An Efficient K-Way Graph Partitioning Algorithm for Task Allocation in Parallel Computing Systems

Cheol H. Lee, M. Kim

Department of Electrical Engineering
KAIST
P.O. Box 150, Seoul 130-650, Korea.

Chan I. Park

Department of Computer Science
POSTECH
P.O. Box 125, Pohang 790-600, Korea.

Abstract

Graph partitioning is an important example of a class of combinatorial optimization problems. Especially, we consider k-way graph partitioning: given a graph $G(V, E)$ with weights on its edges and sizes on its nodes, partition the nodes of G into k subsets of the equal size such that the the weight sum of edges connecting different subsets is minimized. It is shown that we can transform the graph partitioning problem into max-cut problem by incorporating node size information into edge weight. After transformation, we can devise a very simple cost function which make our algorithm more efficient than Kernighan-Lin's (K-L's) algorithm. The computing time per iteration of the algorithm is $O(k \cdot N^2)$, where N is the number of nodes in the given graph. Experimental results show that the proposed algorithm outperforms K-L's algorithm both in the quality of solutions and in the elapsed time. It is also shown that as the difference among sizes of the nodes increases, the performance gap between ours and K-L's becomes larger.

1 Introduction

In this paper we deal with the following combinatorial problem: given a graph $G(V, E)$ with costs on its edges and sizes on its nodes, partition the nodes of G into k subsets of the equal size such that the weight sum of edge cut is minimized. This problem is called uniform k-way partitioning problem.

The uniform k-way partitioning problem in general is known to be NP-hard. Nevertheless, we need a fast heuristic algorithm because it does have numerous applications, particularly in clustering-related setups, such as VLSI cell placement [2] and task allocation [3,10]. The edge weights represent the similarity between each pair of objects (represented by nodes with the weights representing the object size). The k clusters of the equal size that maximize the similarity between objects in same clusters or alternatively maximize the dissimilarity between objects in separate clusters is identical to the uniform k-way partitioning problem. Conventionally, a famous heuristic algorithm called Kernighan-Lin's algorithm [1] (from now on we denote it as K-L's algorithm) has been used to solve the uniform partitioning problem of $k = 2$. However, K-L's algorithm has three difficulties. First, the equality in size is maintained during the execution of K-L's algorithm by using so-called *pairwise-exchange* scheme to transform an existing partition. Thus, once initial partition is constructed satisfying the size constraint, all subsequent partitions also meet the size constraint, not to speak of the final partition. Using pairwise-exchange scheme makes the time complexity of K-L's algorithm greater than $O(N^2 \log_2 N)$, where N is the number of nodes in the graph. Second, the efficiency of K-L's algorithm decreases when the nodes of a graph are not all of the same size, because any node of size $p > 1$ is converted to a cluster of p nodes of size 1, connected by edges of appropriately high cost. So it is necessary to sacrifice some accuracy to keep the number of generating nodes within reasonable bounds when the amount of size differences among nodes is large. Third, K-L's method is highly oriented to 2-way partitions of graphs. In the reference [1], Kernighan and Lin proposed the method for k-way partitions of $k \geq 3$ that K-L's algorithm is applied several times to each pair of k subsets to reach global optimality through pairwise optimality. However, the efficiency of K-L's algorithm decreases with the value of k, since the process of pairwise optimality can disturb each other during the execution of K-L's algorithm on each pair of k subsets.

Several authors have suggested some improved heuristic algorithms based on the basic idea of K-L's method. Fiduccia and Mattheyses [4] suggested a fast heuristic for improving min-cut partitions of VLSI networks by using efficient data structures. The computing time of it, per pass, grows linearly with the total number of pins in the network. This linear-time behavior is achieved by using a process of moving one element at a time as the basic technique to transform an existing partition. We call the process *one-move*. The idea of one-move came originally from the reference [8]. Krishnamurthy [5] improved the method in the reference [4] by using more sophisticated heuristics. Goldberg and Burstein [6], and Ng *et. al.* [7] proposed another improvements for the method in the reference [4] by contracting an input graph before partitioning, thus reducing the number of nodes in the graph. However, the three difficulties in K-L's algorithm have not been completely overcome because all improvements are made for the 2-way partitioning problem in VLSI networks only.

