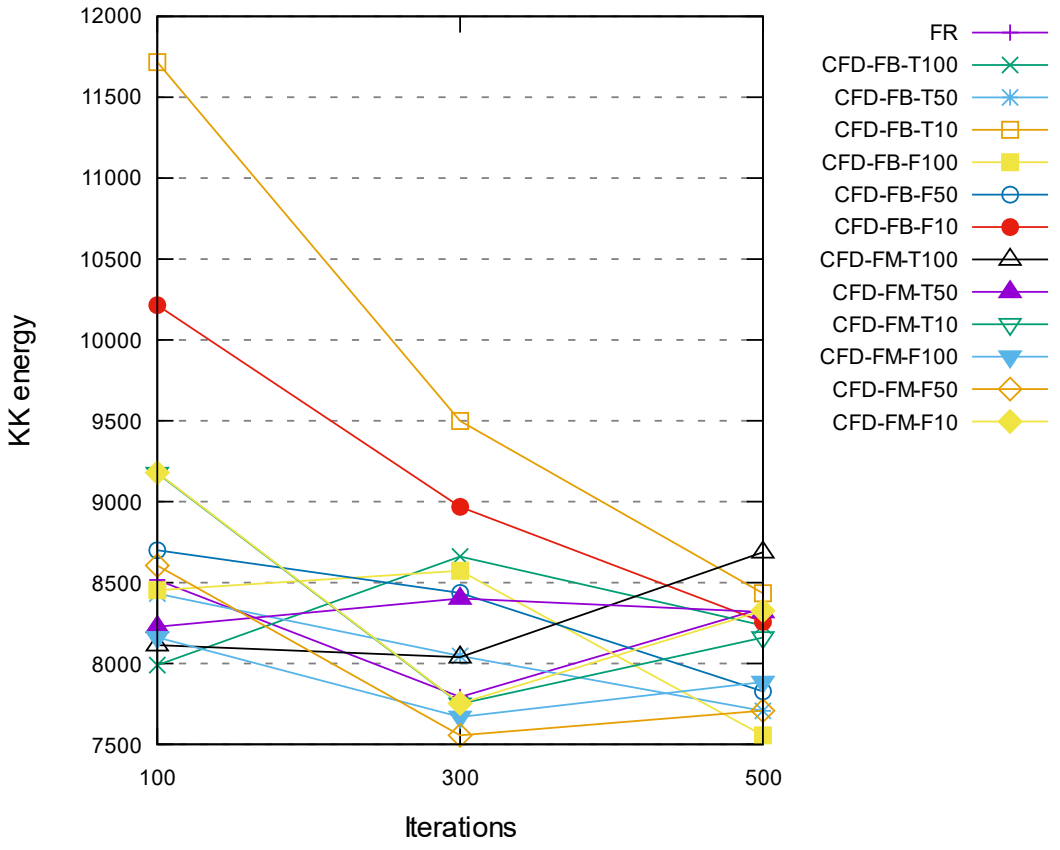
(d)

Figure 8: Comparison of KK, CFD-KB and CFD-KM with 60-minute time limit. (a) standardized variance (b) KK energy (c) edge crossings (d) total time usage.

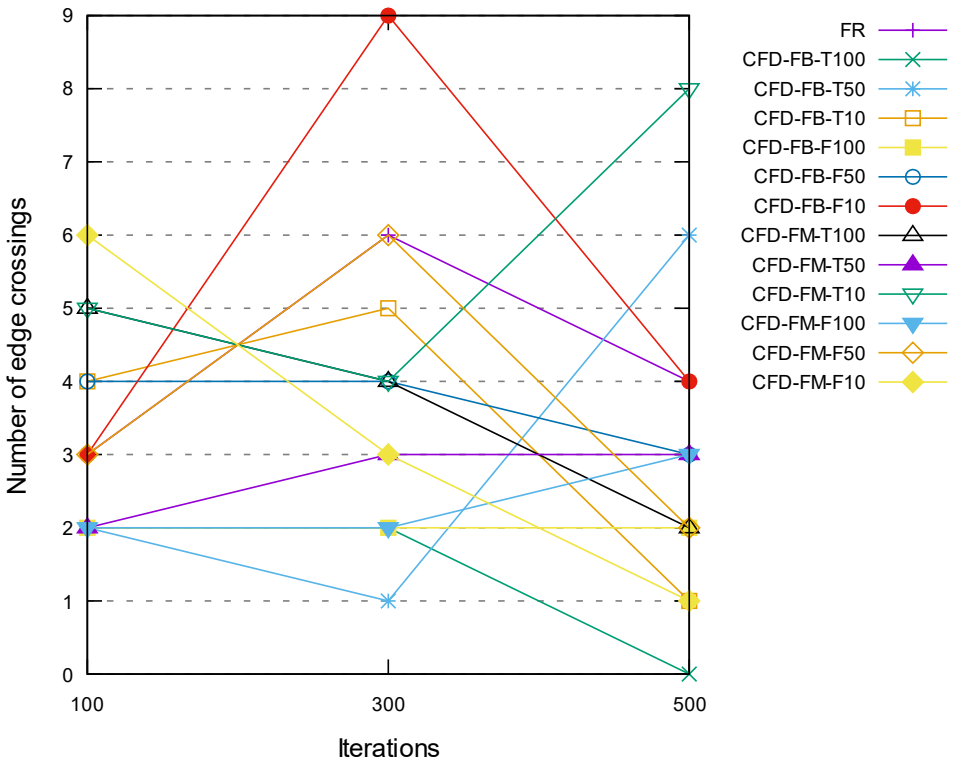
Dataset: dolphins (62 nodes, 159 edges) [35]: First, we compare the results of FR, CFD-FB and CFD-FM for 10 minutes execution time. Figure 9(a), (b) and (c) show that there is no significant difference among these algorithms when the input topology is a small graph. Specifically, Figure 9(a) and (b) show that the difference in results is less than 50% for all algorithms. Figure 9(c) shows that the number of edge crossings is less than 10 for all algorithms. Figure 9(d) shows that BigCLAM requires slightly more time compared to MCL. Experiment results of KK, CFD-KB and CFD-KM for 10 minutes execution time are given Figure 10. The results of the experiments for KK, CFD-KB and CFD-KM are also similar to the case of FR, CFD-FB and CFD-FM.


Standardized variance
0.12
0.11
0.1
0.09
0.08
0.07
0.06
0.05
100
300
500
Iterations
FR
CFD-FB-T100
CFD-FB-T50
CFD-FB-T10
CFD-FB-F100
CFD-FB-F50
CFD-FB-F10
CFD-FM-T100
CFD-FM-T50
CFD-FM-T10
CFD-FM-F100
CFD-FM-F50
CFD-FM-F10

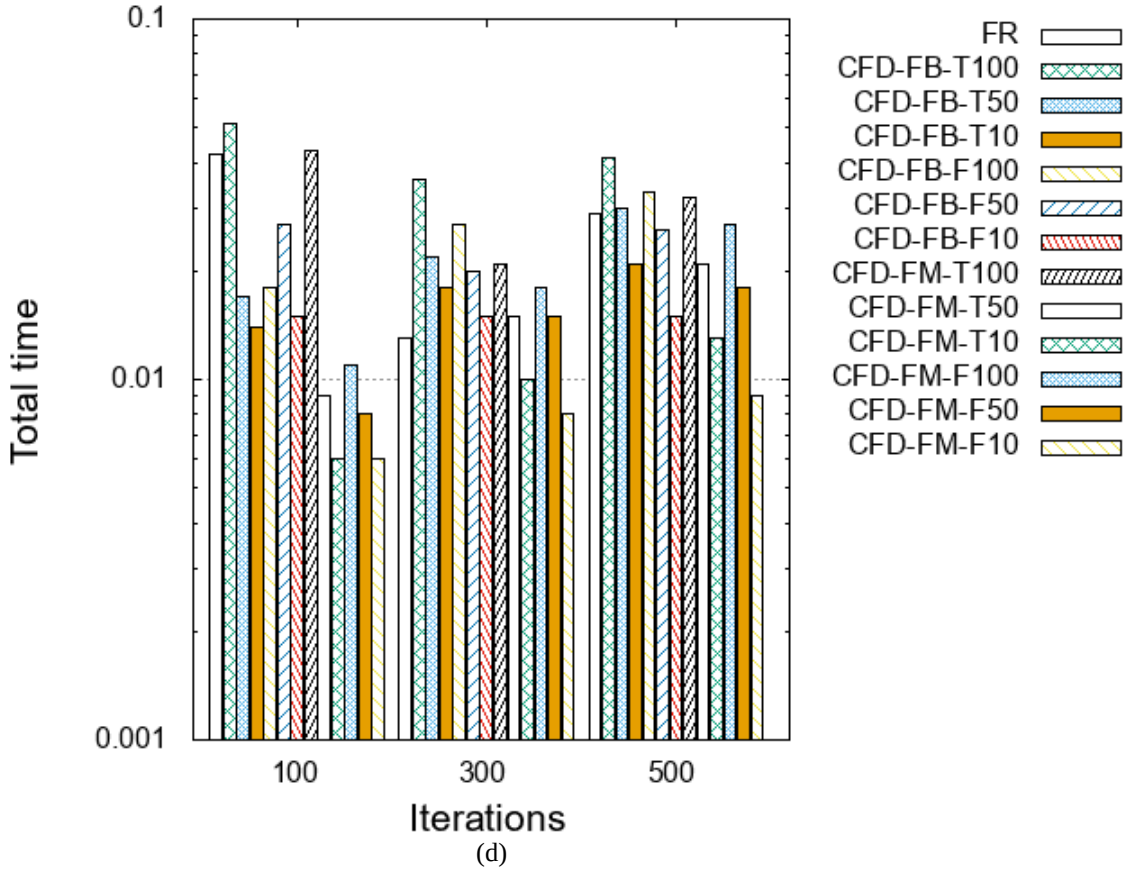
(a)


KK energy
12000
11500
11000
10500
10000
9500
9000
8500
8000
7500
100
300
500
Iterations
FR
CFD-FB-T100
CFD-FB-T50
CFD-FB-T10
CFD-FB-F100
CFD-FB-F50
CFD-FB-F10
CFD-FM-T100
CFD-FM-T50
CFD-FM-T10
CFD-FM-F100
CFD-FM-F50
CFD-FM-F10

(b)

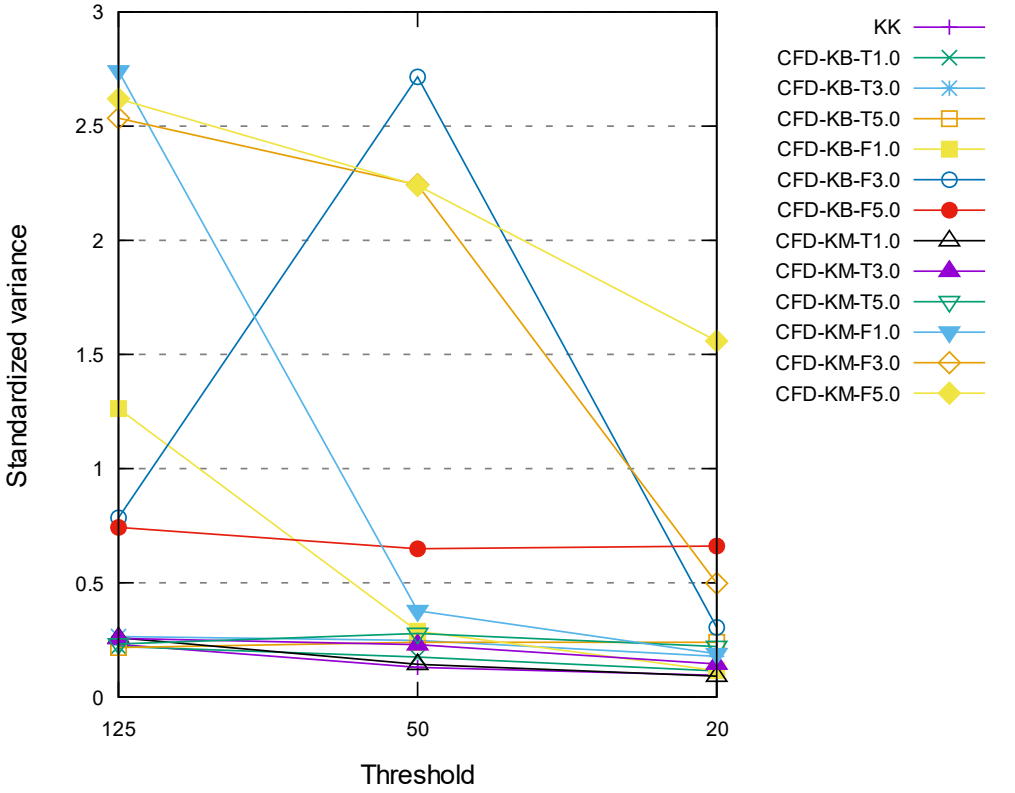


(c)

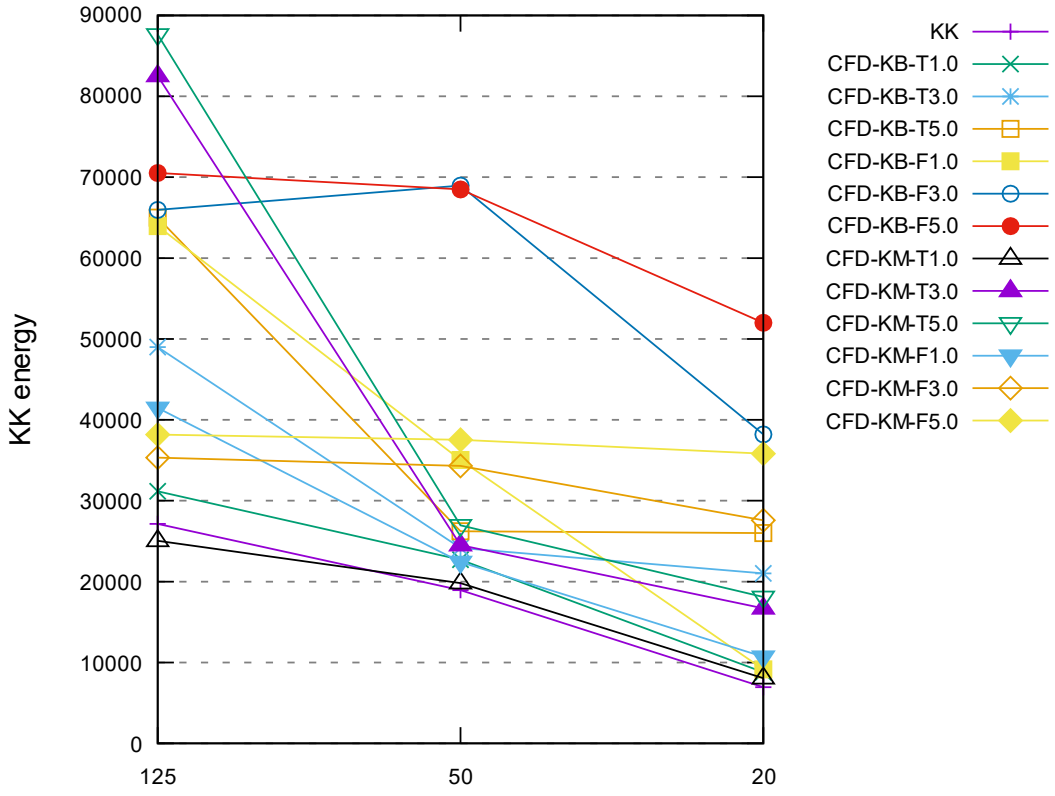


(d)

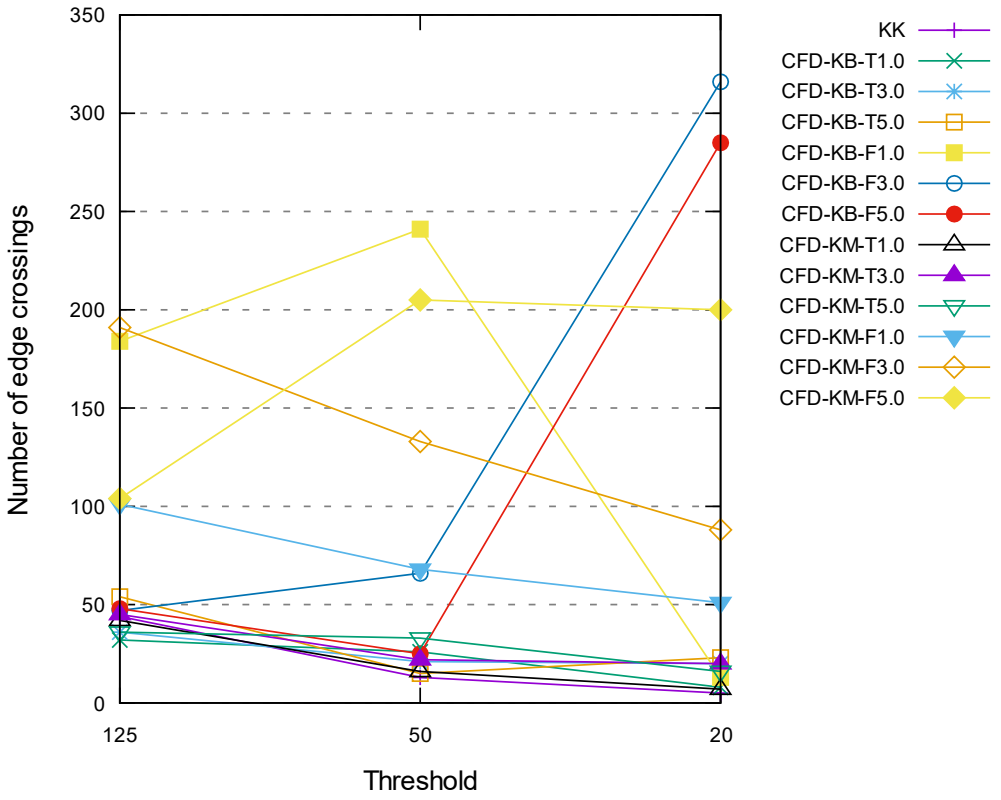
Figure 9: Comparison of FR, CFD-FB and CFD-FM with 10-minute time limit. (a) standardized variance (b) KK energy (c) edge crossings (d) total time usage.


Standardized variance
Threshold
3
2.5
2
1.5
1
0.5
0
125
50
20
KK
CFD-KB-T1.0
CFD-KB-T3.0
CFD-KB-T5.0
CFD-KB-F1.0
CFD-KB-F3.0
CFD-KB-F5.0
CFD-KM-T1.0
CFD-KM-T3.0
CFD-KM-T5.0
CFD-KM-F1.0
CFD-KM-F3.0
CFD-KM-F5.0

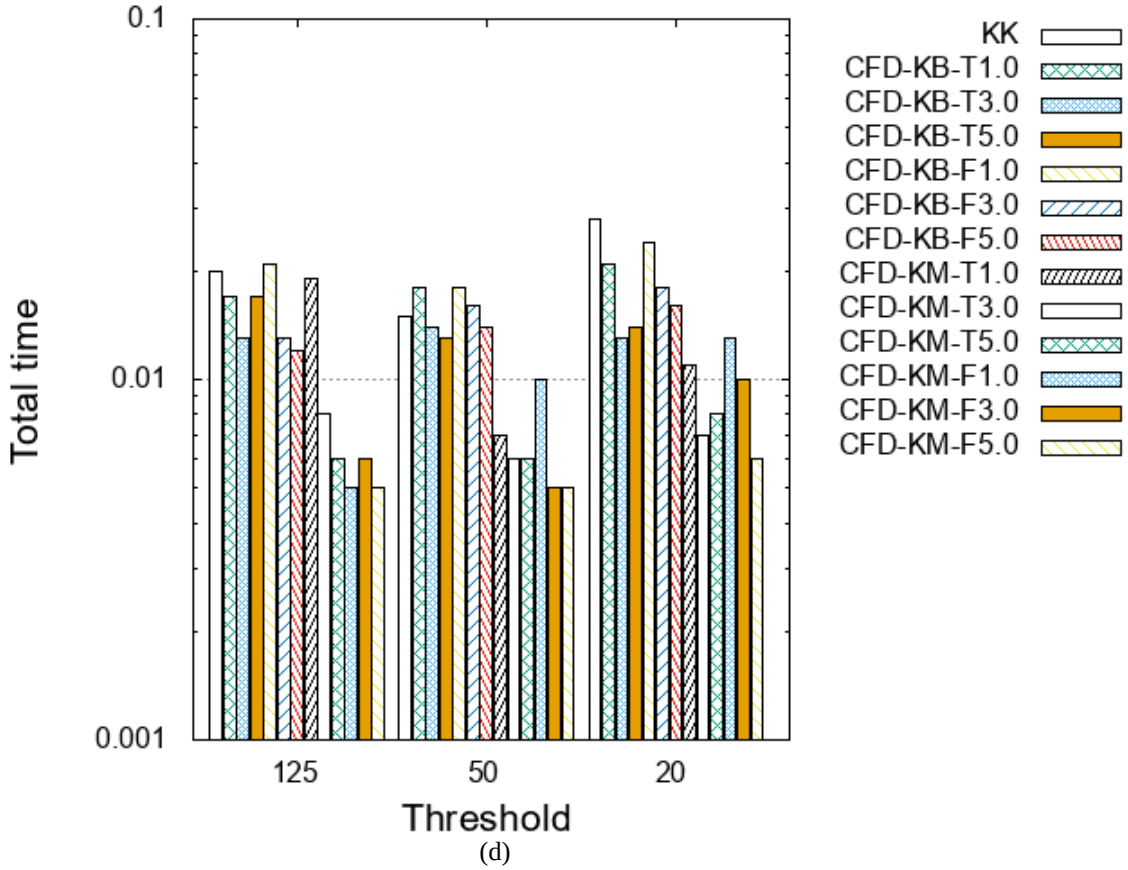
(a)


