IMPROVING THE ESAU-WILLIAMS ALGORITHM FOR DESIGNING LOCAL ACCESS NETWORKS

By

SHINJI FUJINO

Bachelor of Science

University of Aizu

Aizu-Wakamatsu, Fukushima, Japan

1997

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
May 2000

IMPROVING THE ESAU-WILLIAMS ALGORITHM

FOR DESIGNING LOCAL ACCESS NETWORKS

Thesis Approved:

Thesis Adviser

Dean of the Graduate College

PREFACE

Most network design problems are computationally intractable. For obtaining approximations, greedy heuristics are commonly employed. The Esau-Williams algorithm adopts a better greedy heuristic in solving (constrained) capacitated minimum spanning tree (CMST) problem, using a tradeoff function computing the potential saving in the cost of a link. In this study, the component-oriented tradeoff computation was employed instead of the node-oriented one to implement the heuristic efficiently. The improved Esau-Williams algorithm was modified for variations of the CMST problems with order, degree, and depth constraints.

In the problems, a common operation was to check if accepting the link could satisfy the constraint. In the weight- and the order-constraint problems, once accepting the link would fail to satisfy the constraint, the link can be discarded. In the degree- and the depth-constraint problems, however, some links have possibility to be accepted later even though accepting them would violate the constraint at the moment. While the heuristic for the depth constraint presented in this study may be overcome by other alternative approaches, decision of accepting the link can be made when the relation between the connected components is analyzed. Thus, this improved Esau-Williams algorithm can be used as the basic algorithm for designing local access networks.

ACKNOWLEDGMENTS

I wish to express my sincere appreciation to Dr. H. K. Dai, Chairman of the Advisory Committee, for his guidance, assistance, and patience throughout my study at Oklahoma State University. I would also like to thank my committee members, Dr. John P. Chandler and Dr. Mansur H. Samadzadeh, for their helpful contributions and advice.

I would like to extend special appreciation and gratitude to my parents, who always believed in me and my abilities, for their moral and financial support and encouragement. I am also grateful to my friends and colleagues for their invaluable suggestions and support.

TABLE OF CONTENTS

Chapter Page

LIST OF FIGURES

CHAPTER I

INTRODUCTION

A network is an interconnected collection of hosts/sites for data communication. The telephone system is an example of voice networks in which sites are connected directly or indirectly by cables, fiber optics, microwaves, or communication satellites.

We can classify the networks based on their transmission technology:

1. Broadcast networks: All hosts on the network share a single communication channel/link.
2. Point-to-point networks: Many interconnection channels/links exist between pairs of hosts.

Another classification based on the distance metric results in local area networks, metropolitan area networks, and wide area networks.

Large wide area networks are typically comprised of a collection of peripheral local access networks amalgamated together by a backbone network [1]. The topology of the local access network is a tree (rooted at a backbone node) or a forest (rooted at multiple backbone nodes); all local access nodes communicate through the backbone node(s) at the root(s). The backbone network is usually irregular and meshy with high connectivity.

The design and reliability analysis of the wide area networks can be translated to those of tree networks and of a mesh backbone network [5]. Majority of the nodes in most wide area networks are in the local access portion. Therefore, efficient and optimal design of local access (tree) networks is important for wide area network design.

The thesis work is to study heuristic algorithms for some families of local access design problems, each of which is defined by a collection of constraints on data traffic, link utilization, etc.

1.1 Network Design Problems

A problem instance of a network design problem consists of:

1. The decision on the geographical placement of hosts,
2. The selection of links that satisfy traffic requirements, utilization threshold, and other constraints, and
3. The optimization of network cost and performance.

The brute-force approach in solving a network design problem is to simply enumerate all possible network configurations (host placement decision and link selection) and evaluate their costs and performance. This naive approach guarantees to work and is easy to implement, but it is impractical in that it considers even obviously absurd network configurations, and suffers a combinational explosion in the enumeration in general.

Graph theory is commonly employed to model networks. Fundamental graph algorithms are used to analyze and design networks. For example, the following uncon-

strained network design problem (minimum spanning tree problem) is solved by using some greedy algorithms, such as Kruskal's algorithm [2] or Prim's algorithm [6].