Equal-sized partition is generalized into *balanced* partition since equality cannot be achieved in general. A partition is called *balanced* when the total sum of size differences of all pairs of subsets is minimum. Especially, a uniform partition is obtained when the total sum of size differences is zero. Then, the k-way partitioning problem is restated as: find a balanced partition such that the total cost of the edges cut is minimized.

In this paper, we propose an efficient method for the k-way partitioning problem which deals with the three difficulties in K-L's method by problem transformation. To begin with, it is shown that the k-way partitioning problem can be transformed into the problem of finding a maximum k-cut of a graph. Using the prob-

TH0309-5/90/0000/0748$01.00 © 1990 IEEE

lem transformation, we can devise efficient heuristics which deal with nodes of various sizes without any performance degradation. Then we present an efficient heuristic algorithm for finding a maximum k-cut, i.e., finding a balanced k-way partition with the minimum cost. The algorithm is based on K-L's method except the fact that it progresses by moving one element at a time between the subsets of the partition.

A technique of problem transformation is given in Section II which serves as a framework for the design of efficient heuristics. We describe a heuristic algorithm for the 2-way partitioning problem in Section III and its extension to solve the k-way partitioning problem in Section IV. Section V gives an illustrative example which shows the effectiveness of the algorithm. Experimental results and the comparison with K-L's algorithm are given in Section VI.

2 Problem Transformation

Consider a graph $G(V, E)$ with N nodes (v_1, v_2, ..., v_N) and E edges (e_1, e_2, ..., e_E). Let the size of v_i be denoted by $S(v_i)$ and the cost of e_j by $W(e_j)$. The size of each node and the cost of each edge are assumed to be positive integer values in this paper. The size of a set is equal to the sum of the sizes of all nodes in the set. A k-cut of G is a subset of edges that partitions the nodes of G into k disjoint subsets. The cost of a k-cut is equal to the sum of the costs of all edges in the cut. A cut is called *balanced* when the associated partition is balanced.

In the original graph, $G(V, E)$, each cut has no information on the size of the resulting subsets. Thus it is difficult to develop efficient heuristics on the original graph, since one must consider the balancing of an existing partition as well as the cost of the partition. K-L's algorithm, for example, keeps an existing partition uniform by using pairwise-exchange given an initial uniform partition. Thus, the pairwise-exchange scheme makes K-L's algorithm efficient only for the case that all the nodes of the graph are of the same size. If each edge has the information on the size of the associated nodes not to speak of the cost of the edge, it becomes easy to devise an efficient heuristics because we only pay attention to the cost of a cut, not to the size of the corresponding subsets.

We modify the original graph, G, such that any cut on the resulting graph has the information on the size of each subset. The scheme for graph modification is as follows. For any two nodes v_i and v_r of G,

1. if there is an edge e_j between them, modify the cost of e_j to be $S(v_i) \cdot S(v_r) \cdot R - W(e_j)$, where the augmenting factor R is set to an appropriately large positive value as described later.

2. otherwise, make a new edge between them which has the cost of $S(v_i) \cdot S(v_r) \cdot R$.

The resulting graph is denoted by $\hat{G}(\hat{V}, \hat{E})$. It can be easily seen that $V = \hat{V}$ and $|\hat{E}| = |V|(|V| - 1)/2$ (i.e., $\hat{G}$ is a complete graph with the same number of nodes as in G). For example, if the original graph G in Figure 1 (a) is given, the graph $\hat{G}$ is obtained from modifying G as shown in Figure 1 (b), where the number specified in each circle denotes the size of the corresponding node. The augmenting factor R is set to 20 in this case.

A k-cut of a graph is a subset of edges that partitions the nodes of the graph into k disjoint subsets. Then the k-way partitioning problem is the problem of finding a minimum-cost balanced k-cut. This problem can also be transformed into the max-cut problem using the technique of graph modification described above.

Lemma 1 Given a graph G and its modified graph $\hat{G}$, let $\hat{C}$ (C) partition the nodes of $\hat{G}$ (G) into k subsets $P_1, P_2, \cdots, P_k$. Then, $W(\hat{C}) = \sum_{i<j} S(P_i) \cdot S(P_j) \cdot R - W(C)$, for $1 \le i < j \le k$.