KK energy
Threshold
90000
80000
70000
60000
50000
40000
30000
20000
10000
0
125
50
20
KK
CFD-KB-T1.0
CFD-KB-T3.0
CFD-KB-T5.0
CFD-KB-F1.0
CFD-KB-F3.0
CFD-KB-F5.0
CFD-KM-T1.0
CFD-KM-T3.0
CFD-KM-T5.0
CFD-KM-F1.0
CFD-KM-F3.0
CFD-KM-F5.0

(b)



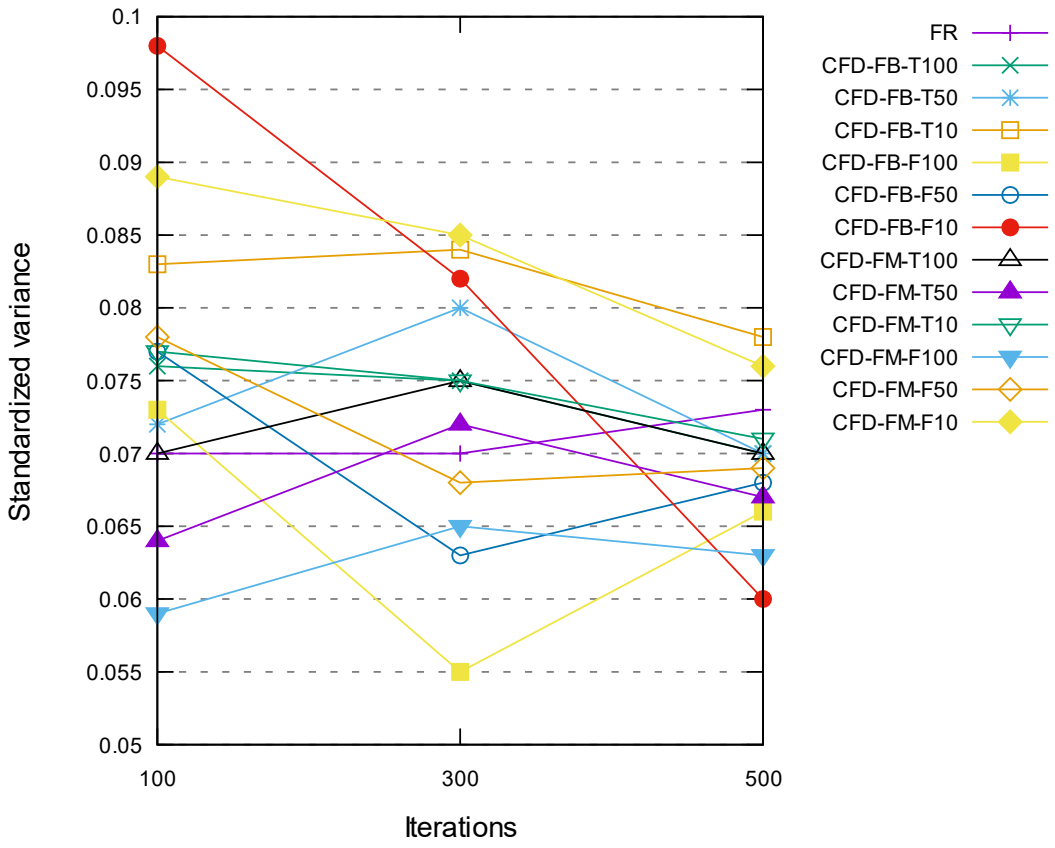
(c)



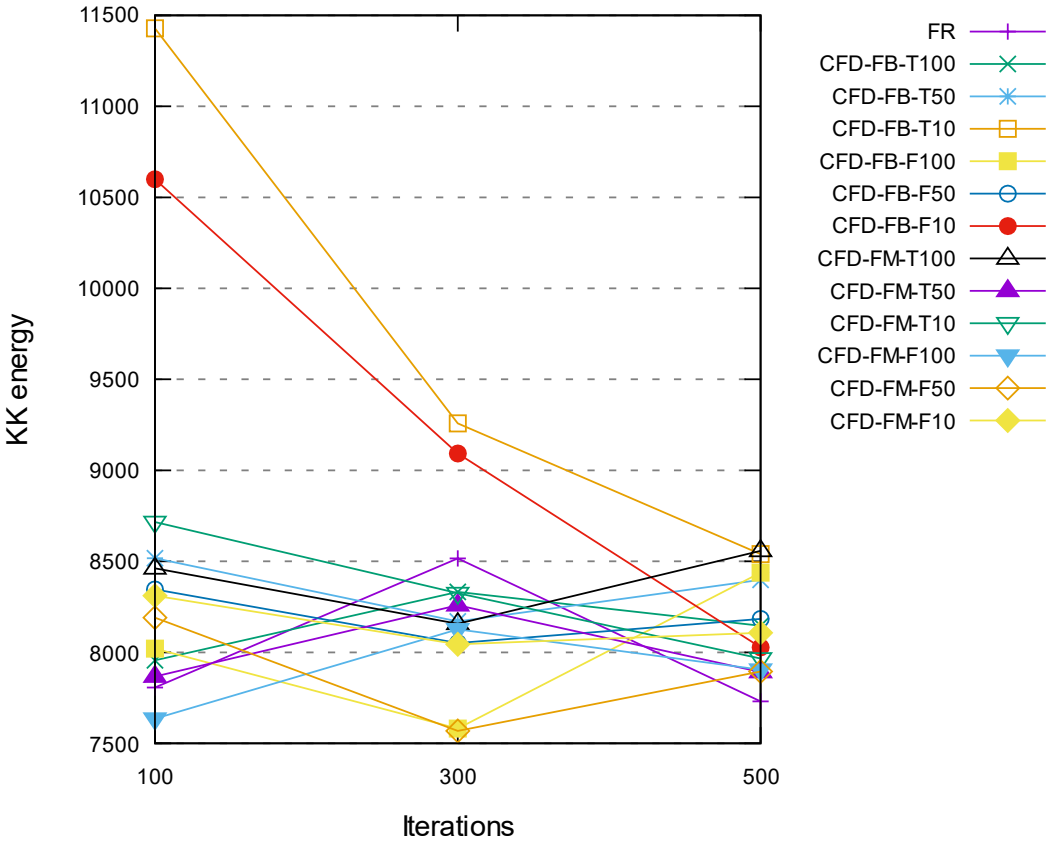
(d)

Figure 10: Comparison of KK, CFD-KB and CFD-KM with a 10-minute time limit. (a) standardized variance (b) KK energy (c) edge crossings (d) total time usage.

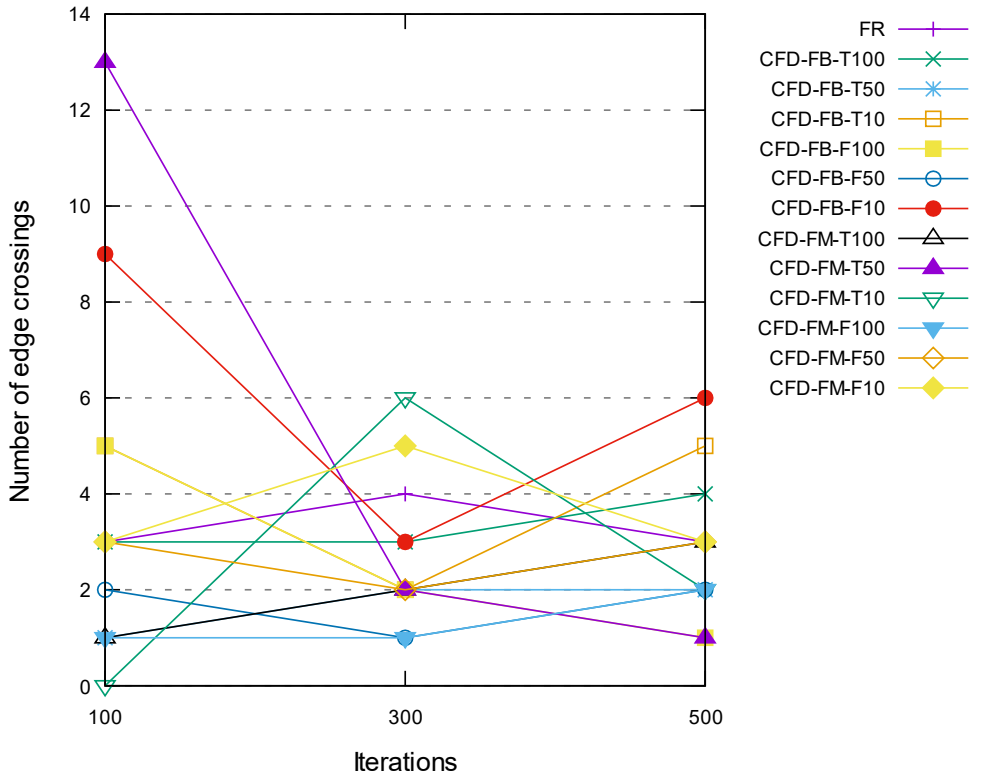
From these results, we can observe that CFD with community FDP enabled generally performs better than CFD with community FDP disabled. For instance, in Figure 10(a), we can see that CFD-KB-F1.0, CFD-KB-F3.0, CFD-KB-F5.0, CFD-KM-F1.0, CFD-KM-F3.0 and CFD-KM-F5.0 perform worse than other algorithms. The quality of the layouts generated by KK, CFD-KB and CFD-KM is much worse than FR, CFD-FB, and CFD-FM. The quality of the layouts generated by KK, CFD-KB and CFD-KM improves if a low threshold is used in KK. However, CFD-FB is still superior in reducing edge crossings. Figure 11 and Figure 12 depict the results of 30-minute time limit whereas Figure 13 and Figure 14 show the results of 60-minute time limit. Those results are similar to the results of 10-minute time limit depicted in Figure 9 and Figure 10.


Standardized variance
0.1
0.095
0.09
0.085
0.08
0.075
0.07
0.065
0.06
0.055
0.05
100
300
500
Iterations
FR
CFD-FB-T100
CFD-FB-T50
CFD-FB-T10
CFD-FB-F100
CFD-FB-F50
CFD-FB-F10
CFD-FM-T100
CFD-FM-T50
CFD-FM-T10
CFD-FM-F100
CFD-FM-F50
CFD-FM-F10

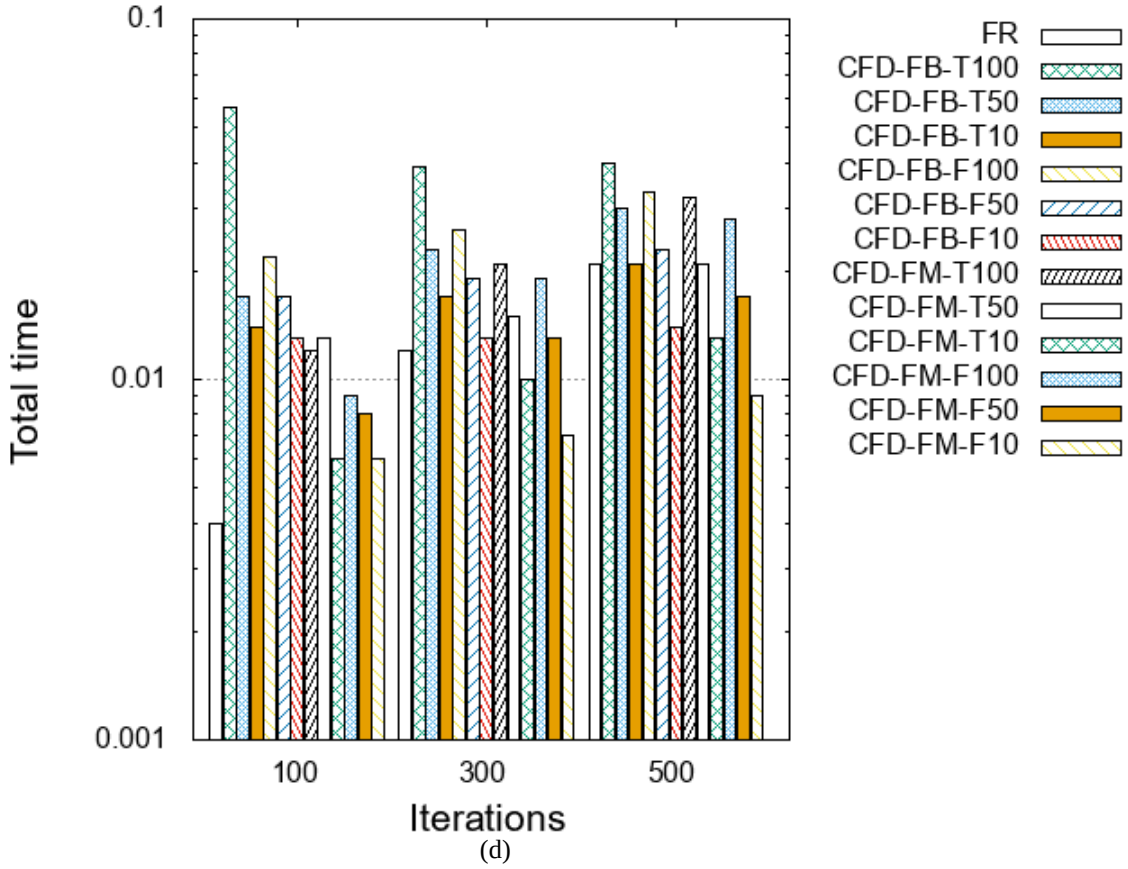
(a)


KK energy
11500
11000
10500
10000
9500
9000
8500
8000
7500
100
300
500
Iterations
FR
CFD-FB-T100
CFD-FB-T50
CFD-FB-T10
CFD-FB-F100
CFD-FB-F50
CFD-FB-F10
CFD-FM-T100
CFD-FM-T50
CFD-FM-T10
CFD-FM-F100
CFD-FM-F50
CFD-FM-F10

(b)

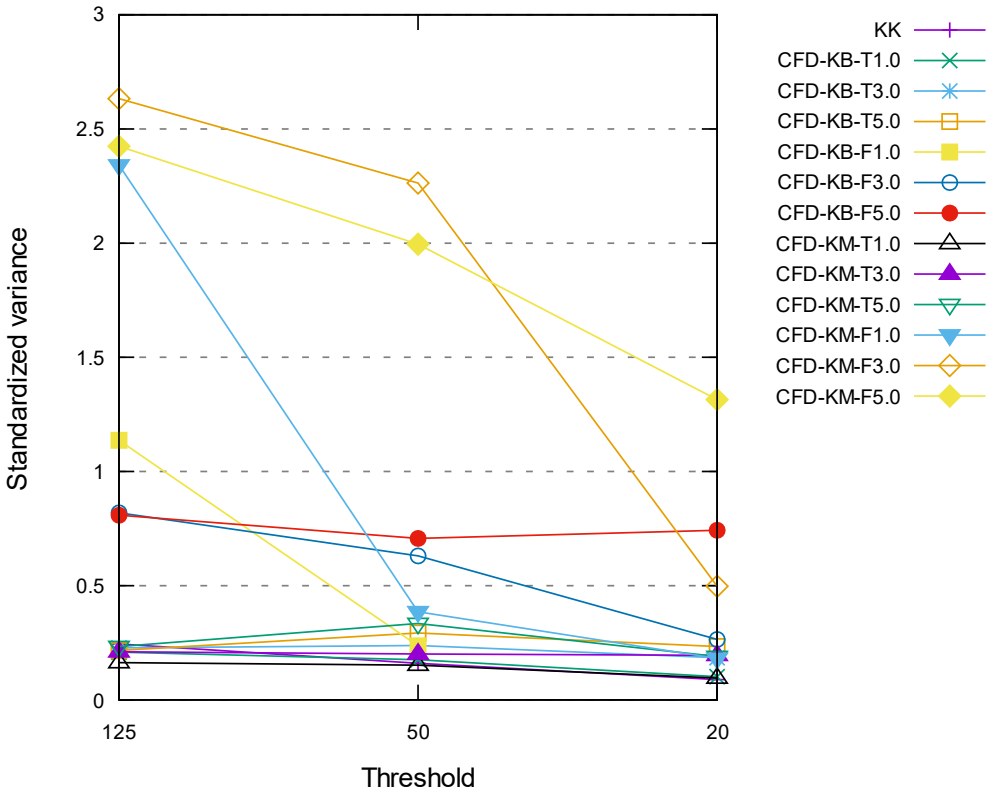


(c)

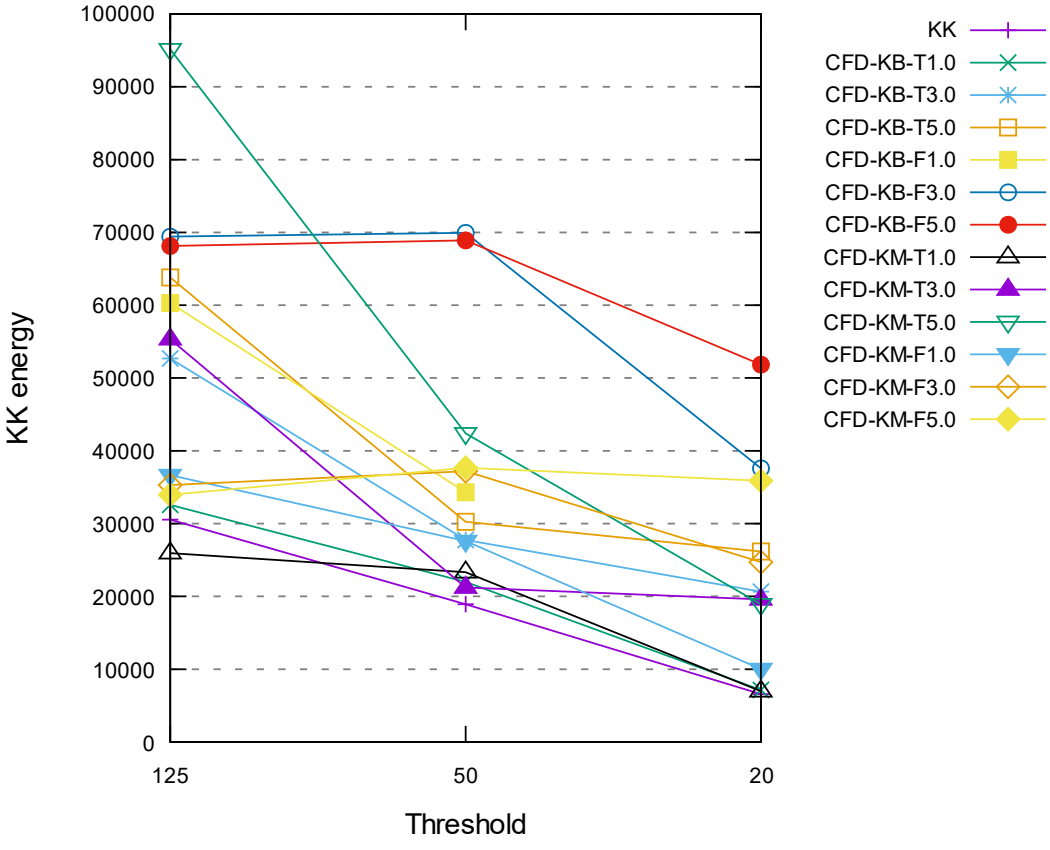


(d)

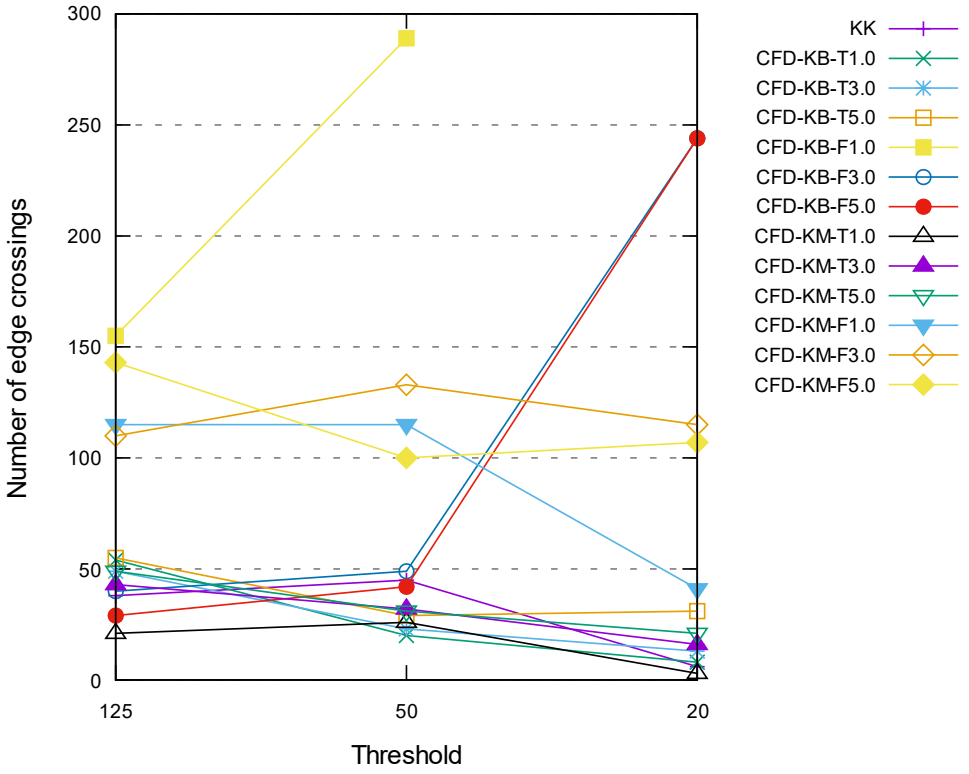
Figure 11: Comparison of FR, CFD-FB and CFD-FM with 30-minute time limit. (a) standardized variance (b) KK energy (c) edge crossings (d) total time usage.