Instance: Given a graph $G = (V, E)$, where V represents the node set and E represents all possible links of network candidates with the node set V, and a real-valued cost function C on E.

Objective: Compute a minimum cost spanning tree of G (that is, minimum cost connected network on V).

Unfortunately, most network design problems are computationally intractable. Many variations of the minimum spanning tree problem are NP-complete [4]. For example, the following (constrained) capacitated minimum spanning tree (CMST) problem is NP-complete [4].

Instance: Given a graph $G = (V, E)$, a real-valued cost function C on E, a real-valued weight function W on V, and a real bound B.

Objective: Compute a minimum cost spanning tree of G in which the sum of the weights of all nodes in every subtree of the root is at most B.

The following modified Kruskal algorithm [5] adopts a naive heuristic to find a minimum spanning tree such that the capacity constraint is satisfied: the sum of all nodes in every subtree of the root is at most B. The algorithm is based on the greedy Kruskal algorithm with the capacity constraint checked before each link inclusion.

A disjoint-set data structure (see [2]) is used to maintain the disjoint sets of elements in the algorithm. The FIND-SET(u) returns the set S_u, containing a node u. The operation of UNION(u,v) is to merge two disjoint sets, S_u and S_v. Lines 1–3 initialize the set A to be the empty set and create $|V(G)|$ connected components, each

containing a single node in G. The links in $E(G)$ are sorted in order by non-decreasing cost in line 4. The **for** loop in lines 5–18 examines, for each link (u,v), if u and v are in the different connected components and the weight constraint is satisfied. If so, the link (u,v) is added to A and the two connected components are merged in lines 17–18, otherwise the link is discarded.

```
MODIFIED-KRUSKAL(G, C, W, r, B)
/* G: a given graph, G = (V, E)   C: a cost function on E                    */
/* W: a weight function on V      r: a root node   B: a bound                */
1  A ← ∅                          /* A is a set of links that compose a spanning tree */
2  for each node n ∈ V(G)
3      do MAKE-SET(n)             /* create a set, each containing a single node */
4  sort the links of E(G) by non-decreasing cost C
5  for each link (u,v) ∈ E(G), in order by non-decreasing cost C
6      do if v = r
7             then swap u and v
8          S_u ← FIND-SET(u)      /* return a set S_u, containing the node u    */
9          S_v ← FIND-SET(v)      /* return a set S_v, containing the node v    */
10         if S_u ≠ S_v
11            then if u = r
12                    then W' ← Σ_{i∈S_u} W(i)  /* store the weight of the set S_v */
13                 elseif r ∈ S_u ∪ S_v
                      /* calculate the total weight of the sets excluding the */
                      /* root r                                               */
14                    then W' ← Σ_{i∈S_u} W(i) + Σ_{j∈S_v} W(j) - W(r)
                   /* calculate the total weight of the sets                  */
15                 else W' ← Σ_{i∈S_u} W(i) + Σ_{j∈S_v} W(j)
                   /* check if the weight constraint is satisfied             */
16                 if W' ≤ B
17                    then A ← A ∪ {(u, v)}
18                         UNION(u,v)     /* merge the sets, S_u and S_v      */
19 return A
```

Kruskal's algorithm is based on a generic minimum spanning tree algorithm that is fast, simple, and effective, and therefore the modified Kruskal algorithm is a reasonable consideration. However, the solution is sub-optimal since the generic greedy algorithm usually encounters the worst-case scenario: It can leave nodes which are connected

to the root through the relatively expensive link stranded when all neighbor nodes already belong to the tree components that fill up the capacity, and the relatively expensive link to the root has to be included in the spanning tree as a result.

1.2 Contribution of the Work

To cope with the NP-hardness of most network design problems, greedy heuristics for obtaining approximations are adopted. The following two approaches will be deployed: (1) balance the greedy choices among all iterations so that the ending iterations do not suffer limited greedy choices that cannot avoid worst-case scenarios, and (2) implement the greedy algorithms using robust data structures so that the implementation can easily be adapted to similar network design problems.