Proof : Let each subset P_i be $\{v_{i_l} | 1 \le l \le p_i\}$. Then,

$$\begin{cases} W(\hat{C}) & = \frac{1}{2} \cdot \sum_{\substack{1 \le i \le k \\ 1 \le j \le k}} \left(\sum_{\substack{1 \le l \le p_i \\ 1 \le m \le p_j}} \left(S(v_{i_l}) \cdot S(v_{j_m}) \cdot R - W\left((v_{i_l}, v_{j_m})\right) \right) \right) \\ & = \frac{1}{2} \cdot \sum_{\substack{1 \le i \le k \\ 1 \le j \le k}} \left(S(P_i) \cdot S(P_j) \cdot R - \sum_{\substack{1 \le l \le p_i \\ 1 \le m \le p_j}} W\left((v_{i_l}, v_{j_m})\right) \right) \\ & = \sum_{1 \le i < j \le k} S(P_i) \cdot S(P_j) \cdot R - W(C) \end{cases}$$

□

Theorem 1 A maximum k-cut on $\hat{G}$ corresponds to a minimum-cost balanced k-cut on G, if $R > \sum_{i=1}^{|E|} W(e_i)$.

Proof : Let a maximum k-cut $\hat{C}$, partition the nodes of $\hat{G}$ into k subsets $P_1, P_2, \cdots, P_k$. Let the corresponding k-cut on G to $\hat{C}$ on $\hat{G}$ be C. Assume that the partition $(P_1, P_2, \cdots, P_k)$ is not balanced. Let cut C_o(C_o) partition the nodes of $\hat{G}$(G) into k balanced subsets $P_{1o}, P_{2o}, \cdots, P_{ko}$. Then, $\sum_{i<j} S(P_{io}) \cdot S(P_{jo}) \ge \sum_{i<j} S(P_i) \cdot S(P_j) + p$ for some positive integer p, since $\sum_{i<j} |S(P_{io}) - S(P_{jo})| < \sum_{i<j} |S(P_i) - S(P_j)|$ and $\sum_{i=1}^{k} S(P_{io}) = \sum_{i=1}^{k} S(P_i) =$ constant. $\hat{C}$ is a maximum cut, thus

$$\begin{cases} \sum_{i<j} S(P_i) \cdot S(P_j) \cdot R - W(C) & \ge \sum_{i<j} S(P_{io}) \cdot S(P_{jo}) \cdot R - W(C_o) \\ & \ge \sum_{i<j} S(P_i) \cdot S(P_j) \cdot R + p \cdot R - W(C_o) \end{cases}$$

Then, $W(C_o) - W(C) \ge p \cdot R$. This is contradictory to that $R > \sum_{i=1}^{|E|} W(e_i)$. Therefore, $(P_1, P_2, \cdots, P_k)$ is a balanced partition. In order for $W(\hat{C})$ ($= \sum_{i<j} S(P_i) \cdot S(P_j) \cdot R - W(C)$) to be maximum, $W(C)$ must be minimum among the costs of balanced k-cuts since $\sum_{i<j} S(P_i) \cdot S(P_j) \cdot R$ is constant for all balanced k-cuts. This proves the theorem. □

By Theorem 1 and Lemma 1, the k-way partitioning problem on G can be transformed into the problem of finding a maximum k-cut on $\hat{G}$ irrespectively of the value of k. In other words, if we transform the partitioning problem into the max-cut problem, we need not try to keep the resulting partition balanced. Consequently, we can devise a heuristic algorithm of which the performance is insensitive to the size value of each node, whereas that of K-L's algorithm is highly dependent.

3 K-way Partitions

Given a graph $G(V, E)$, first we obtain $\hat{G}(V, \hat{E})$ by the method described in the previous section. Then, the problem is to partition V into k subsets $V_1, V_2, \ldots, V_k$, so as to maximize the cost of the k-cut, $W(\hat{C}_k)$, where

$$W(\hat{C}_k) = \sum_{i<j} \sum_{\substack{v_a \in V_i \\ v_b \in V_j}} \hat{W}(a, b).$$

The basic approach is to start with an arbitrary partition and to improve it by iteratively choosing one node from one subset and moving it to another. The node to be moved is chosen so that a maximum increase in cut cost may be obtained (or minimum decrease if no increase is possible). The algorithm consists of a series of passes: in each pass, one node is moved in turn until all

$|V|$ nodes have been moved. At each iteration, the nodes to be moved are choosen from among the ones that have not yet been moved during the pass. The $|V|$ partitions produced during the pass are examined and the one with the maximum cut is chosen as the starting partition for the next pass. Passes are performed until no improvement in cut cost can be obtained. This optimization technique is a simple modification of Kernighan-Lin's [1].