Standardized variance
Threshold
3
2.5
2
1.5
1
0.5
0
125
50
20
KK
CFD-KB-T1.0
CFD-KB-T3.0
CFD-KB-T5.0
CFD-KB-F1.0
CFD-KB-F3.0
CFD-KB-F5.0
CFD-KM-T1.0
CFD-KM-T3.0
CFD-KM-T5.0
CFD-KM-F1.0
CFD-KM-F3.0
CFD-KM-F5.0

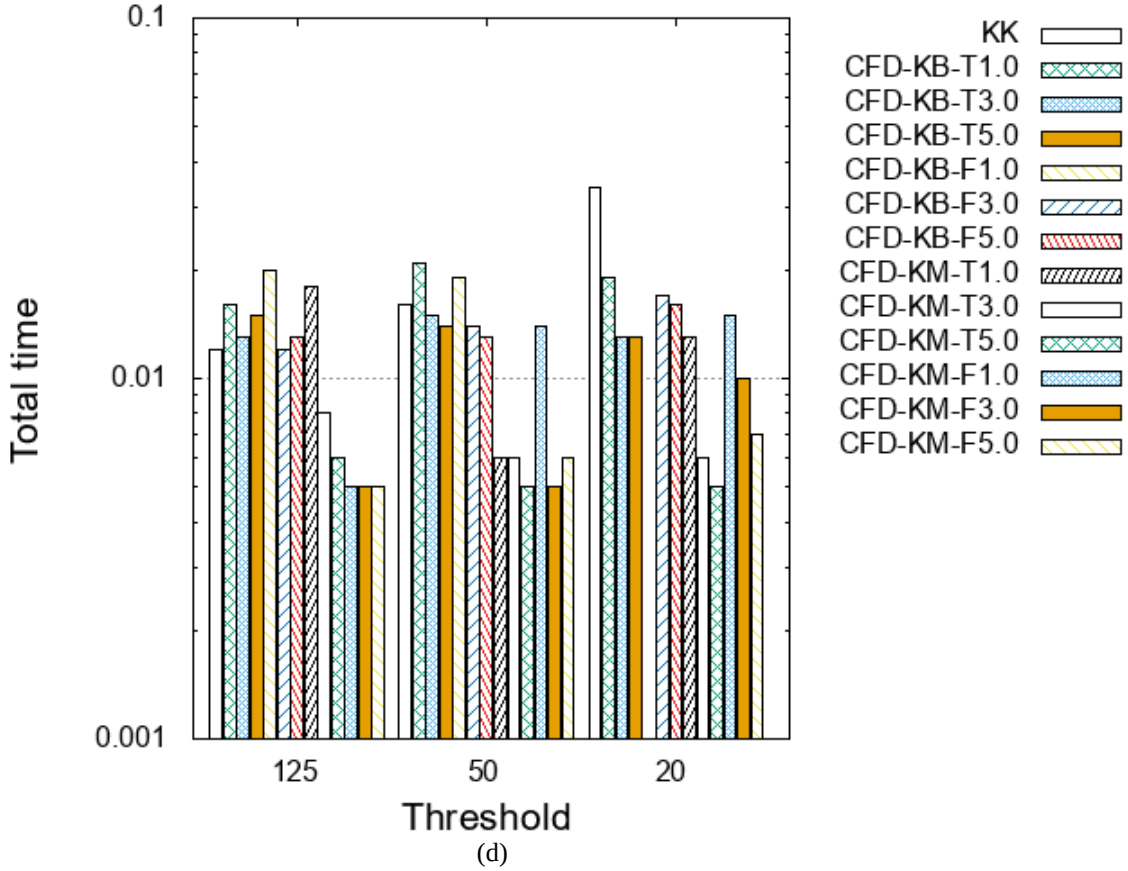
(a)


KK energy
Threshold
100000
90000
80000
70000
60000
50000
40000
30000
20000
10000
0
125
50
20
KK
CFD-KB-T1.0
CFD-KB-T3.0
CFD-KB-T5.0
CFD-KB-F1.0
CFD-KB-F3.0
CFD-KB-F5.0
CFD-KM-T1.0
CFD-KM-T3.0
CFD-KM-T5.0
CFD-KM-F1.0
CFD-KM-F3.0
CFD-KM-F5.0

(b)

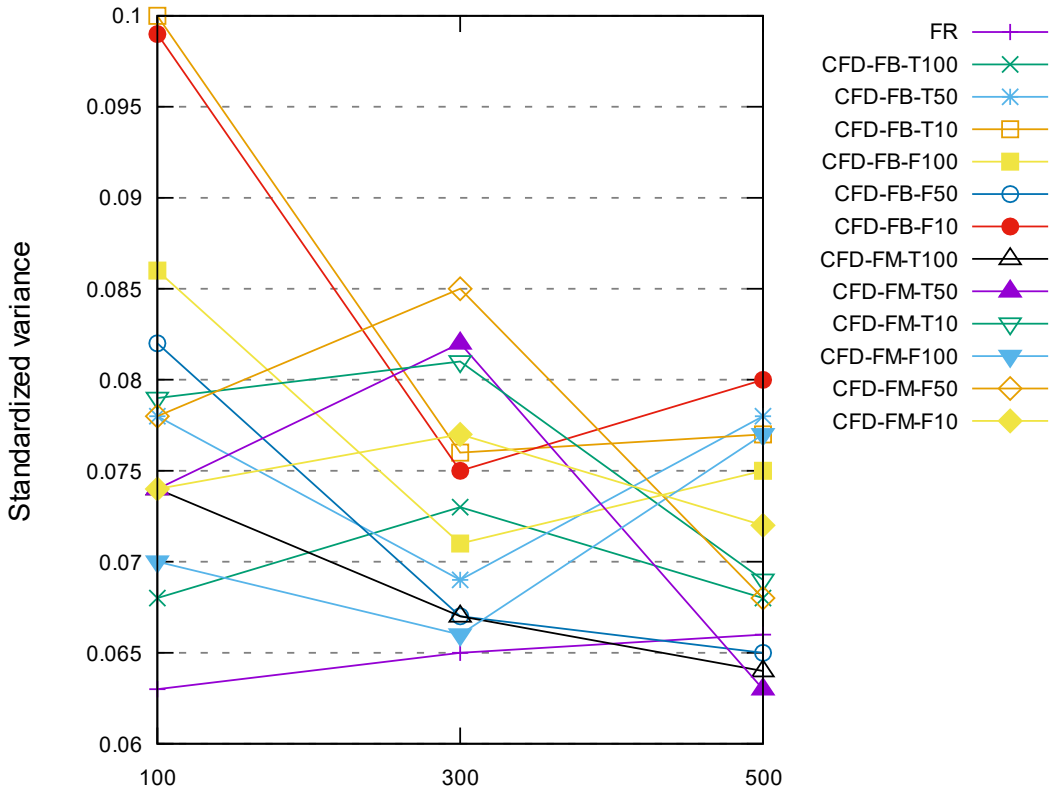


(c)

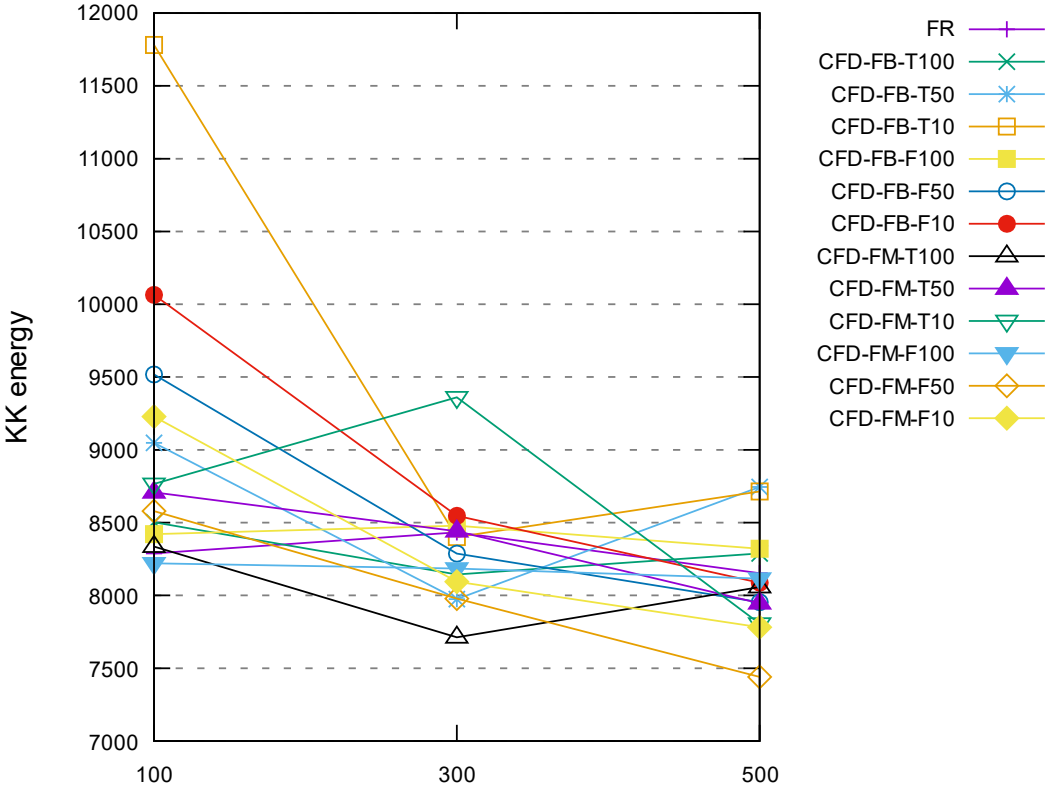


(d)

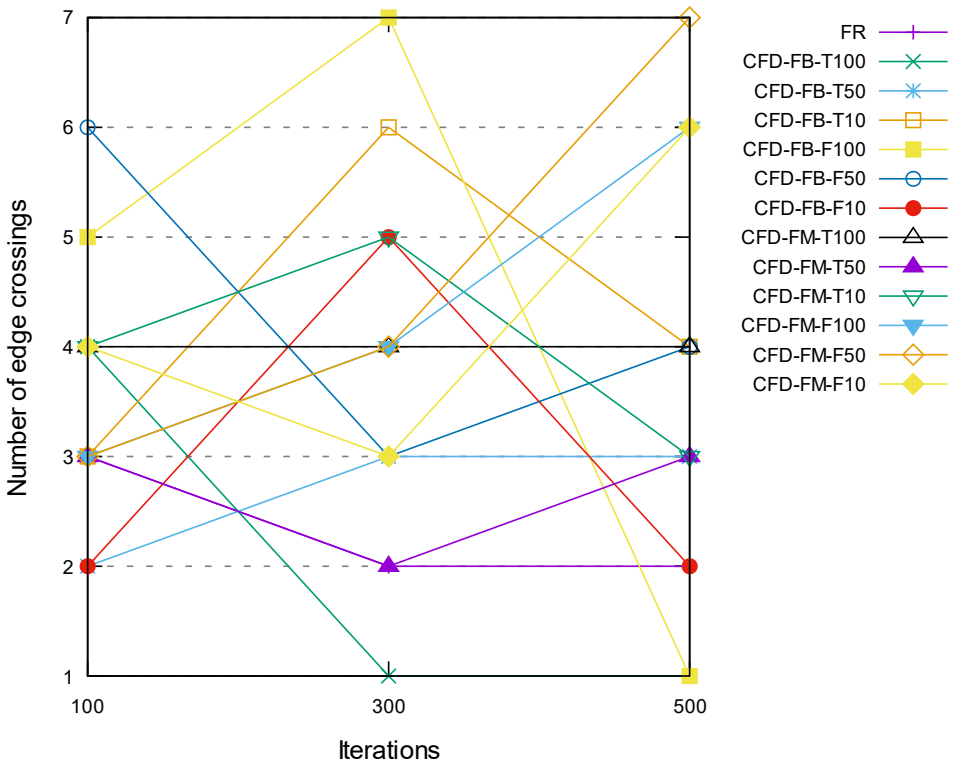
Figure 12: Comparison of KK, CFD-KB and CFD-KM with a 30-minute time limit. (a) standardized variance (b) KK energy (c) edge crossings (d) total time usage. (Missing data denotes timeout for the corresponding execution).


Standardized variance
0.1
0.095
0.09
0.085
0.08
0.075
0.07
0.065
0.06
100
300
500
Iterations
FR
CFD-FB-T100
CFD-FB-T50
CFD-FB-T10
CFD-FB-F100
CFD-FB-F50
CFD-FB-F10
CFD-FM-T100
CFD-FM-T50
CFD-FM-T10
CFD-FM-F100
CFD-FM-F50
CFD-FM-F10

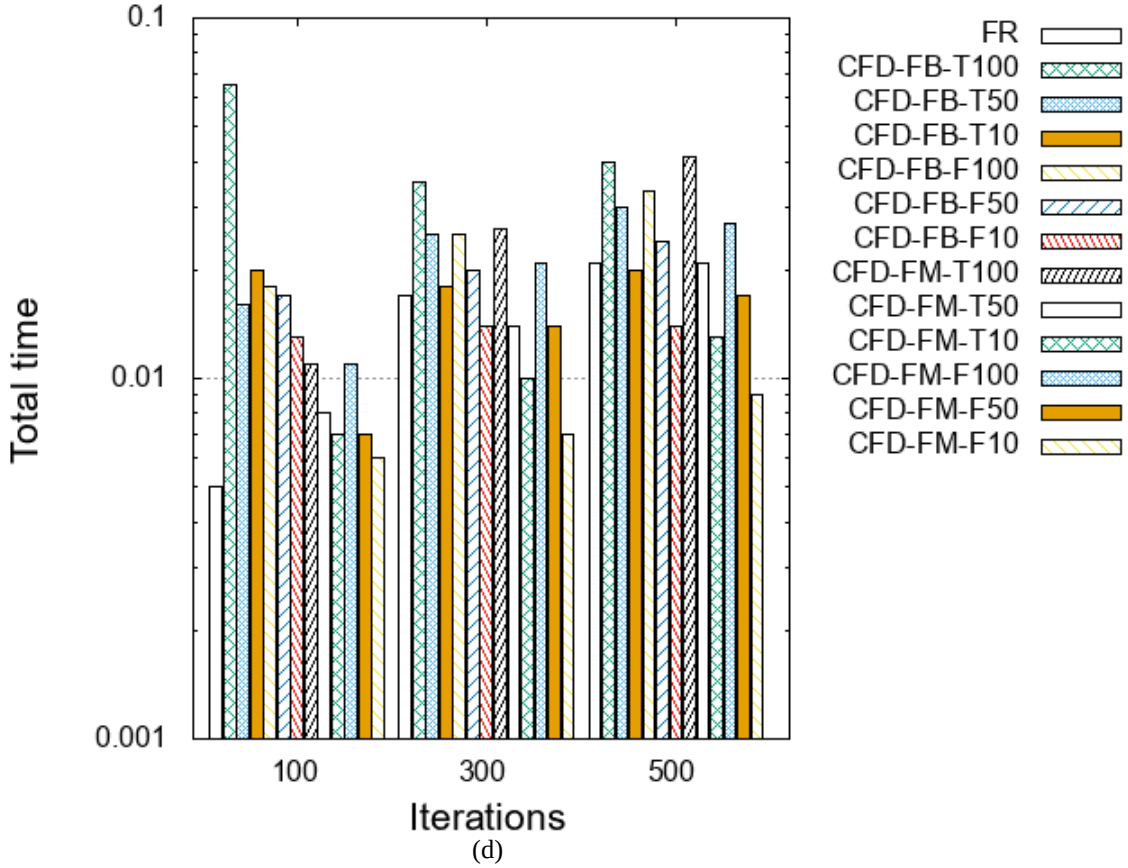
(a)


KK energy
12000
11500
11000
10500
10000
9500
9000
8500
8000
7500
7000
100
300
500
Iterations
FR
CFD-FB-T100
CFD-FB-T50
CFD-FB-T10
CFD-FB-F100
CFD-FB-F50
CFD-FB-F10
CFD-FM-T100
CFD-FM-T50
CFD-FM-T10
CFD-FM-F100
CFD-FM-F50
CFD-FM-F10

(b)

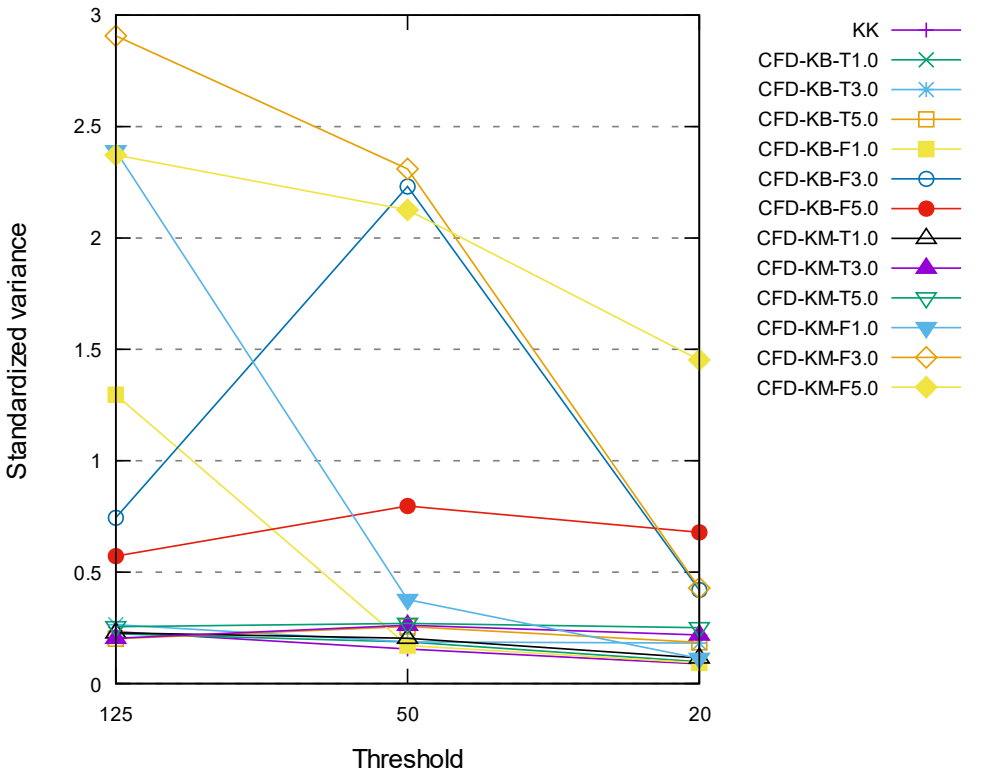


(c)

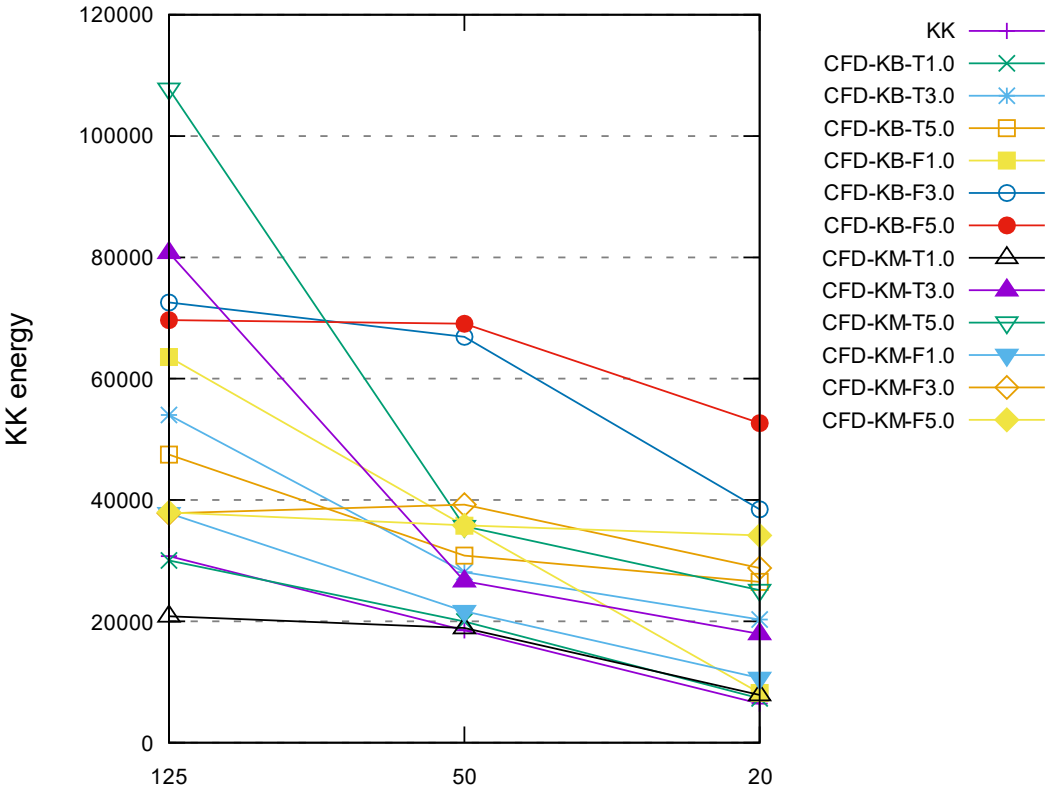


(d)

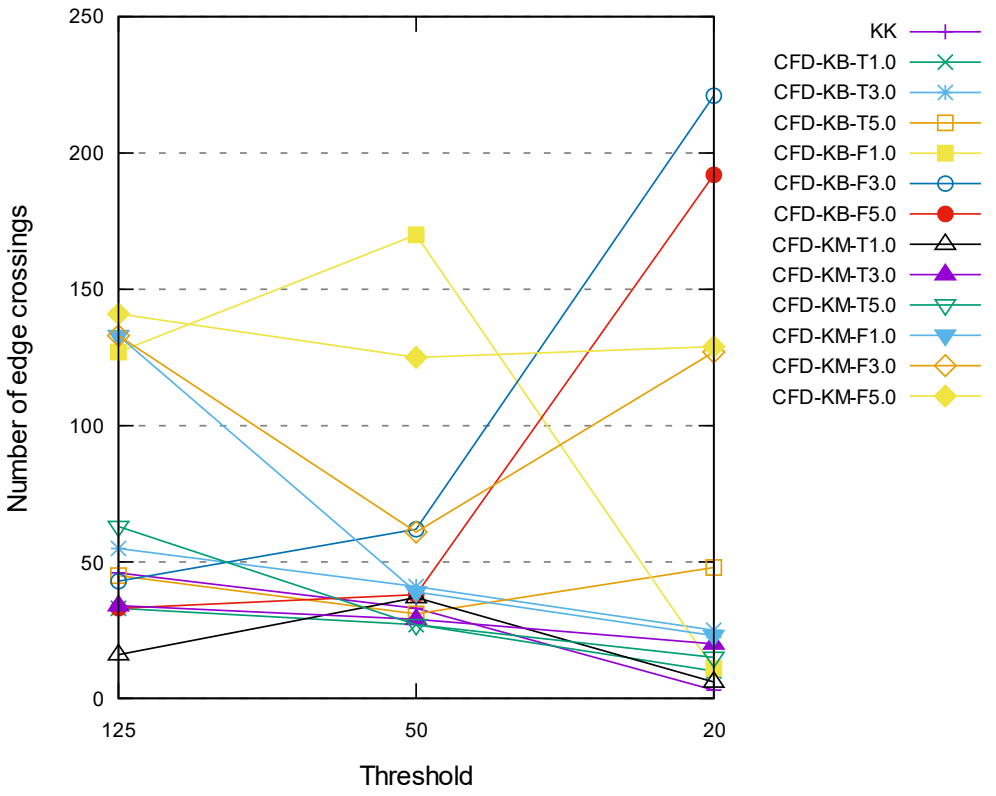
Figure 13: Comparison of FR, CFD-FB and CFD-FM with 60-minute time limit. (a) standardized variance (b) KK energy (c) edge crossings (d) total time usage.