The modified Kruskal algorithm computes a feasible, but likely sub-optimal solution to the CMST problem since the generic greedy heuristic often encounter the worst-case scenarios. The Esau-Williams algorithm [3] adopts a better greedy heuristic in solving the problem, using a tradeoff function computing the potential saving in cost of the link.

Unlike the generic greedy heuristic in the modified Kruskal algorithm, the heuristic of the Esau-Williams algorithm is not simple. The improvements in the proposed implementation of the Esau-Williams algorithm over the known ones are two-fold:

1. Using more efficient data structures, and
2. Changing to a component-oriented tradeoff computation from a node-oriented one.

Based upon the improved Esau-Williams algorithm, heuristic algorithms are stud-

ied to solve some variations of the CMST problem, constrained by the order of subtree, the degree of node, and the depth of node.

CHAPTER II

THE ESAU-WILLIAMS ALGORITHM

The greedy heuristic of the Esau-Williams algorithm is based on a tradeoff function. Consider an input graph for the CMST problem, in which a node i is adjacent to a node j and the root r with $C(i,j) < C(i,r)$, where C is the cost function on the link set. The greedy heuristic suggests to select the link (i,j).

In general, the difference between the link costs of a node i to a neighbor node j and to the root gives a potential saving or the tradeoff for the link (i,j). The lower the tradeoff, the more effective is the saving.

The pseudocode of the Esau-Williams algorithm [3] is outlined as follows.

```
ESAU-WILLIAMS(G, C, W, r, B)
/* G: a given graph, G = (V, E)  C: a cost function on E                    */
/* W: a weight function on V     r: a root node   B: a bound                */
1  A ← ∅        /* A is a set of links that compose a spanning tree         */
2  L ← E(G)   /* L is the set of all possible links for creating the spanning tree */
3  while |A| < |V(G)| − 1
4      do for each node n ∈ V(G)
5             do compute the tradeoff for node n, t_n /* defined in Eq.(1) below */
6          find a node i that has the lowest tradeoff t_i
7          find the cheapest link (i, j) in L where j ∈ V(G) and i ≠ j
8          if A ∪ {(i, j)} is acyclic and the weight of each connected
             component in A ∪ {(i, j)} is at most the bound B
9             then A ← A ∪ {(i, j)}
10         L ← L − {(i, j)}
11 return A
```

Lines 1–2 initialize the set A to be the empty set and L to contain candidate links for creating a spanning tree. The **while** loop in lines 3–11 finds a node i that has the lowest tradeoff among $V(G)$ and examines the cheapest candidate link from i to a neighbor node j. If the graph with link set $A \cup \{(i,j)\}$ is acyclic and the capacity constraint is satisfied, then the link is added to A in line 8, otherwise the link is removed from L in line 11. The process is repeated until the number of the links in A is $|V(G)| - 1$.

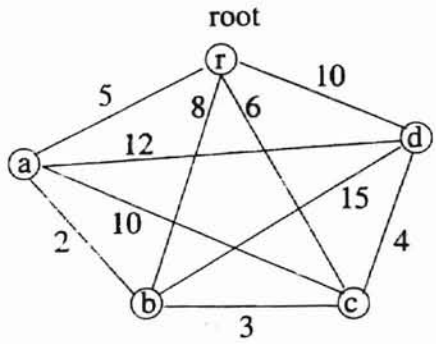
A node can be considered as a component in a graph. As two nodes are connected by a link, the two connected components are merged. The Esau-Williams algorithm merges them so that the spanning tree grows and no cycle is created. The cost of a link to the root from the connected component is used to compute the tradeoff for each node, but the cheapest link to the root will change as the connected components are merged. Then, the tradeoff for each node i is defined by:

$$\text{Tradeoff}(i) = \min_{j \notin V(T)} \text{Cost}(i,j) - \min_{k \in V(T)} \text{Cost}(k,r), \tag{1}$$

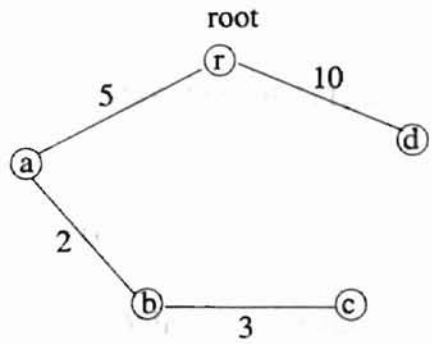
where T is the connected component containing i, and r is the root.