We define the *gain* $g(v_r, V_j)$ of v_r in $V_i (i \neq j)$ over V_j as the amount of cost by which cut cost would increase if v_r is moved from V_i to V_j, *i.e.*

$$\left\{ g(v_r, V_j) = \sum_{v_a \in V_i} \hat{W}(r,a) \; - \; \sum_{v_b \in V_j} \hat{W}(r,b), \quad \text{for all } v_r \in V_i \right.$$

Among those that have not yet been moved during the pass, a node with the largest gain is selected as the next one to be moved. For example, if the gain of v_r in V_i over V_j, $g(v_r, V_j)$, is maximum, v_r will be moved from V_i to V_j. After moving it, the gains of all the other nodes are recalculated as follows:

$$\begin{cases} \hat{g}(v_a, V_j) & = g(v_a, V_j) - 2\hat{W}(a,r), \text{forall} v_a \in V_i, \\ \hat{g}(v_b, V_i) & = g(v_b, V_i) + 2\hat{W}(b,r), \text{forall} v_b \in V_j, \\ \hat{g}(v_c, V_i) & = g(v_c, V_i) + \hat{W}(c,r), \text{forall} v_c \in V_l, 1 \leq l(\neq i,j) \leq k, \\ \hat{g}(v_c, V_j) & = g(v_c, V_j) - \hat{W}(c,r), \text{forall} v_c \in V_l, 1 \leq l(\neq i,j) \leq k \end{cases}$$

The algorithm called **PARTITION** for the maximum k-cut problem is described in Figure 2. Especially, $|V|$-tuple implemented in the form of an array PART[1..|V|] is used to describe current partition. HISTORY[1..|V|][1..2] is used to save the history information on move operations.

For time analysis, we define a *pass* to be the operation involved in making one cycle from step 2 to step 5 of the algorithm **PARTITION**. The computing time needed for step 2 is $O(k|V|^2)$ since we need $O(|V|)$ time to compute the gain of each element over each subset. Each iteration of step 3 needs $O(k|V|)$ computing time mainly due to the selection of an element of the largest gain. Thus, the total time needed for step 3 is $O(k|V|^2)$. The computing time of $O(|V|)$ is sufficient for step 4 and 5. Therefore, the total computing time for a pass is $O(k|V|^2)$.

4 Experimental results

The experiments were performed under UNIX BSD4.2 running on a SUN3/50. The heuristic solution approximates the combination of the "balance" and the cost of the k-cut, *i.e.* $\sum_{i<j} S(V_i) \cdot S(V_j) \cdot R - W(C_k)$. The combination factor R is determined experimentally with more emphasis on the "balance" condition. We have run our algorithm on a large number of random graphs with up to 200 nodes of sizes varing from 1 to 20. Our algorithm has almost always found balanced partitions (about 99.5 percentages) by setting R to $\max_i\left(\sum_{j=1}^{|V|} W(i,j)/2S(v_i)\right)$. Even in the cases where the algorithm has failed to find balanced partitions, the resulting partitions were sufficiently close to being balanced ($\max_{i,j}(S(V_i) - S(V_j)) = 1$). On the basis of this experimental evidence, R is set to $\max_i\left(\sum_{j=1}^{|V|} W(i,j)/2S(v_i)\right)$.

The experiments in cases of k = 2,4, and 10 were performed to compare our algorithm with Kernighan-Lin's algorithm for the k-way graph partitioning problem. For each experiment, 100 random graphs were generated with 100 nodes and about 1500 edges. Each experiment is divided into three cases: case 1) all the nodes are of the same size of 1, case 2) the sizes of nodes are ranging from 1 to 3, case 3) the sizes of nodes are ranging from 1 to 6. For all generated graphs, the weight of each edge was set to one in order to evaluate the performance with clarity from the viewpoint of the number of edges in the final cut.