Standardized variance
3
2.5
2
1.5
1
0.5
0
125
50
20
Threshold
KK
CFD-KB-T1.0
CFD-KB-T3.0
CFD-KB-T5.0
CFD-KB-F1.0
CFD-KB-F3.0
CFD-KB-F5.0
CFD-KM-T1.0
CFD-KM-T3.0
CFD-KM-T5.0
CFD-KM-F1.0
CFD-KM-F3.0
CFD-KM-F5.0

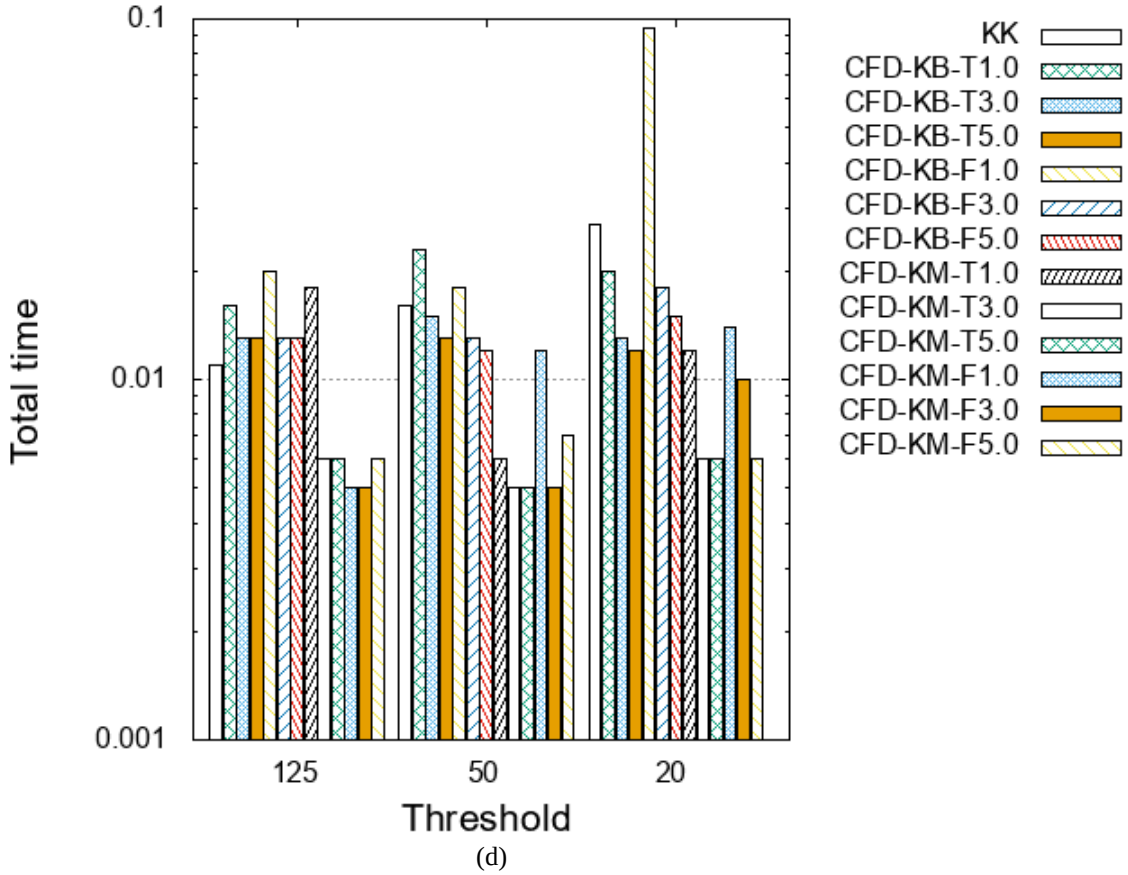
(a)


KK energy
120000
100000
80000
60000
40000
20000
0
125
50
20
Threshold
KK
CFD-KB-T1.0
CFD-KB-T3.0
CFD-KB-T5.0
CFD-KB-F1.0
CFD-KB-F3.0
CFD-KB-F5.0
CFD-KM-T1.0
CFD-KM-T3.0
CFD-KM-T5.0
CFD-KM-F1.0
CFD-KM-F3.0
CFD-KM-F5.0

(b)



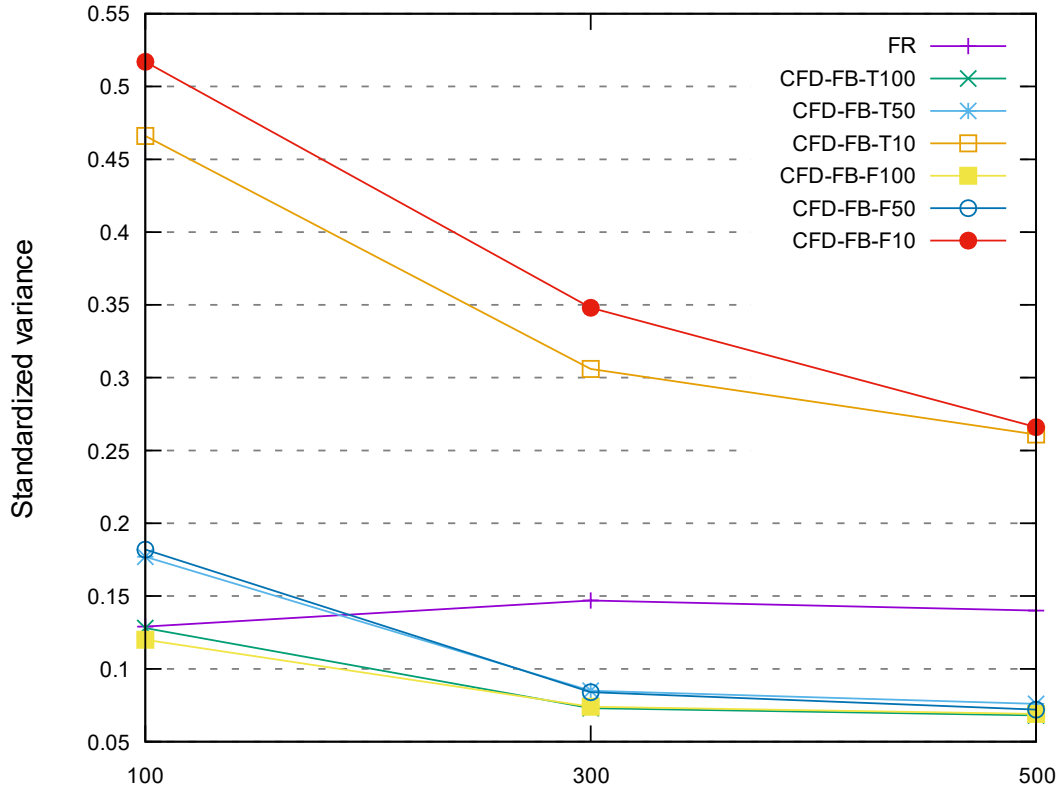
(c)



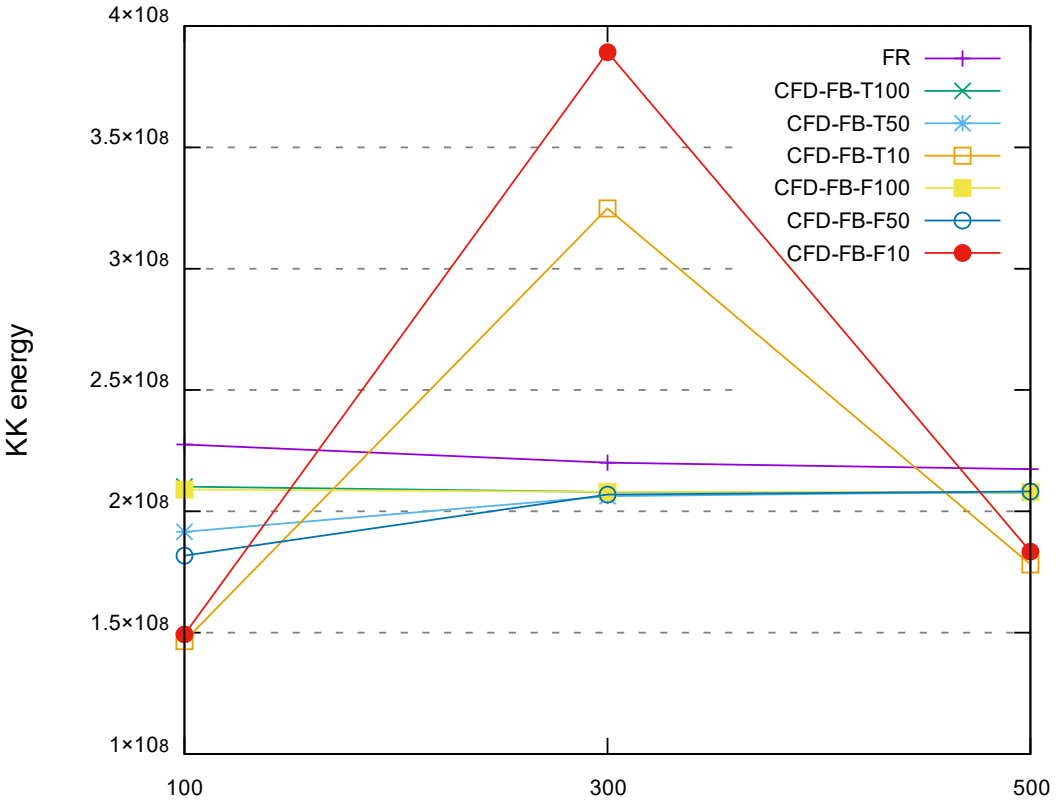
(d)

Figure 14: Comparison of KK, CFD-KB and CFD-KM with 60-minute time limit. (a) standardized variance (b) KK energy (c) edge crossings (d) total time usage.

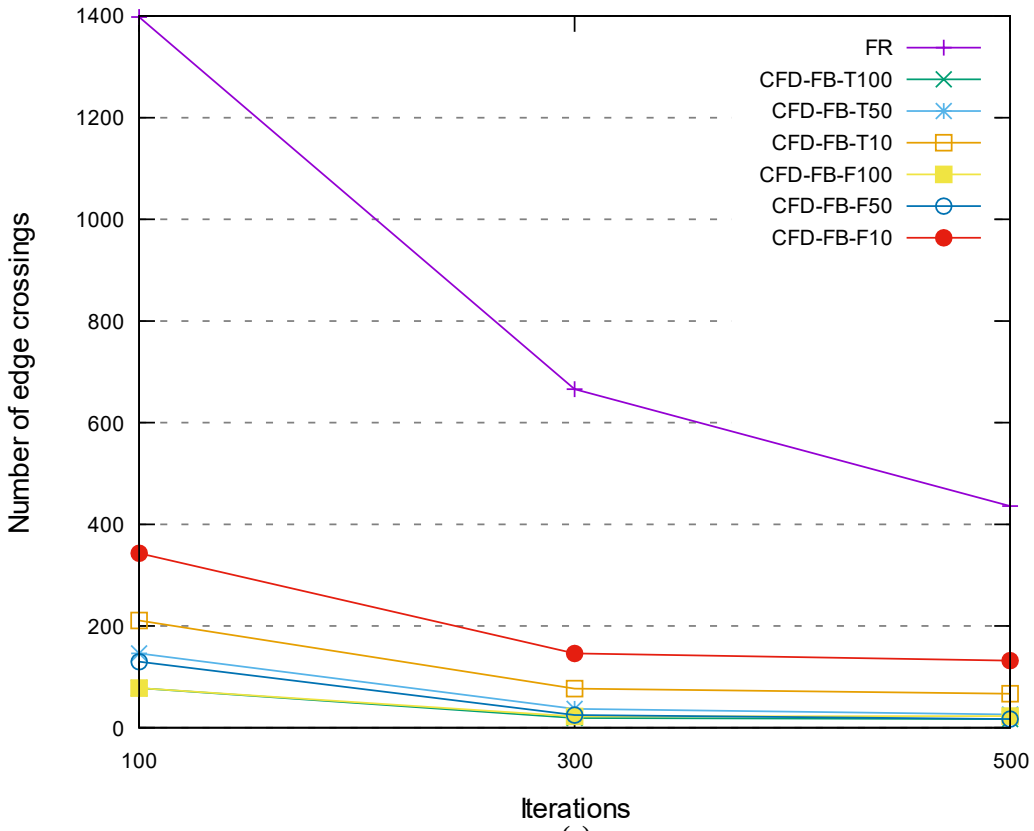
Dataset: Power grid (4941 nodes, 6594 edges) [36]: First, we compare the results of FR, CFD-FB and CFD-FM for 10 minutes execution time. There is no result for CFD-FM because MCL cannot produce output results within 10 minutes. MCL is inefficient in handling large graphs due to its complexity. Figure 15(a), (b) and (c) reveal that CFD-FB with appropriate configuration outperforms FR in all 3 aspects. Figure 15(d) shows that CFD-FB requires more time compared to FR. Experiments are also performed for KK, CFD-KB and CFD-KM with a 10-minute time limit. The conclusions for the experiment results given in Figure 16 are also similar to FR, CFD-FB and CFD-FM tests.


Standardized variance
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
100
300
500
Iterations
FR
CFD-FB-T100
CFD-FB-T50
CFD-FB-T10
CFD-FB-F100
CFD-FB-F50
CFD-FB-F10

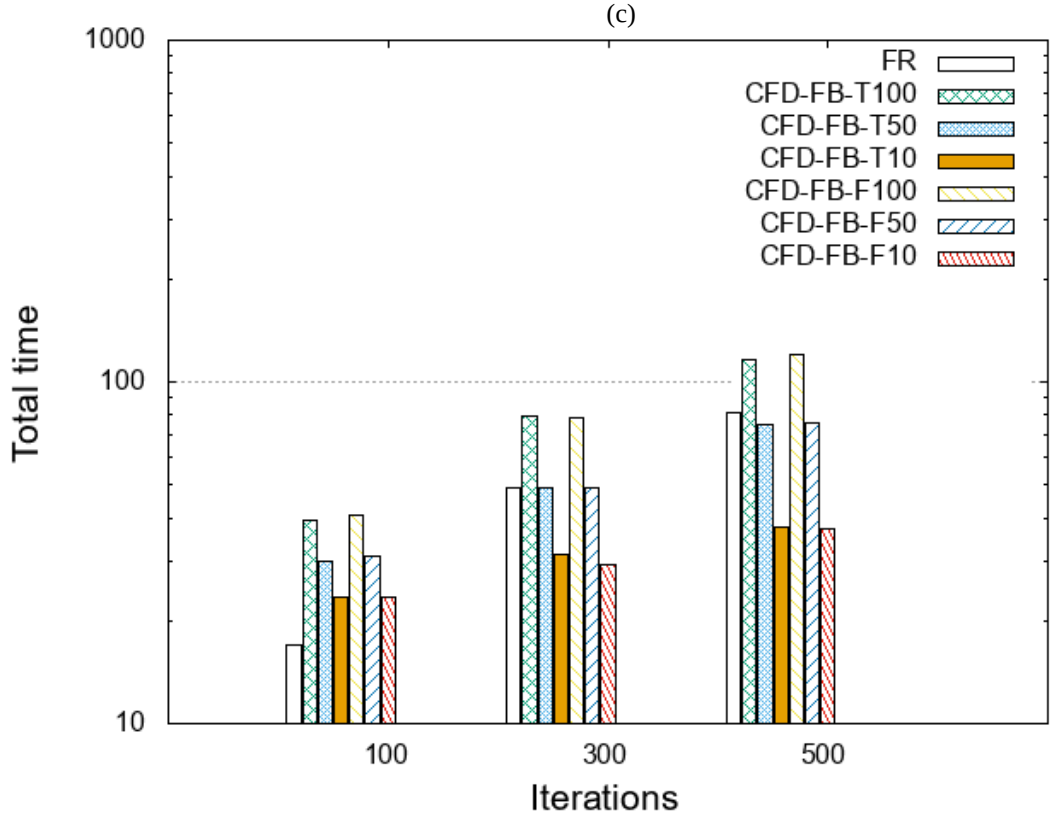
(a)


KK energy
4×10⁸
3.5×10⁸
3×10⁸
2.5×10⁸
2×10⁸
1.5×10⁸
1×10⁸
100
300
500
Iterations
FR
CFD-FB-T100
CFD-FB-T50
CFD-FB-T10
CFD-FB-F100
CFD-FB-F50
CFD-FB-F10

(b)

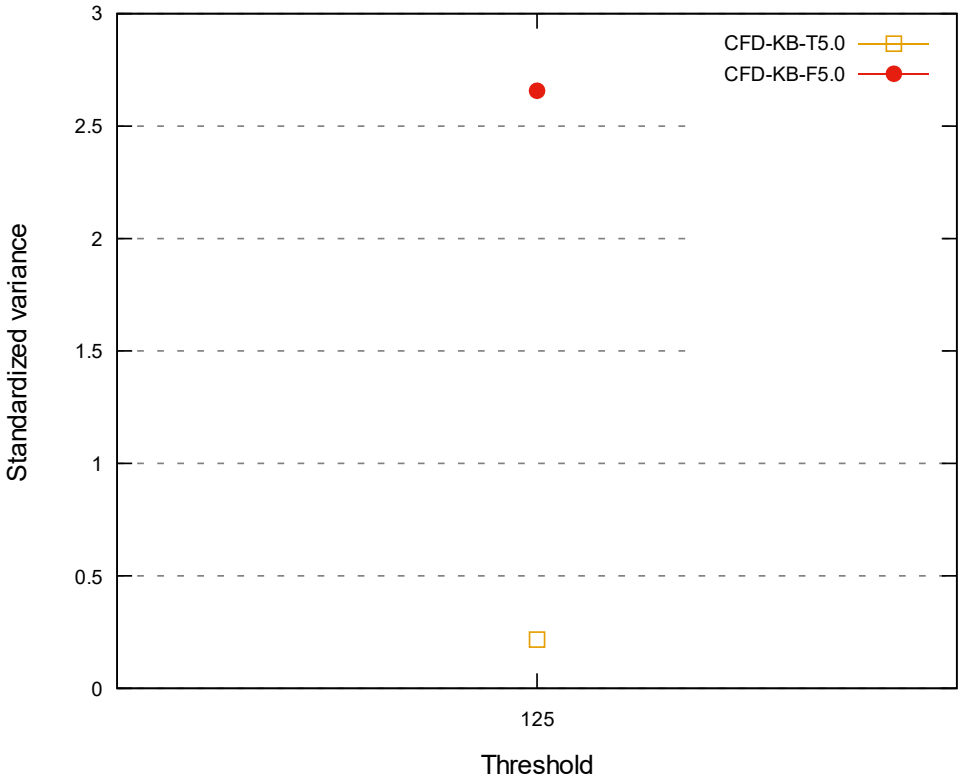


(c)