On all graphs tested, both of our and Kernighan-Lin's algorithm have always found balanced partitions. The major performance measure, therefore, is the cost of the resulting cuts. Table 1, 2, and 3 summarize the results of the experiments for k = 2, 4, and 10. The results show that our algorithm outperforms Kernighan-Lin's in the quality of solutions even for case 1 which seems to be most suitable for Kernighan-Lin's algorithm. Also, the time consumed in our algorithm is less than that in Kernighan-Lin's. At case 2, we see that the performance of our algorithm is better beyond comparison and the time consumed in ours is much less than that in Kernighan-Lin's. This effect in performance and time becomes obvious as either the amount of size difference among the nodes increases, as shown in the tables (case 3) or the value of k increases. The required execution time is shown in Figure 3 which shows that the time complexity of our algorithm is roughly $O(k \cdot |V|^2)$.

5 Conclusions

We have shown that the k-way graph partitioning problem can be transformed into the maximum k-cut problem using the proposed technique of graph modification. Also we have proposed an algorithm to solve the maximum k-cut problem based on Kernighan-Lin's method. The computing time per pass of the proposed algorithm is $O(k|V|^2)$. Experimental results indicate that the proposed algorithm performs well on a variety of graphs for 2-way partitioning problem with all sizes equal to 1, even when compared with the famous Kernighan-Lin's algorithm. We have also shown that the proposed algorithm works well Kernighan-Lin's irrespectively of k as the amount of size difference among the nodes increases. All these improvements come from the problem transformation. We are currently applying the algorithm to task allocation problem in parallel computing system called POPA which is to be built this year using 64 Transputers.

References

[1] Kernighan, B. W. and Lin, S., "An efficient heuristic procedure for partitioning graphs," *Bell Syst. Tech. J.*, Vol. 49, Feb. 1970, pp. 291-307.

[2] Ullman, J. D., *Computational Aspects of VLSI*, Computer Science Press, Maryland, 1984.

[3] Arora, R. K. and Rana, S. P., "Analysis of the module assignment problem in distributed computing systems with limited storage," *Information Processing Letters*, Vol. 10, April 1980, pp. 111-115.

[4] Fiduccia, C. M. and Mattheyses, R. M., "A linear-time heuristic for improving network partitions," *Proc. 19th Design Automation Conf.*, 1982, pp. 175-181.

[5] Krishnamurthy, B., "An improved Min-Cut Algorithm for partitioing VLSI Networks," *IEEE Trans. on Computers*, Vol. C-33, May 1984, pp. 438-446.

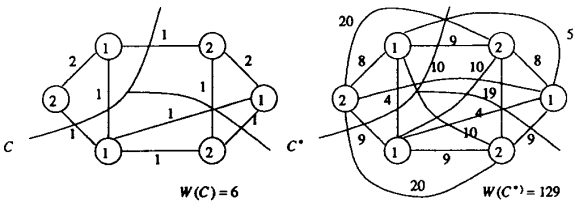
[6] Goldberg, M. and Burstein, M., "Heuristic improvement technique for bisection of VLSI networks," *Proc. Int. Conf. Comput. Des.*, 1983, pp. 122.

[7] Ng, T., Oldfield, J., and Pitchumani, V., "Improvements of a Mincut partition algorithm," *Proc. Int. Conf. Comput.-Aided Des.*, 1987, pp. 470-473.

[8] Shiraishi, H. and Hirose, F., "Efficient placement and routing for masterslice LSI," *Proc. 17th Design Automation Conf.*, June 1980, pp. 458-464.

[9] Horowitz, E. and Sahni, S., *Fundamentals of COMPUTER ALGORITHMS*, Computer Science Press, Maryland, 1978.

[10] Efe, K., "Heuristic models of task assignment scheduling in distributed systems," *IEEE Computer*, June 1982, pp. 50-56.



(a) The original graph G. (b) The modified graph G^*

Figure 1. An example graph.

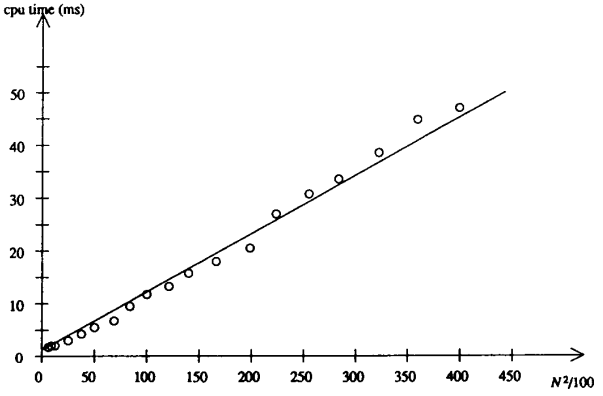


Figure 3. Running time.