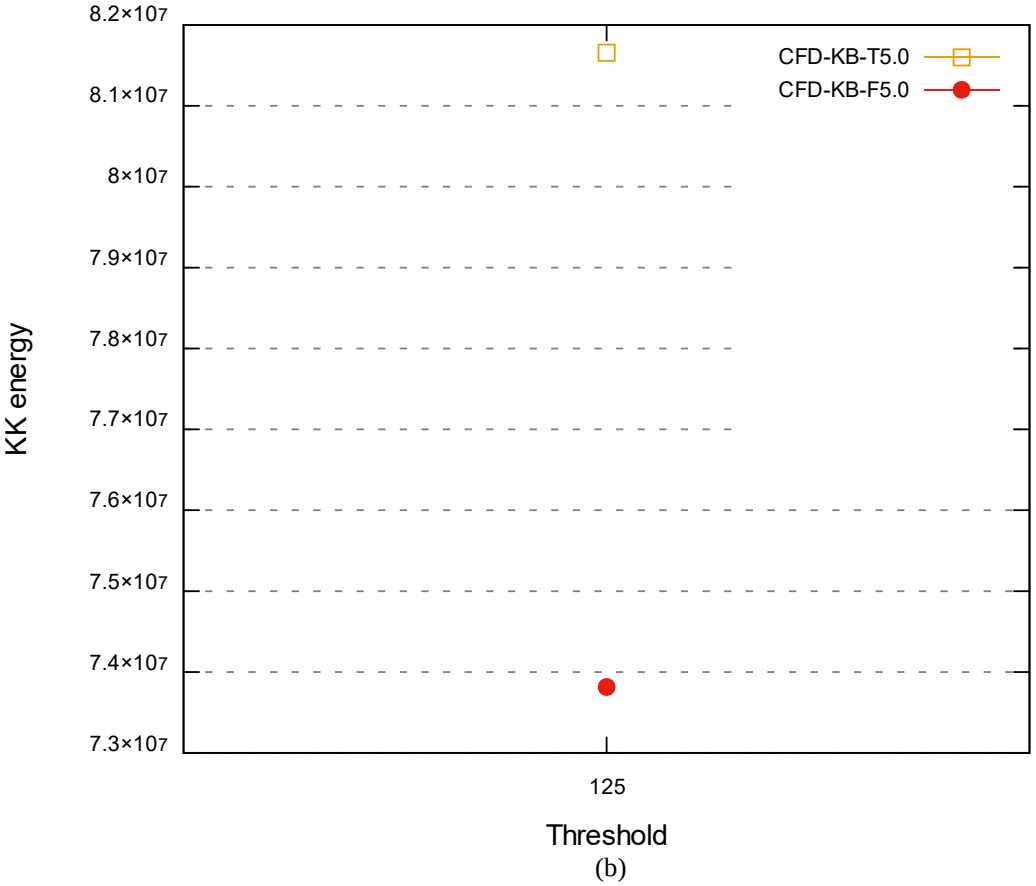


(d)

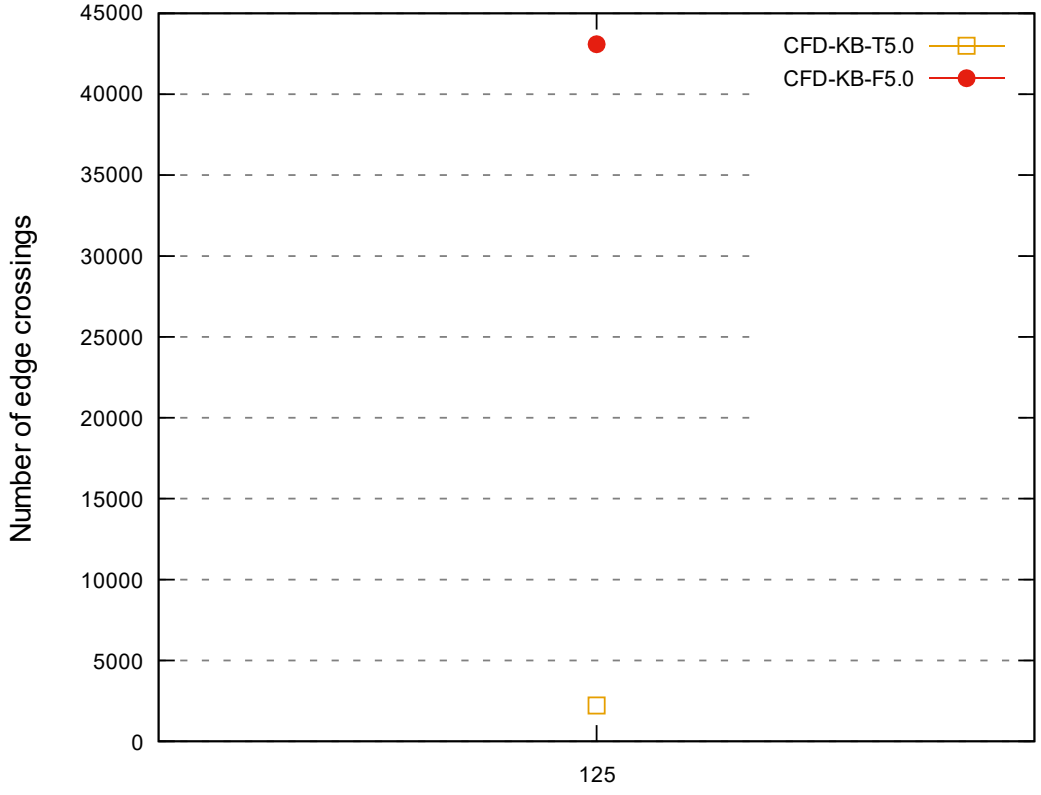
Figure 15: Comparison of FR, CFD-FB and CFD-FM with 10-minute time limit. (a) standardized variance (b) KK energy (c) edge crossings (d) total time usage.


CFD-KB-T5.0
CFD-KB-F5.0
Standardized variance
3
2.5
2
1.5
1
0.5
0
125
Threshold

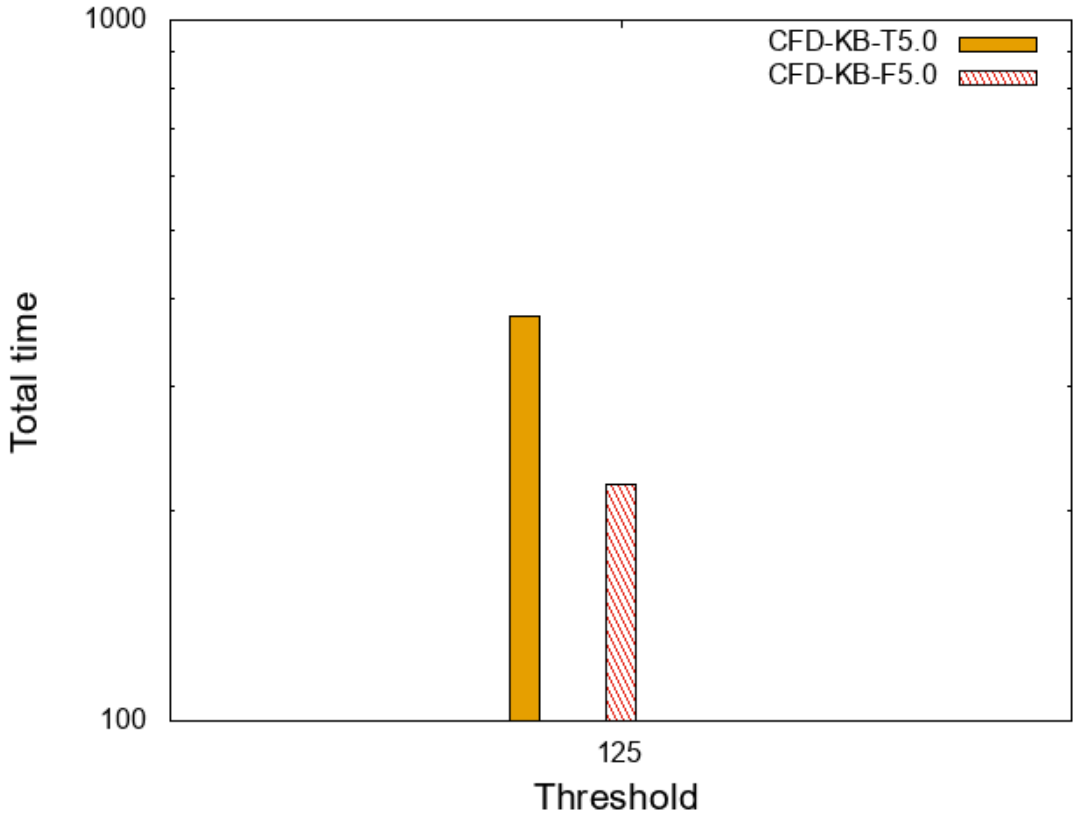
(a)


CFD-KB-T5.0
CFD-KB-F5.0
KK energy
8.2×10⁷
8.1×10⁷
8×10⁷
7.9×10⁷
7.8×10⁷
7.7×10⁷
7.6×10⁷
7.5×10⁷
7.4×10⁷
7.3×10⁷
125
Threshold

(b)



(c)

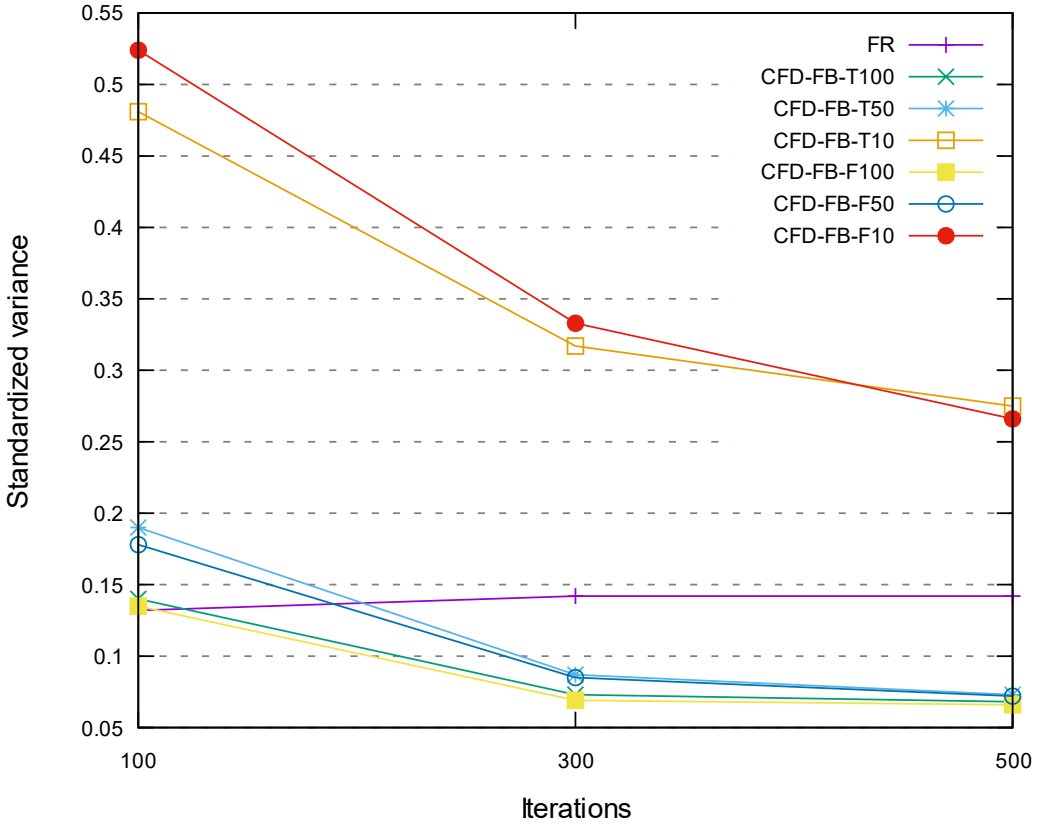


(d)

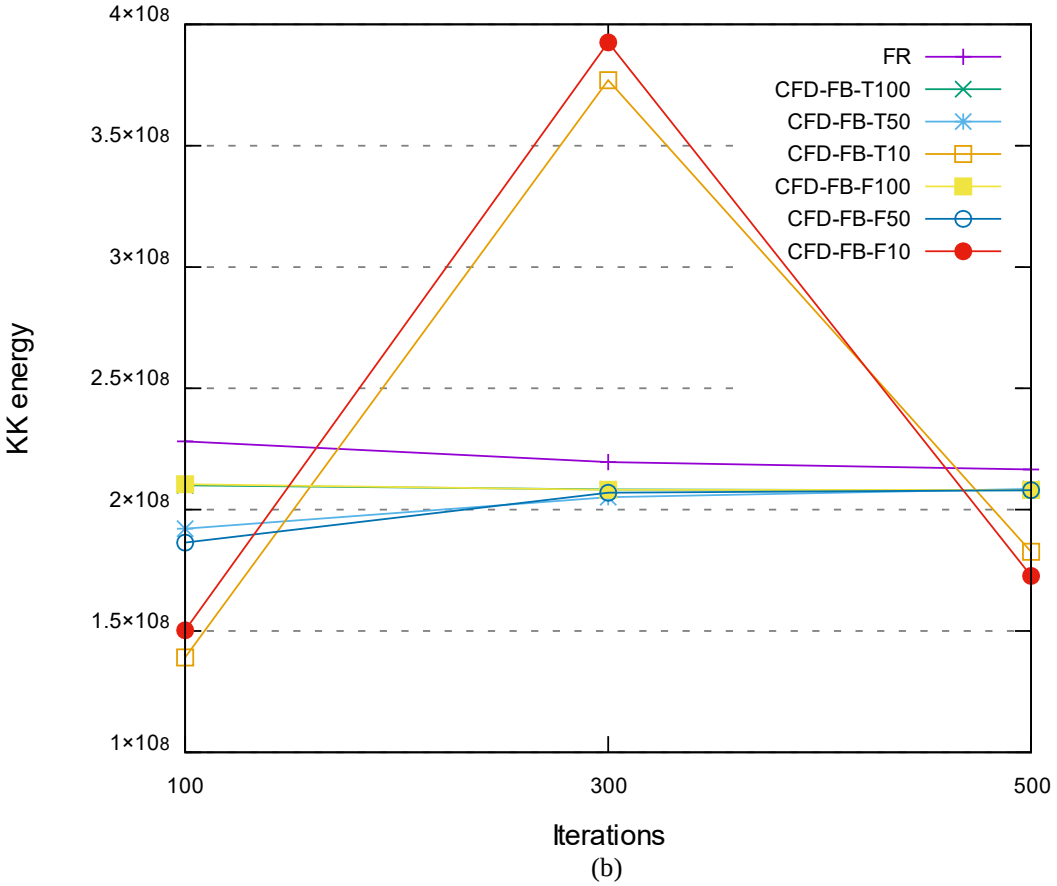
Figure 16: Comparison of KK, CFD-KB and CFD-KM with 10-minute time limit. (a) standardized variance (b) KK energy (c) edge crossings (d) total time usage.

There is almost no result for KK, CFD-KB and CFD-KM because most runs of these algorithms cannot be finished within 10 minutes. This is expected because KK has high computational cost compared to FR. Therefore, KK is less efficient in dealing with large graphs.

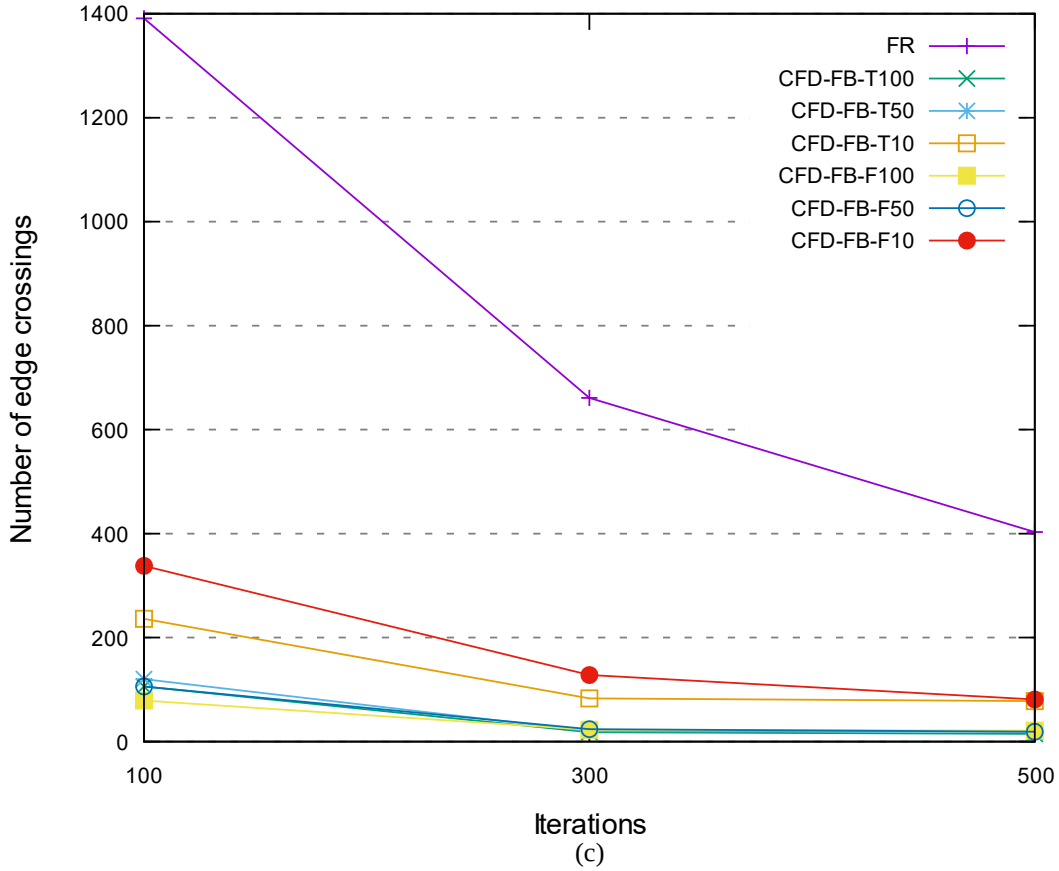
Figure 17 and Figure 18 show the results of the experiments with 30-minute time limit and Figure 19 and Figure 20 show the results of 60-minute time limit.


Standardized variance
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
100
300
500
Iterations
FR
CFD-FB-T100
CFD-FB-T50
CFD-FB-T10
CFD-FB-F100
CFD-FB-F50
CFD-FB-F10

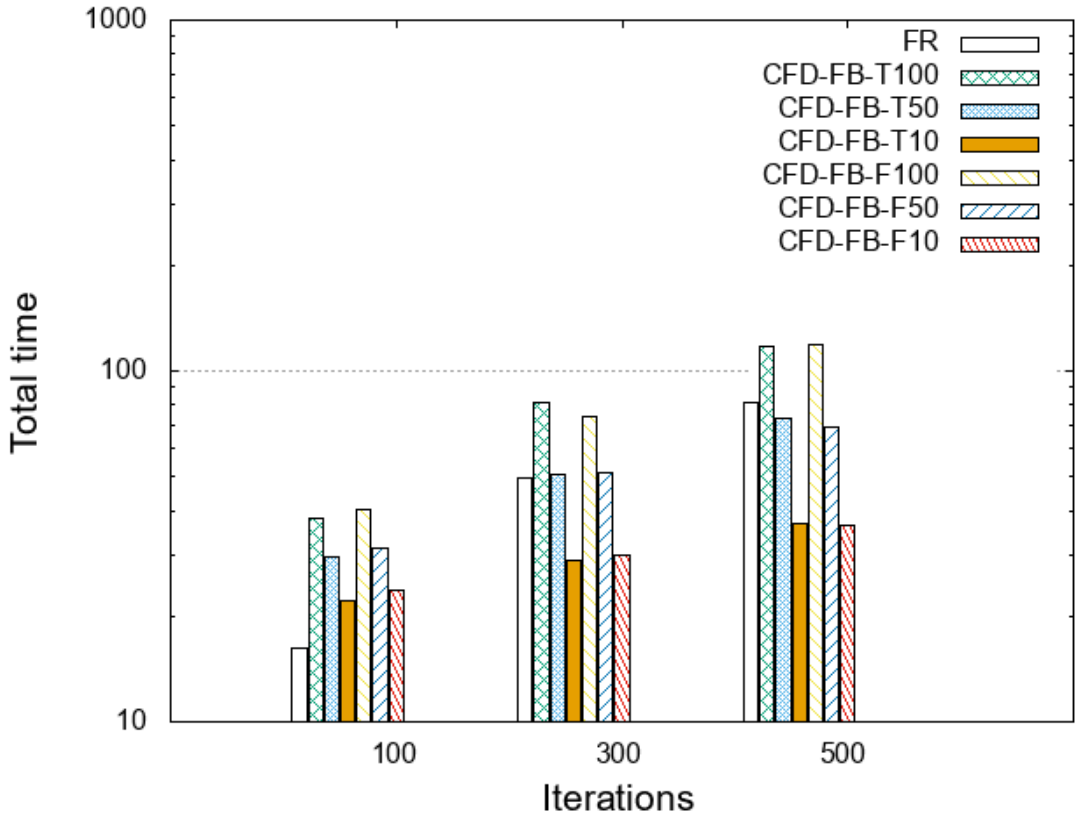
(a)


KK energy
4×10⁸
3.5×10⁸
3×10⁸
2.5×10⁸
2×10⁸
1.5×10⁸
1×10⁸
100
300
500
Iterations
FR
CFD-FB-T100
CFD-FB-T50
CFD-FB-T10
CFD-FB-F100
CFD-FB-F50
CFD-FB-F10

(b)

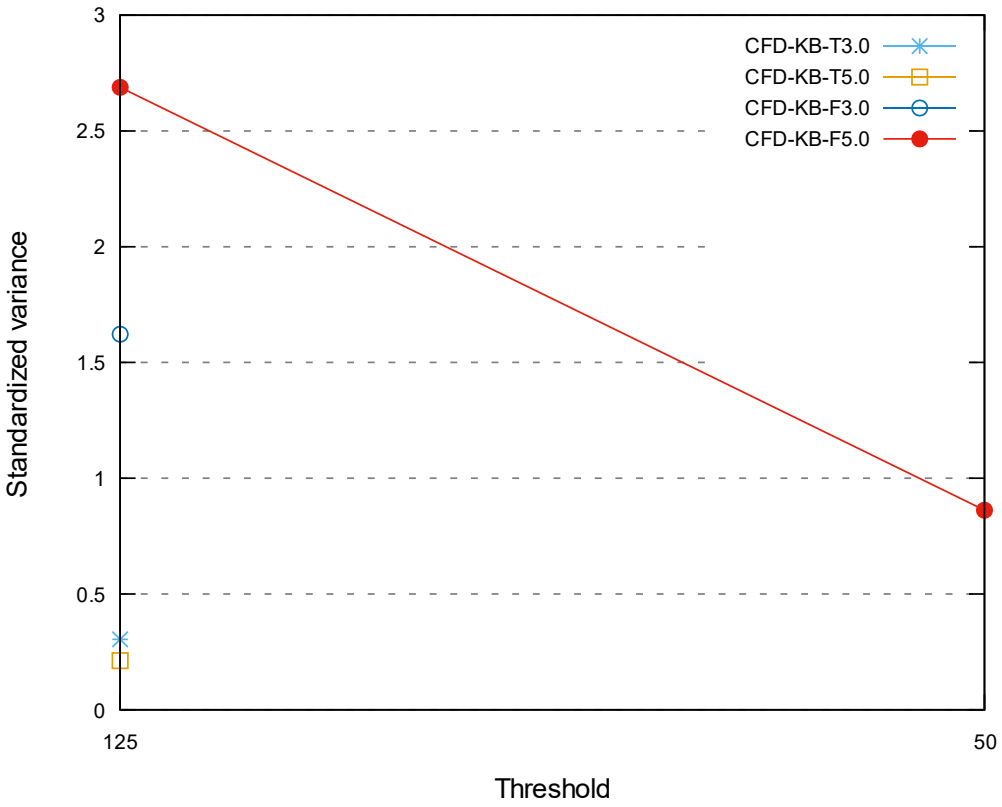


(c)

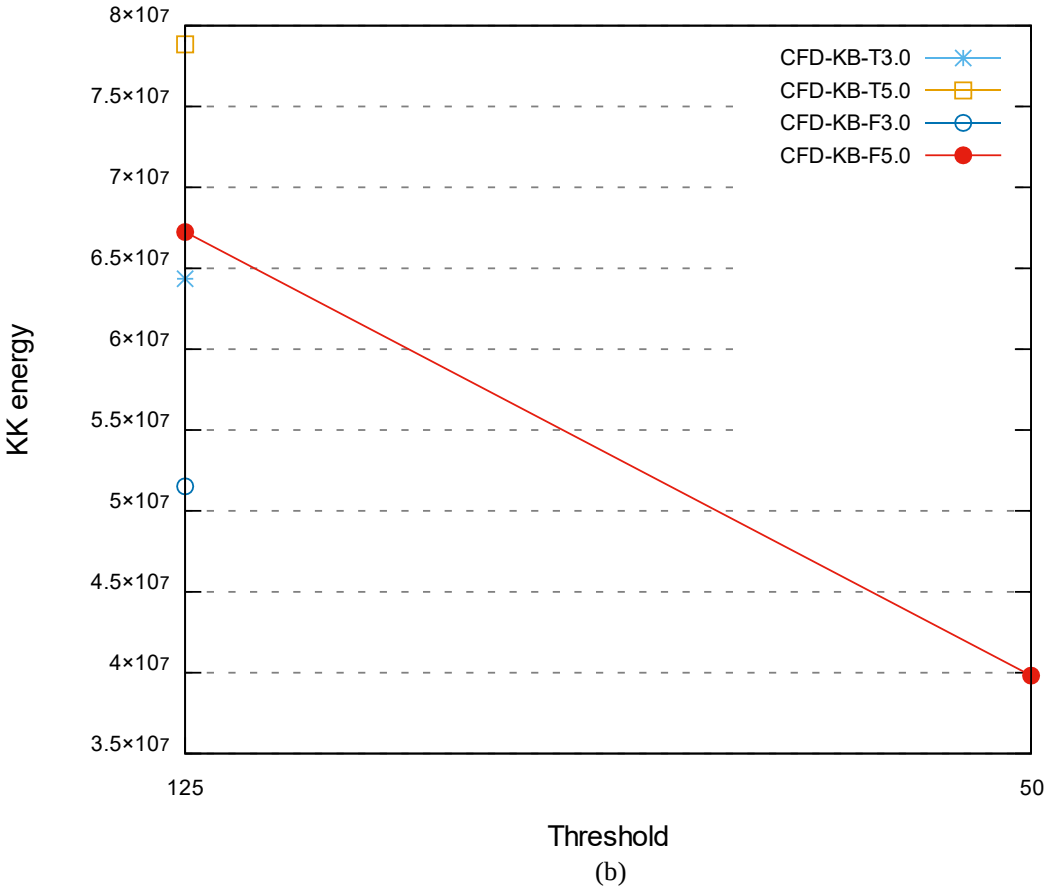


(d)

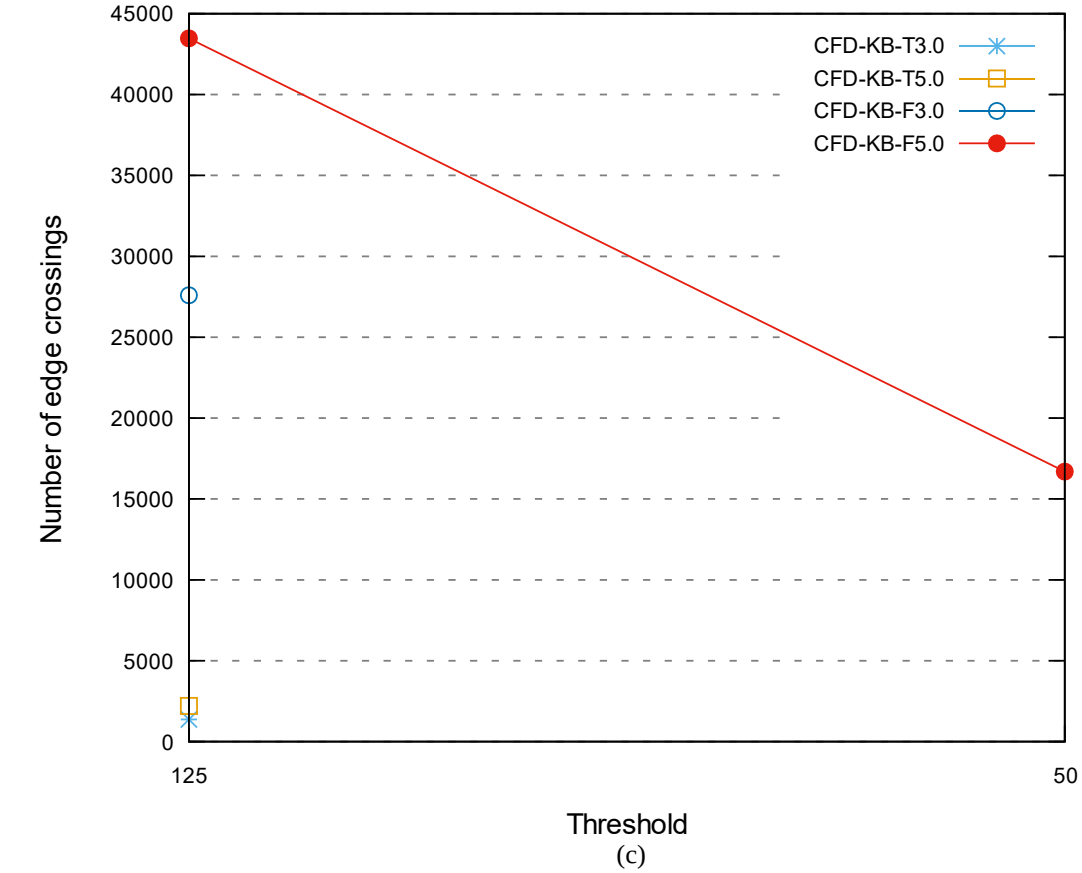
Figure 17: Comparison of FR, CFD-FB and CFD-FM with 30-minute time limit. (a) standardized variance (b) KK energy (c) edge crossings (d) total time usage.


CFD-KB-T3.0
CFD-KB-T5.0
CFD-KB-F3.0
CFD-KB-F5.0
Standardized variance
3
2.5
2
1.5
1
0.5
0
125
50
Threshold

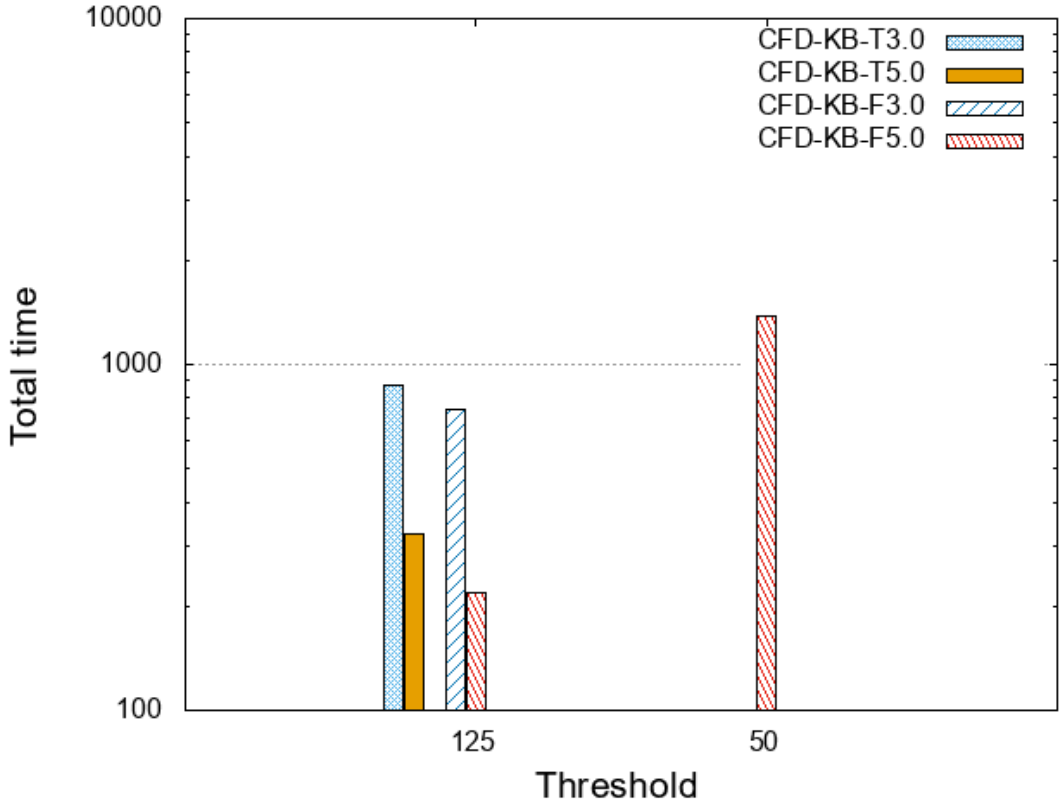
(a)


CFD-KB-T3.0
CFD-KB-T5.0
CFD-KB-F3.0
CFD-KB-F5.0
KK energy
8×107
7.5×107
7×107
6.5×107
6×107
5.5×107
5×107
4.5×107
4×107
3.5×107
125
50
Threshold

(b)

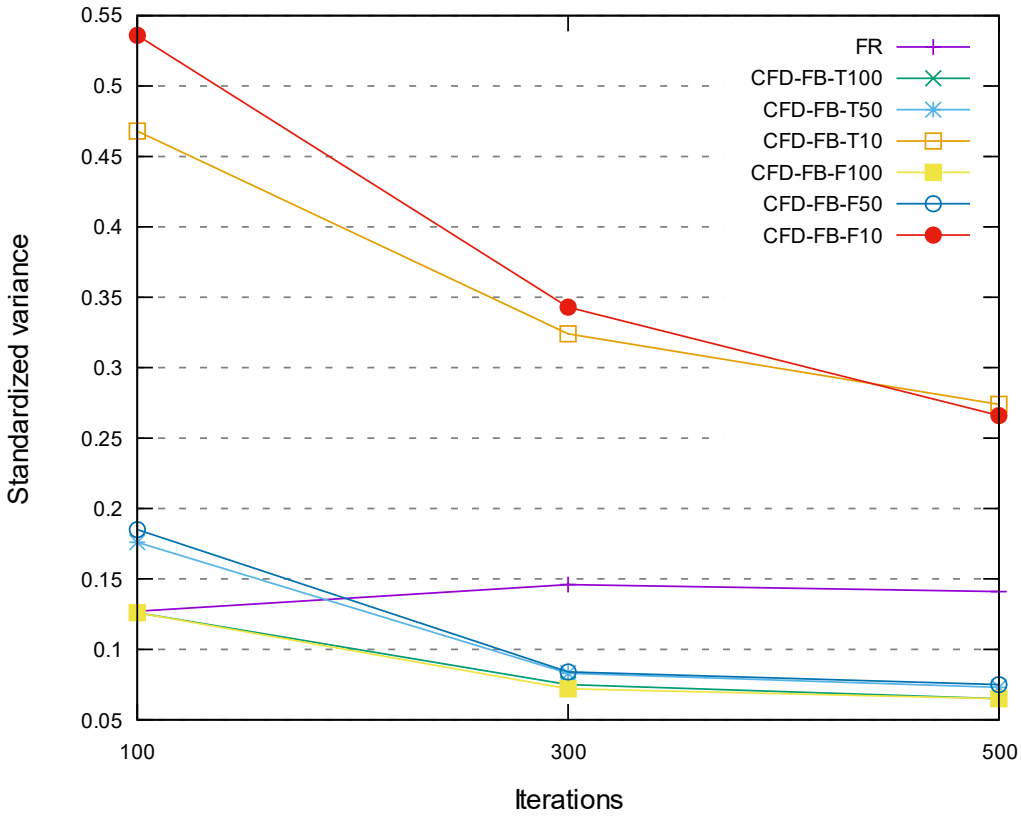


(c)

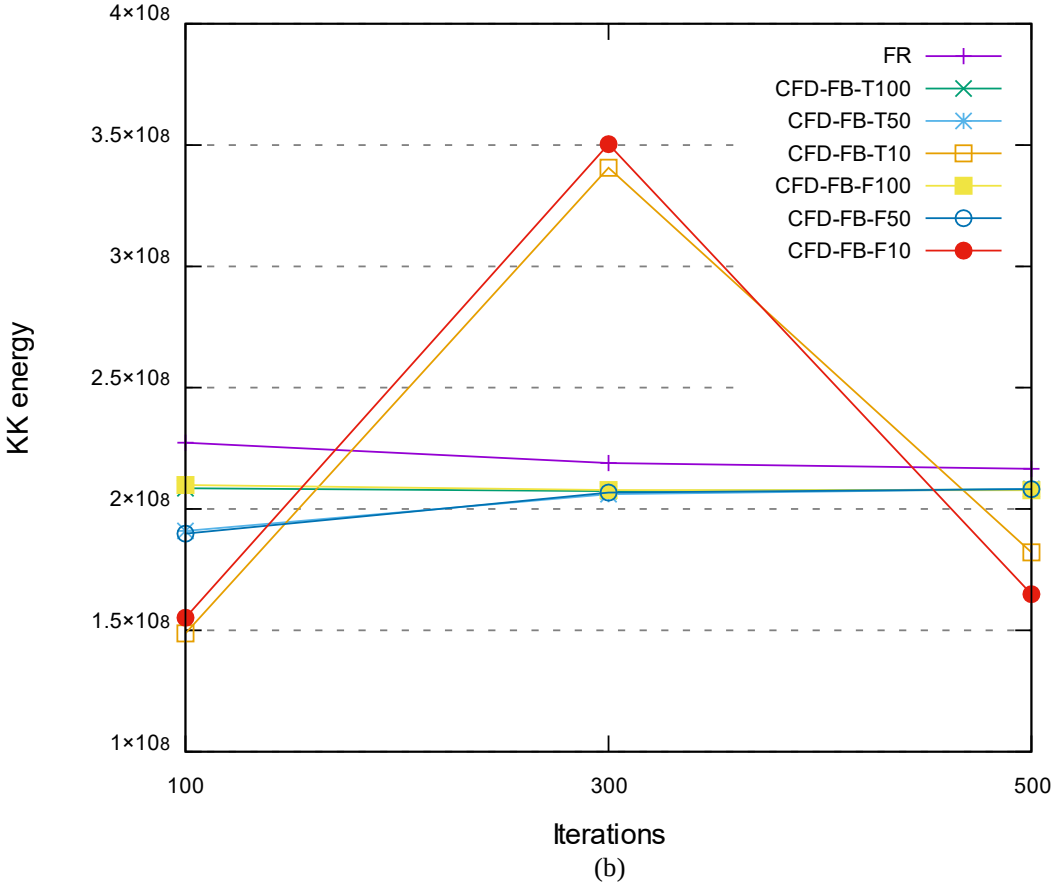


(d)

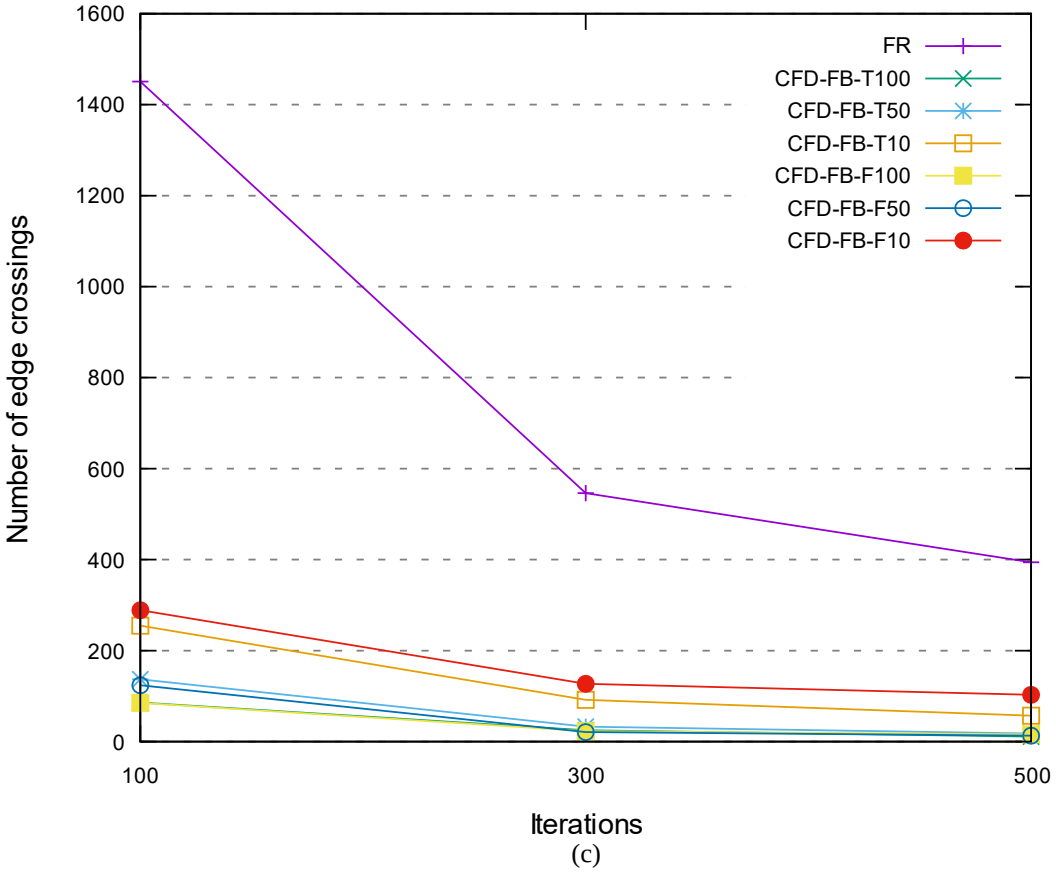
Figure 18: Comparison of KK, CFD-KB and CFD-KM with 30-minute time limit. (a) standardized variance (b) KK energy (c) edge crossings (d) total time usage.