Algorithm : PARTITION
Input : the transformed graph $\hat{G}$
Output : k – *partition* // $PART[1..|V|]$ //
Variables
$PART[1..|V|]$: **integer**; // $PART[i]$ = cluster number of v_i //
$GAIN[1..|V|][1..k]$: **real**; // $GAIN[i][j] = g(v_i, V_j)$ //
$STATE[1..|V|]$: **boolean**; // 0 = unused, 1 = used state //
$HISTORY[1..|V|][1..2]$: **integer**; // history information on move operations //
$TEMP[1..|V|]$: **integer**; // temporary gain for move history //
1) Construct an initial partition; // $PART[i] = 1$ *for all* i //
2) **for** $i := 1$ **to** $|V|$ **do**
  $STATE[i] := 0$; // *UNUSED* //
  for $j := 1$ **to** k **do**
    $GAIN[i][j] := g(v_i, V_j)$;
  endfor
endfor
3) **for** $i := 1$ **to** $|V|$ **do**
3.1) select unused v_d such that $g(v_d, V_l) = \max_{j,p}(GAIN[j][p])$; // $v_d \in V_m$ //
3.2) $STATE[d] := 1$; // v_d is to be moved //
3.3) $PART[d] := l$; // one – move //
3.4) $HISTORY[i][1] := d$; // save information of one – move //
3.4) $HISTORY[i][2] := m$; // save information of one – move //
3.5) $TEMP[i] := GAIN[d][l]$; // save gain of v_d //
3.6) **for** $j := 1$ **to** $|V|$ **do** // recalculate gains //
  if $STATE[j] = 1$ **thencontinue**;
  if $PART[j] = m$ **then** $GAIN[j][l] := GAIN[j][l] - 2\hat{W}(d,j)$;
  else if $PART[j] = l$ **then** $GAIN[j][m] := GAIN[j][m] + 2\hat{W}(d,j)$;
  else do
    $GAIN[j][m] := GAIN[j][m] + \hat{W}(d,j)$;
    $GAIN[j][l] := GAIN[j][l] - \hat{W}(d,j)$;
  endelse
endfor // step 3.6 //
endfor // step 3 //
4) choose t to maximize $G = \sum_{j=1}^{t} TEMP[j]$;
5) **if** $G > 0$ **then** // complete one pass //
  for $j := t+1$ **to** $|V|$ **do**
    $PART[HISTORY[j][1]] := HISTORY[j][2]$; // restore //
  endfor
  goto 2);
endif

Figure 2. The algorithm **PARTITION**.

	Ours				K-L's			
	MEAN	DEV.	PASS	TIME	MEAN	DEV.	PASS	TIME
Case 1	589.37	17.11	4.28	12.50	590.67	17.43	4.21	32.43
Case 2	590.62	17.24	4.69	11.19	640.27	20.50	4.76	238.80
Case 3	598.66	20.79	4.56	10.36	679.24	22.27	4.52	1343.55

MEAN : average cost
DEV. : standard deviation
PASS : average number of passes
TIME : average CPU time in ms

Table 1. The results of Experiment 1 ($k = 2$).

	Ours				K-L's			
	MEAN	DEV.	PASS	TIME	MEAN	DEV.	PASS	TIME
Case 1	926.37	26.27	5.21	19.67	927.36	25.80	8.78	79.24
Case 2	945.97	26.31	4.78	17.24	979.69	25.69	6.57	430.40
Case 3	953.65	27.12	4.92	15.72	992.90	30.34	6.86	1568.19

Table 2. The results of Experiment 2 ($k = 4$).

	Ours				K-L's			
	MEAN	DEV.	PASS	TIME	MEAN	DEV.	PASS	TIME
Case 1	1179.47	30.12	5.43	22.93	1187.92	29.72	13.90	68.77
Case 2	1197.20	34.93	5.15	21.65	1211.22	35.12	17.88	542.73
Case 3	1203.98	32.76	4.78	23.08	1232.49	34.74	17.80	2236.79

Table 3. The results of Experiment 3 ($k = 10$).