Standardized variance
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
100
300
500
Iterations
FR
CFD-FB-T100
CFD-FB-T50
CFD-FB-T10
CFD-FB-F100
CFD-FB-F50
CFD-FB-F10

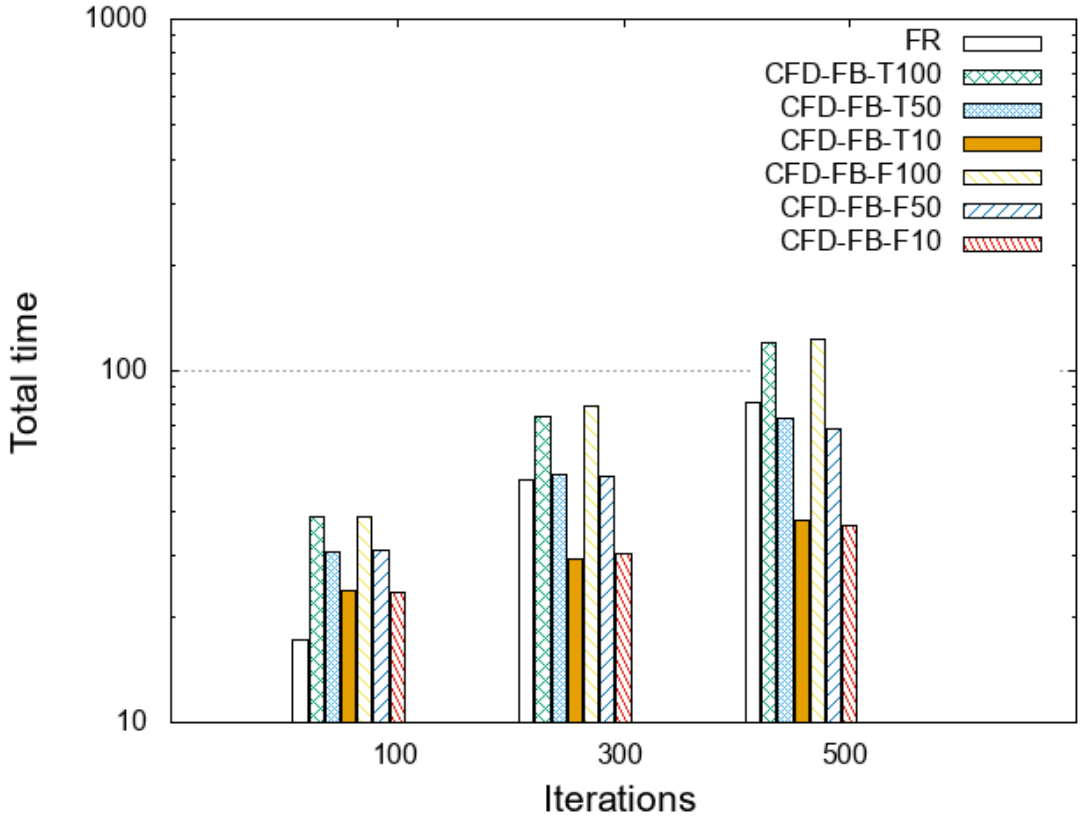
(a)


KK energy
4×10⁸
3.5×10⁸
3×10⁸
2.5×10⁸
2×10⁸
1.5×10⁸
1×10⁸
100
300
500
Iterations
FR
CFD-FB-T100
CFD-FB-T50
CFD-FB-T10
CFD-FB-F100
CFD-FB-F50
CFD-FB-F10

(b)

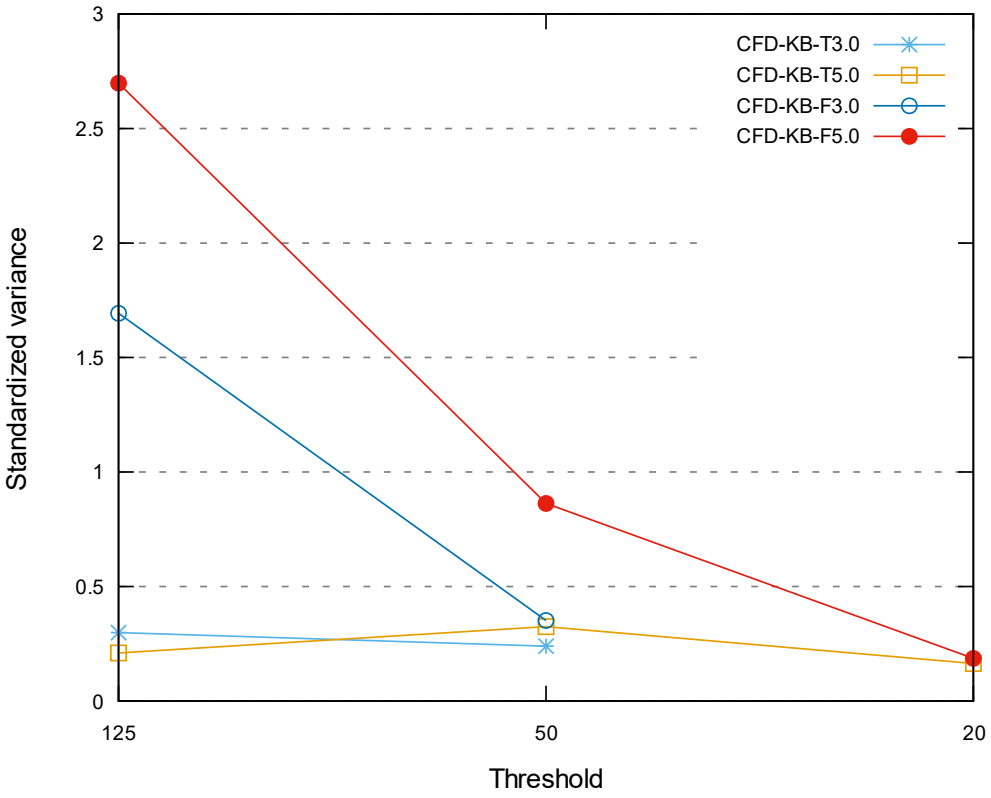


(c)

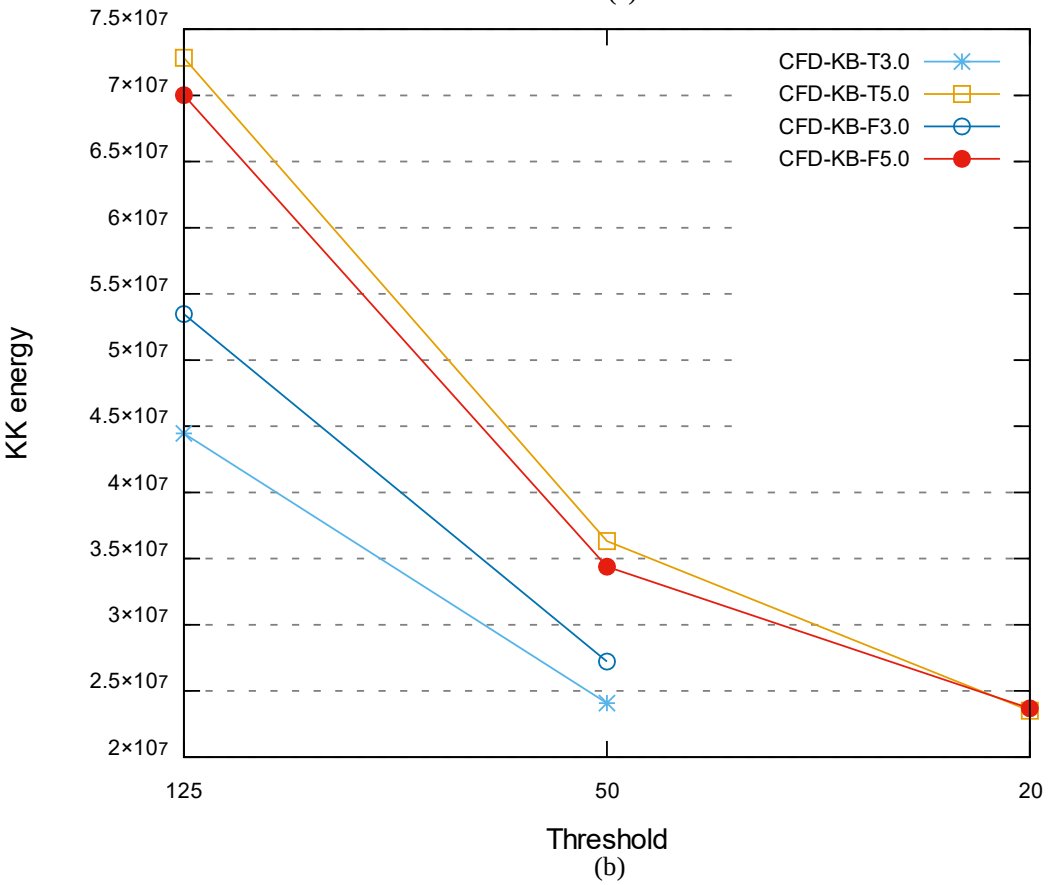


(d)

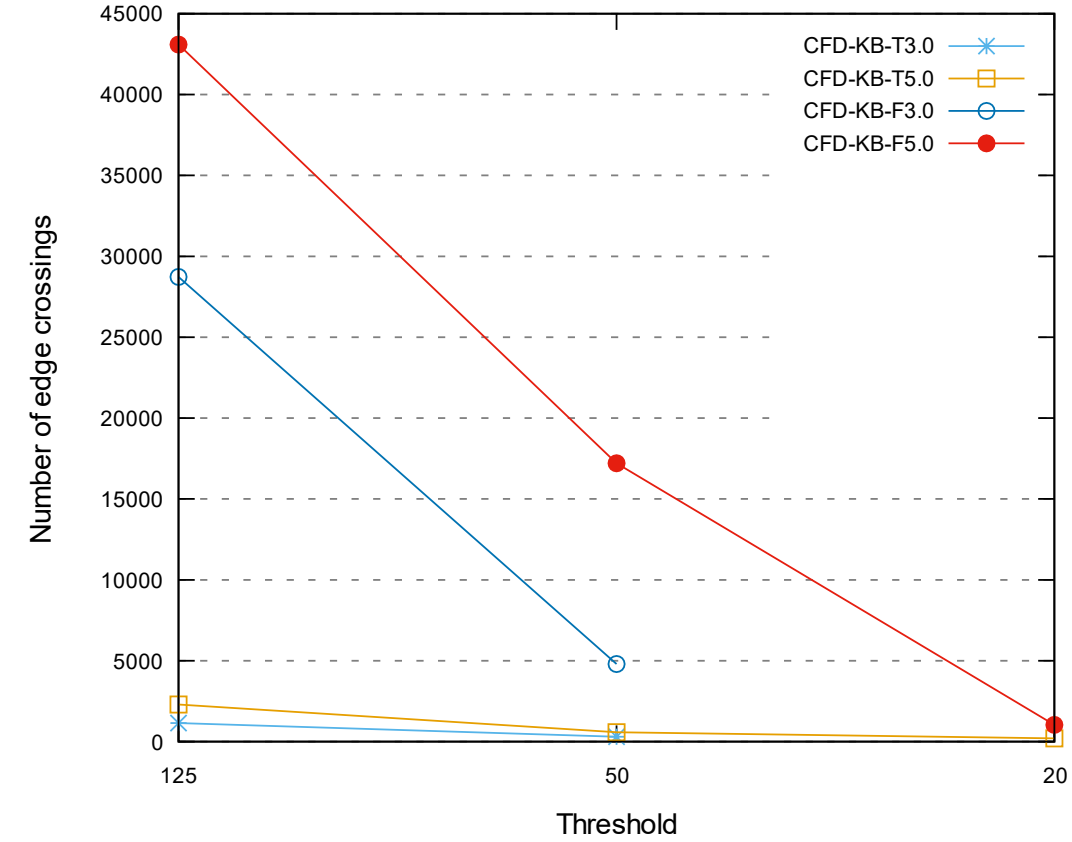
Figure 19: Comparison of FR, CFD-FB and CFD-FM with 60-minute time limit. (a) standardized variance (b) KK energy (c) edge crossings (d) total time usage.


CFD-KB-T3.0
CFD-KB-T5.0
CFD-KB-F3.0
CFD-KB-F5.0
Standardized variance
3
2.5
2
1.5
1
0.5
0
125
50
20
Threshold

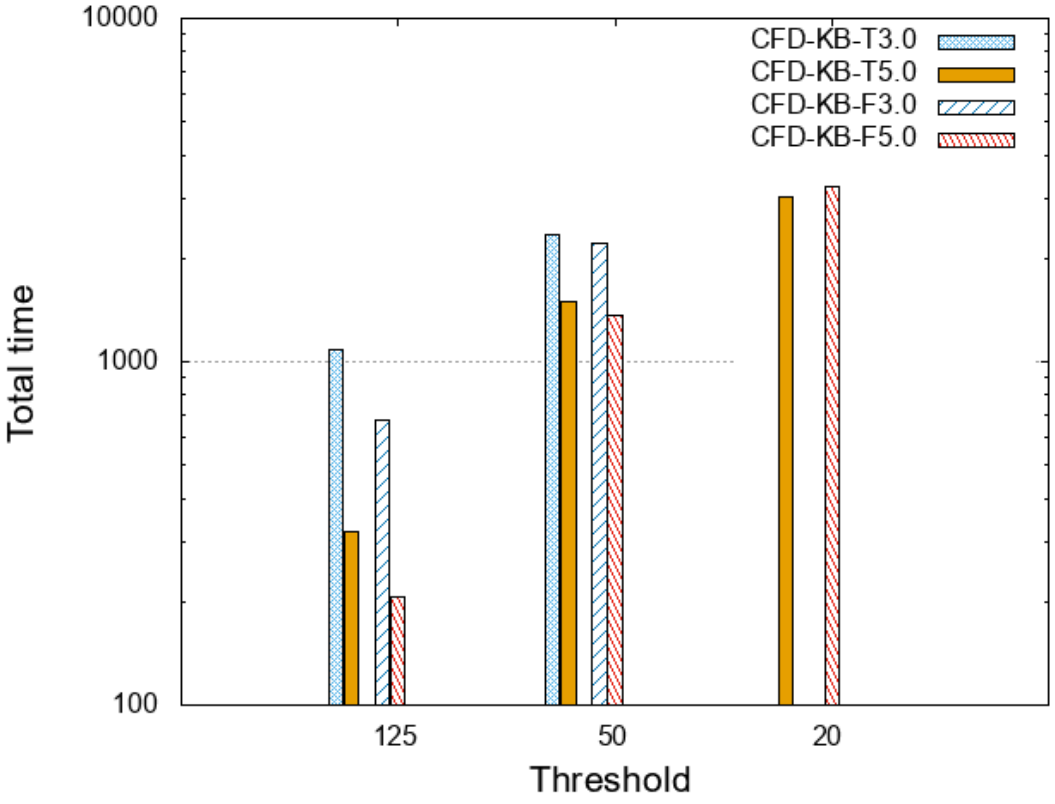
(a)


CFD-KB-T3.0
CFD-KB-T5.0
CFD-KB-F3.0
CFD-KB-F5.0
KK energy
7.5×10⁷
7×10⁷
6.5×10⁷
6×10⁷
5.5×10⁷
5×10⁷
4.5×10⁷
4×10⁷
3.5×10⁷
3×10⁷
2.5×10⁷
2×10⁷
125
50
20
Threshold

(b)



(c)



(d)

Figure 20: Comparison of KK, CFD-KB and CFD-KM with a 60-minute time limit. (a) standardized variance (b) KK energy (c) edge crossings (d) total time usage.

In the above experiments, we can observe that KK is effective in reducing KK energy. It is expected since the main objective of KK algorithm is to reduce the energy of the layout. We found that layouts generated by KK have 50% lower energy than those generated by FR. However, the computational cost of KK is high compared to FR. Moreover, KK algorithm can be trapped in a loop if the threshold ε is set too low. FR, on the other hand, can process large graphs more efficiently when it is compared to KK.

In summary, CFD-FB achieves the best overall performance compared to other algorithms. CFD-FB delivers the most significant improvement in reduction of edge crossings and CFD-FB is the most efficient in terms of total time consumed besides FR. Specifically, CFD-FB-T100 and CFD-FB-F100 achieve the best results for most cases except for KK energy.

5.1.2 Comparison on efficiency of the algorithms

In this experiment, we measure the execution time of algorithms for different settings. The execution of an algorithm will be stopped if the generated layout satisfies a specific criterion. In every execution, one of the three criteria (standardized variance, KK energy, and number of edge crossings) is selected in turn for each execution. In this experiment, we compare CFD-FB (iterations=100, 300, 500) and CFD-KB (threshold=125, 50, 20). The variables that are kept constant for this experiment are: Time limit (minutes) = 10, Community FDP = true, and Fine-tuning FR = 100 for FR or Fine-tuning threshold multiplier = 1.0 for KK.

In this experiment, the CFD algorithms are slightly modified for comparing the efficiency. For CFD-FB, fine-tuning step is not limited by the number of iterations; instead, the fine-tuning step of CFD-FB will be stopped if and only if it meets the stopping criterion. For CFD-KB, threshold ε (as discussed in section 4.3.2) which is used in fine-tuning step will be gradually decreased until the stopping criterion is met. The thresholds for each dataset are selected based on the result of the previous experiments to guarantee that each run can be terminated.

Dataset: PhD students in computer science (1025 nodes, 1043 edges) [34] (Stopping thresholds used for the experiments with this dataset: Standardized variance: 0.150, KK energy: 5000000.00, Number of edge crossings: 30)

The results of the experiments are given in Figure 21. CFD-KB is inefficient when it is compared to CFD-FB. Figure 21(a) shows that CFD-KB is least efficient in reducing the standardized variance compared with CFD-FB. CFD-KB is more effective in reducing KK energy and edge crossings. However, Figure 21(b) and Figure 21(c) show that CFD-KB is outperformed by CFD-FB in these 2 aspects in term of efficiency. Figure 21(c) shows that setting a low threshold can assist CFD-KB in reducing edge crossings more effectively.

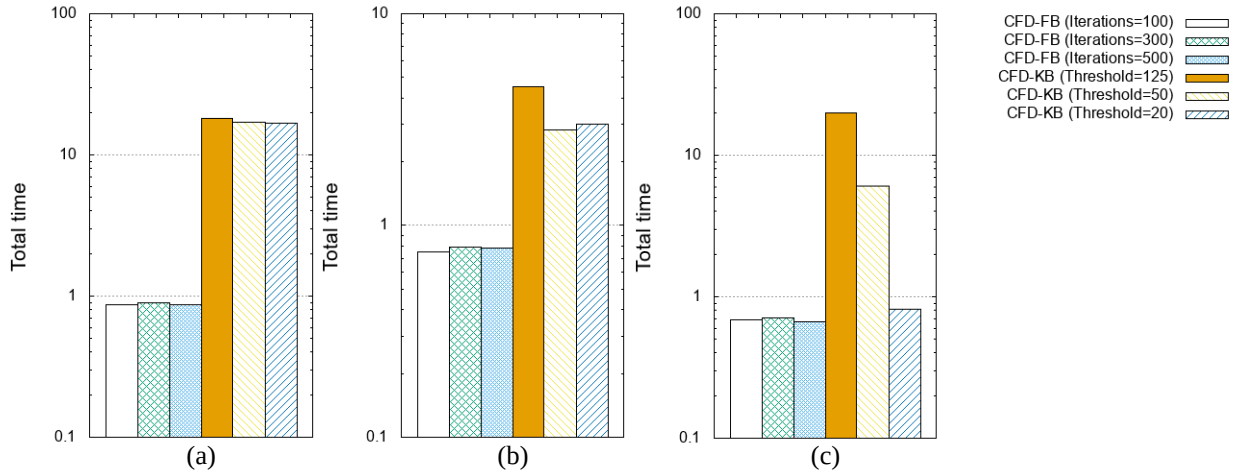


Figure 21: Comparison between CFD-FB and CFD-KB with respect to the time required to reach the stopping thresholds. (a) time required to reach stopping threshold for standardized variance (b) time required to reach stopping threshold for KK energy (c) time required to reach stopping threshold for the number of edge crossings.

Dataset: dolphins (62 nodes, 159 edges) [35] (Stopping thresholds used for the experiments with this dataset: Standardized variance: 0.150, KK energy: 10000.00, Number of edge crossings: 20)

In Figure 22, we can observe that for small graphs, the efficiency and effectiveness of CFD-KB is similar to CFD-FB when a threshold ε smaller than 50 is used in CFD-KB.

We will not discuss the experiment results for dataset "power grid" [36] because most runs by CFD-KB cannot be completed. We can conclude that CFD-KB is less efficient when it is compared to CFD-FB in producing layouts for large graphs.

From these results, we can conclude that CFD-FB is more efficient than CFD-KB in reaching stopping thresholds for every measurement except for small graph such as "dolphins" when a low threshold value is used in CFD-KB. Setting the number of iterations for CFD-FB to 300 results longer execution time for reaching the threshold. However, the additional time incurred is insignificant. According to Figure 22(a) and (b), CFD-FB is slightly more efficient if the number of iterations is set to 100, although this setting does not significantly affect the overall efficiency of CFD-FB.

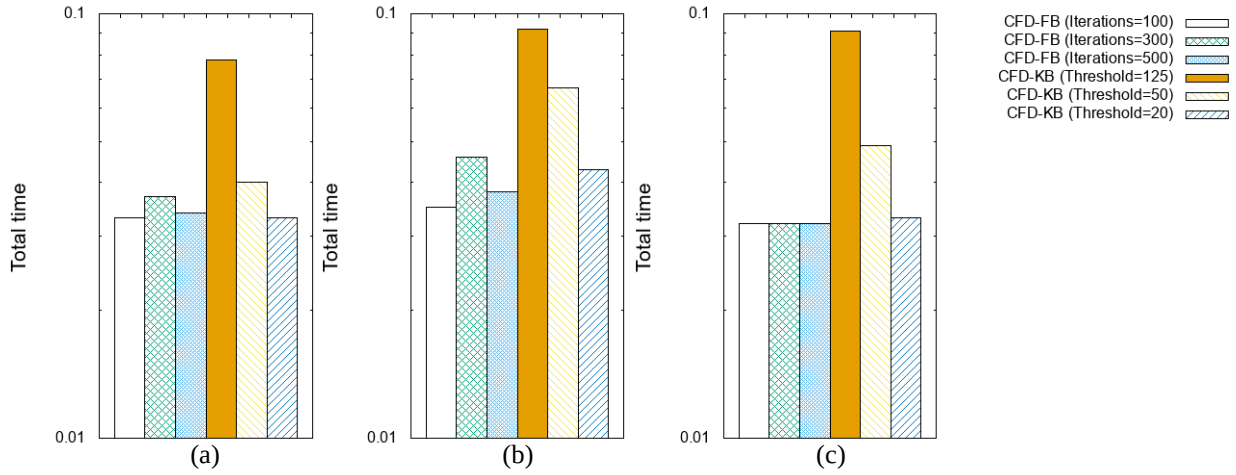


Figure 22: Comparison between CFD-FB and CFD-KB about time required to reach the stopping thresholds. (a) time required to reach stopping threshold for standardized variance (b) time required to reach stopping threshold for KK energy (c) time required to reach stopping threshold for number of edge crossings.

5.2 Visualizations

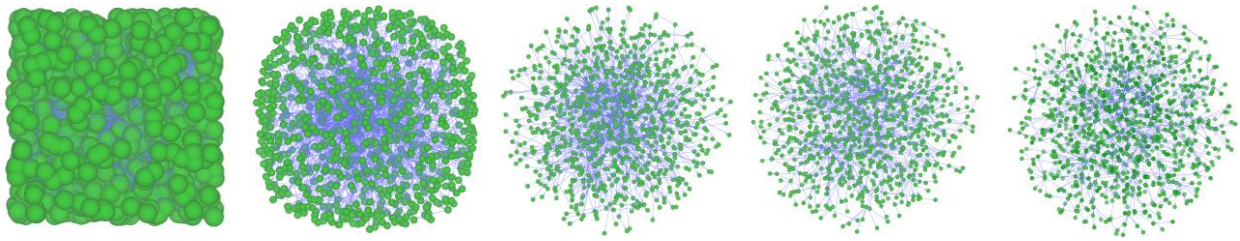
In this section, we compare the visualizations produced by FR and CFD-FB since the rest of the algorithms cannot process large graphs. The experiment settings are given in Table 3. The time limit for both algorithms is set to 10 minutes. We set the fine-tuning step to 100% since it can produce the best results in reducing the number of edge crossings. We use KiNG [30] to generate the visualizations in this experiment.

Table 3: Configuration for visualization

Algorithm	Time limit (minutes)	Number of Iterations	Threshold	Community FDP	Fine-tuning FR	Fine-tuning threshold multiplier
FR	10	100	-	-	-	-
CFD-FB	10	100	-	true	100	-

Dataset: PhD students in computer science (1025 nodes, 1043 edges) [34]:

Figure 23 illustrates the snapshots captured during the experiment. Since the generation of output snapshots also increase the complexity, we captured the snapshots from separate executions of the algorithms to ensure the fairness in time measurements. The results of evaluation for the snapshots shown in Figure 23 and Figure 24 are listed in Table 4 and Table 5.

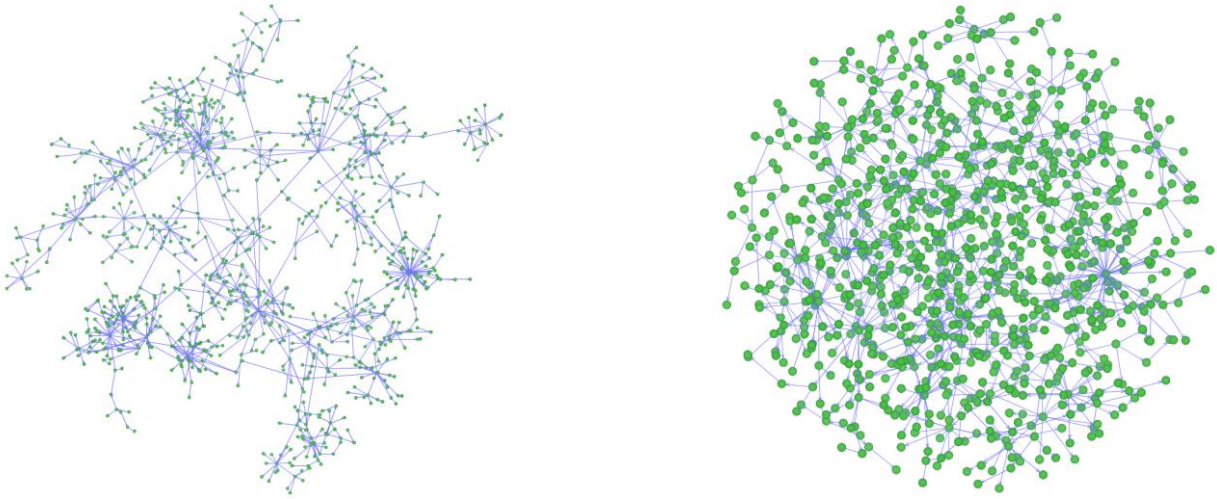


(a) (b) (c) (d) (e)

Figure 23: FR drawing snapshots: (a) layout at the beginning of the algorithm after random positioning of nodes (b) layout at the 10^{th} iteration (c) layout at the 30^{th} iteration (d) layout at the 50^{th} iteration (e) layout at the 100^{th} iteration.

Table 4: Evaluation of layouts from Figure 23

Layout	Standardized variance	KK energy	Number of edge crossings
a	0.140	19753314.23	3789
b	0.110	12545646.12	2451
c	0.064	9316208.57	248
d	0.100	8490673.00	68
e	0.127	8044936.55	32



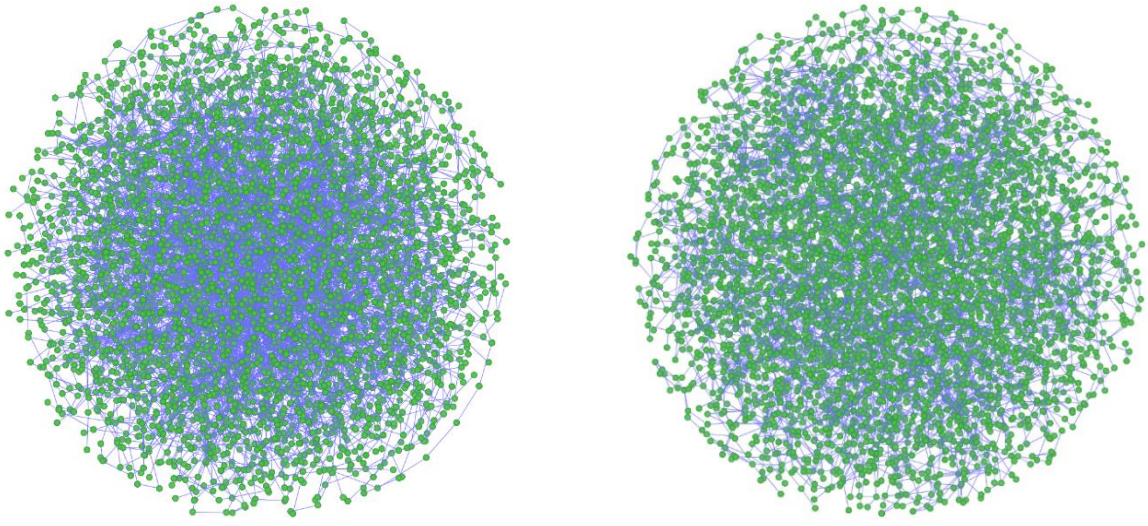
(a) (b)

Figure 24: CFD-FB drawing snapshots (a) layout after FR was applied to each cluster and these clusters were placed at their corresponding positions by weighted FR (b) layout after fine-tuning.

Table 5: Evaluation of layouts from Figure 24

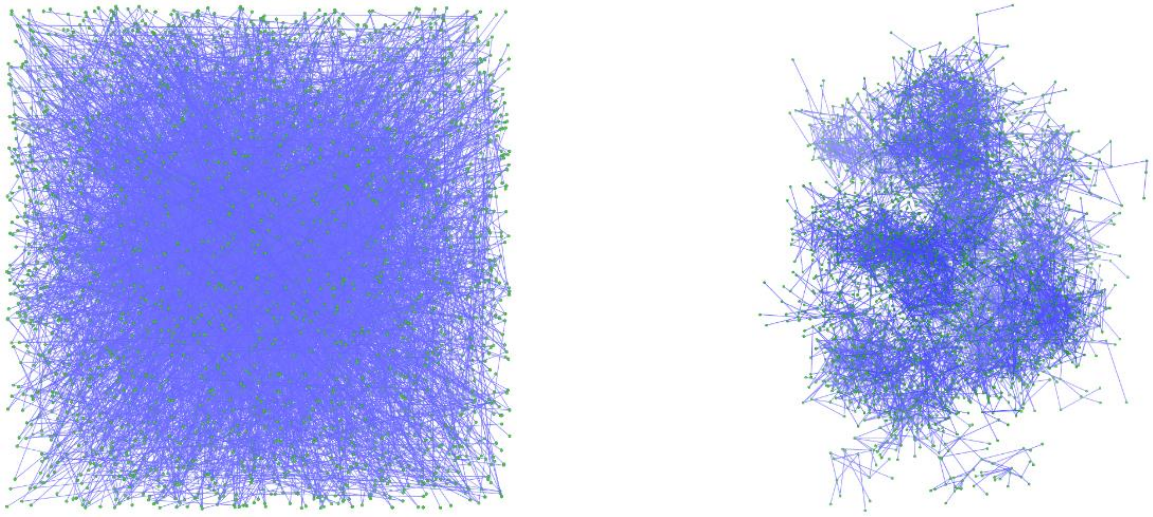
Layout	Standardized variance	KK energy	Number of edge crossings
a	0.485	12368768.33	8
b	0.077	7129640.45	2

Dataset: Power grid (4941 nodes, 6594 edges) [36]:



(a) (b)

Figure 25: (a) Layout generated by FR, Time limit (minutes) = 10, Iterations = 100 (b) Layout generated by CFD-FB, Time limit (minutes) = 10, Iterations = 100, Community FDP = true, Fine-tuning FR = 100



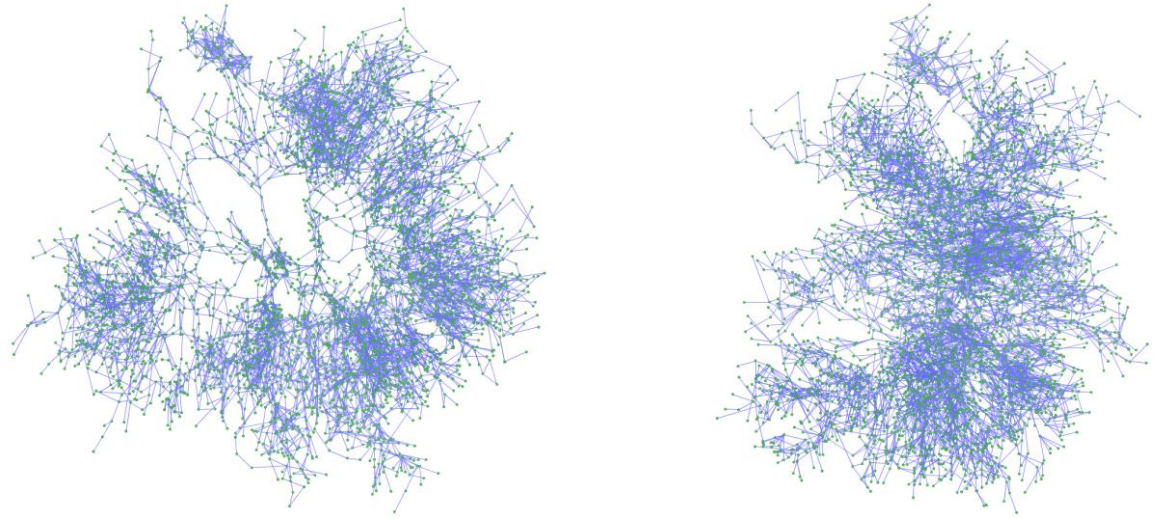
(a) (b)

Figure 26: (a) Layout generated by KK, Time limit (minutes) = 10, Threshold = 125.0 (b) Layout generated by CFD-KB, Time limit (minutes) = 10, Threshold = 125.0, Community FDP = true, Fine-tuning threshold multiplier = 1.0

In Figure 25 (a), a large number of edge crossings can be seen in the generated layout. The edge crossings are reduced in Figure 25 (b) by CFD-FB. The number of edge crossings resulted in FR is not high when dealing with small graphs (hundreds of nodes). However, the layout generated by FR can be cluttered if it contains thousands of nodes.

As for KK and CFD-KB, the execution of these algorithms cannot be completed within 10 minutes. The layouts were captured at the end of the 10 minutes time limit. Figure 26 (a) and Figure 26 (b) are the layouts generated by KK and CFD-KB at the end of the 10 minutes time limit. We can observe the layout from Figure 26 (b) contains clusters and the overall result is better than the layout from Figure 26 (a). We can also notice that fine-tuning was still in progress in Figure 26 (b).

Next, we compare the outputs of KK and CFD-KB when the time limit for execution is not enforced. Figure 27(a) is the layout generated by using KK algorithm after 6266.748 seconds. Figure 27(b) is the layout generated by using CFD-KB algorithm after 2985.188 seconds.



(a) (b)

Figure 27: (a) Layout generated by KK, Time consumed (minutes) = 104, Threshold = 125.0 (b) Layout generated by CFD-K0042, Time consumed (minutes) = 50, Threshold = 125.0, Community FDP = true, Fine-tuning threshold multiplier = 1.0

There is no visualization for CFD-FM and CFD-KM since clustering cannot be completed at the end of the 10-minute time limit at each run. This is caused by the fact that the computational cost of MCL ($O(N^2)$) is extremely high for the graphs containing thousands of nodes. KK algorithm is also not a practical algorithm for big graphs for the same reason. In summary, CFD-FB algorithm gives the best overall performance for big graphs.

6. Conclusion and future work

In this paper, we propose four novel Clustering-based Force-directed (CFD) algorithms for visualization of 3D graphs. CFD utilizes clustering algorithms to improve the quality of the graphs. In addition, weights are also introduced to further enhance the calculation for clusters. Extensive experiments were performed to evaluate the proposed CFD algorithms with three different datasets. The quality of output layouts is measured in terms of standardized variance, KK energy, and total number of edge crossings.

From the experiment results, we can observe that CFD-KB is able to reduce KK energy in its output graphs and it takes less time when it is compared to KK algorithm when handling large graph. However, it is less efficient when compared to CFD-FB. CFD-FM is more efficient than FR algorithm. However, it takes more time than FR when handling large graphs. CFD-KM is slightly better than KK algorithm in some of the cases that we have tested. Although CFD-KM is slightly faster than KK when handling small graphs, it is slower than KK algorithm when processing mid-sized graphs. Both CFD-KM and KK could not handle large graph within the allotted time.

Although BigCLAM and MCL use different approaches for clustering, their results are similar. However, the complexity of MCL is exponential whereas for BigCLAM, it is linear. Thus, MCL is not suitable for large graphs and it becomes slow when processing graphs containing approximately a thousand nodes. For large graphs, the quality of layouts generated by CFD-FB is much better than those generated by FR. According to the experiment results, CFD-FB achieves the best overall performance when handling large graphs. It significantly reduces edge crossings while keeping the computational cost low.

As for the future work, the proposed CFDs can be further extended for the visualization of communicates in social networks. By using a CFD algorithm, we can visualize potential communities in a social network if clusters in the layout representing that social network (graph) are colored differently. By using an effective coloring scheme, discovered clusters can be visualized in the 3D graphs of social networks. In addition, by choosing suitable viewing angles, occlusion in the visualization could also be reduced. One of the possibilities is the use of bird's eye view to improve the visualization.

Acknowledgement

This research was funded by the University of Macau, under grants MYRG2019-00136-FST and MYRG2018-00246-FST.

References

[1] L. C. Freeman, "Methods of Social Network Visualization," in *Computational Complexity: Theory, Techniques, and Applications*, R. A. Meyers, Ed. New York, NY: Springer New York, 2012, pp. 2981-2998.

[2] M. Junger and P. Mutzel, *Graph Drawing Software*. Springer-Verlag, 2003.

[3] N. Quinn and M. Breuer, "A Forced Directed Component Placement Procedure for Printed Circuit Boards," *IEEE Transactions on Circuits and Systems*, vol. 26, no. 6, pp. 377-388, 1979.
[4] T. M. J. Fruchterman and E. M. Reingold, "Graph Drawing by Force-directed Placement," *Software: Practice and Experience*, vol. 21, pp. 1129-1164, 1991.
[5] P. Eades, "A Heuristic for Graph Drawing," *Congressus Numerantium*, vol. 42, pp. 149-160, 1984.
[6] T. Kamada and S. Kawai, "An Aalgorithm for Drawing General Undirected Graphs," *Information Processing Letters*, vol. 31, no. 1, pp. 7-15, 1989.
[7] U. Brandes, N. Indlekofer, and M. Mader, "Visualization Methods for Longitudinal Social Networks and Stochastic Actor-oriented Modeling," *Social Networks*, vol. 34, no. 3, pp. 291-308, 2012.
[8] S. Brasch, G. Fuellen, and L. Linsen, "VENLO: Interactive Visual Exploration of Aligned Biological Networks and Their Evolution," Berlin, Heidelberg, 2012, pp. 229-247: Springer Berlin Heidelberg.
[9] J. Yang and J. Leskovec, "Overlapping Community Detection at Scale: a Nonnegative Matrix Factorization Approach," in *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining*, Rome, Italy, 2013, pp. 587-596: ACM.
[10] S. M. V. Dongen, "Graph Clustering by Flow Simulation," Utrecht University, 2001.
[11] A. Frick, A. Ludwig, and H. Mehldau, "A Fast Adaptive Layout Algorithm for Undirected Graphs (Extended Abstract and System Demonstration)," Berlin, Heidelberg, 1995, pp. 388-403: Springer Berlin Heidelberg.
[12] M. J. Bannister, D. Eppstein, M. T. Goodrich, and L. Trott, "Force-Directed Graph Drawing Using Social Gravity and Scaling," in *Graph Drawing: 20th International Symposium, GD 2012, Redmond, WA, USA, September 19-21, 2012, Revised Selected Papers*, W. Didimo and M. Patrignani, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 414-425.
[13] R. Davidson and D. Harel, "Drawing Graphs Nicely Using Simulated Annealing," *ACM Transactions on Graphics*, vol. 15, pp. 301-331, 1996.
[14] M. Jacomy, T. Venturini, S. Heymann, and M. Bastian, "ForceAtlas2, a Continuous Graph Layout Algorithm for Handy Network Visualization Designed for the Gephi Software," *Plos One*, vol. 9, no. 6, 2014.
[15] A. Noack, "An Energy Model for Visual Graph Clustering," in *Graph Drawing: 11th International Symposium, GD 2003 Perugia, Italy, September 21-24, 2003 Revised Papers*, G. Liotta, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 425-436.
[16] C. Walshaw, "A Multilevel Algorithm for Force-directed Graph Drawing," in *Graph Drawing*, 2000, vol. 1984, pp. 171-182.
[17] Y. Hu, "Efficient, High-Quality Force-Directed Graph Drawing," *The Mathematica Journal* vol. 10, no. 1, pp. 37-71, 2006.
[18] C.-C. Lin and H.-C. Yen, "A New Force-directed Graph Drawing Method Based on Edge-edge Repulsion," *Journal of Visual Languages and Computing*, vol. 23, no. 1, pp. 29-42, 2012.
[19] A. Arleo, W. Didimo, G. Liotta, and F. Montecchiani, "A Distributed Multilevel Force-Directed Algorithm," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 4, pp. 754-765, 2019.
[20] A. Suh, M. Hajij, B. Wang, C. Scheidegger, and P. Rosen, "Persistent Homology Guided Force-Directed Graph Layouts," *IEEE Transactions on Visualization and Computer Graphics*, vol. 26, no. 1, pp. 697-707, 2020.
[21] P. E. Seok-Hee Hong, Marnijati Torkel, Ziyang Wang, David Chae, Sungpack Hong, Daniel Langerenken, Hassan Chafi, "Multi-level Graph Drawing Using Infomap Clustering," in *27th International Symposium, GD 2019*, Prague, Czech Republic, pp. 139-146: Springer Nature.
[22] M. Rosvall and C. T. Bergstrom, "Maps of Random Walks on Complex Networks Reveal Community Structure," *Proceedings of the National Academy of Sciences*, vol. 105, no. 4, p. 23, 2008.
[23] C. Walshaw, "A Multilevel Algorithm for Force-Directed Graph Drawing," Berlin, Heidelberg, 2001, pp. 171-182: Springer Berlin Heidelberg.
[24] Y. Wang, Z. Jin, Q. Wang, W. Cui, T. Ma, and H. Qu, "DeepDrawing: A Deep Learning Approach to Graph Drawing," *IEEE Transactions on Visualization and Computer Graphics*, vol. 26, no. 1, pp. 676-686, 2020.
[25] R. Gove, "A Random Sampling O(n) Force-calculation Algorithm for Graph Layouts," *Computer Graphics Forum*, vol. 38, no. 3, pp. 739-751, 2019.

[26] J. Barnes and P. Hut, "A Hierarchical O(N log N) Force-calculation Algorithm," *Nature*, vol. 324, no. 6096, pp. 446-449, 1986.
[27] D. H. Eberly, "Chapter 14 - Distance Methods," in *3D Game Engine Design* 2 ed. San Francisco: Morgan Kaufmann, 2007, pp. 639-679.
[28] R. W. Floyd, "Algorithm 97: Shortest Path," *Commun. ACM*, vol. 5, no. 6, p. 345, 1962.
[29] M. Himsolt, "GML: A Portable Graph File Format," Available: https://pdfs.semanticscholar.org/d0a5/6b07a59a29b48d6f957763add90e05925c2c.pdf?_ga=2.114878407.1985112014.1581063714-1167339558.1581063714
[30] *KiNG Display Software*. Available: http://kinemage.biochem.duke.edu/software/king.php
[31] A. D. King, "Graph Clustering with Restricted Neighbourhood Search," Graduate Department of Computer Science, University of Toronto, 2004.
[32] T. Kamada and S. Kawai, "An Algorithm For Drawing General Undirected Graphs," *Information Processing Letters*, vol. 31, no. 1, pp. 7-15, 1989.
[33] A. Ben-Israel, "A Newton-Raphson Method for the Solution of Systems of Equations," *Journal of Mathematical Analysis and Applications*, vol. 15, no. 2, pp. 243-252, 1966.
[34] *PhD Students in Computer Science, Pajek Dataset*. Available: http://vlado.fmf.uni-lj.si/pub/networks/data/esna/CSPhD.htm
[35] *Dolphins GML dataset*. Available: http://networkdata.ics.uci.edu/data/dolphins/dolphins.gml
[36] *Power grid dataset*. Available: http://www-personal.umich.edu/~mejn/netdata/power.zip