As an example, we consider the CMST problem with a mesh network as shown in Figure 1(a). Each node has a weight of 1 and the weight constraint of each connected component is 3. The modified Kruskal algorithm selects the links in the order (a,b), (b,c), (a,r), and (d,r). The link (c,d) is cheaper than the link (d,r) but can not be chosen because the connected component containing the nodes a, b, and c violates the weight constraint. The total cost of the spanning tree returned by the algorithm is 20 as shown in Figure 1(b). On the other hand, the Esau-Williams algorithm selects the links in the order (a,b), (c,d), (a,r), and (c,r). The total cost of the spanning

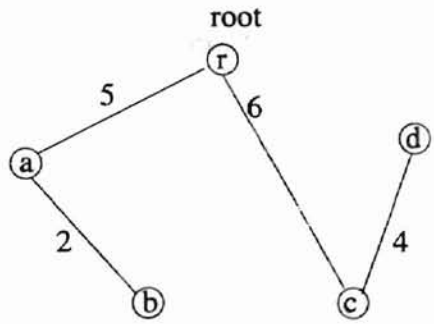
tree returned by the algorithm is 17 as shown in Figure 1(c). Thus, the Esau-Williams algorithm finds a better solution than the modified Kruskal algorithm. Traces of both algorithms are given in Appendix B.



(a) An example of the mesh network



(b) Result of the modified Kruskal algorithm



(c) Result of the Esau-Williams algorithm

Figure 1. A CMST problem. (a) An example of the mesh network. (b) is the result of the modified Kruskal algorithm and the total cost of the network is 20. (c) is the result of the Esau-Williams algorithm and the total cost of the network is 17.

CHAPTER III

IMPROVING THE ESAU-WILLIAMS ALGORITHM

3.1 Data Structures

The key to improving the efficiency of the Esau-Williams algorithm is to reduce the time complexity in finding links that have the most effective saving in the iterations. To improve the Esau-Williams algorithm, the following data structures and tradeoff computation are deployed:

1. disjoint-set,
2. heap, and
3. component-oriented tradeoff computation instead of node-oriented one.

Heaps are appropriate data structures for repeatedly searching the smallest element in a given set [5]. However, as the algorithm proceeds, the cheapest link to the root from one connected component is changed and then the value of the tradeoff for each node is accordingly changed. Thus, because it is not possible to decide the order of the greedy choices within an initial set of tradeoffs of nodes, it is necessary to update the heap as their tradeoffs are changed. To reduce a tedious and time-consuming task of updating all the heaps for the nodes in a connected component, the node-oriented tradeoff computation is shifted to a component-oriented one.

The component-oriented tradeoff computation is more efficient than the node-oriented one in that the number of the tradeoffs for each connected component decreases as two connected components are merged, and the cheapest link for each connected component to its neighbor connected components is retained and other links are discarded so that the heap finds the most potential saving link among the minimum number of candidate links.

Observe that when two connected components are merged, there are two links to a common neighbor connected component. To keep the best link to the merged connected component, a cheaper link is retained and the other is discarded.

In addition, observe that as the algorithm progresses, there will be connected components unavailable to merge to each other due to the capacity constraint. The links between them are no longer used to compute the tradeoff for the connected components; so we remove such connected components from each feasible neighbor connected component set before updating the heap. Then, the tradeoff for each connected component K, is defined by:

$$\text{Tradeoff}(K) = \min_{K' \in N(K)} \text{Cost}(K, K') - \min_{k \in K} \text{Cost}(k, r), \tag{2}$$

where K is a connected component, $N(K)$ is a set of neighbor connected components of K, $\text{Cost}(K, K')$ is the cost of a link (K, K'), and r is the root.

3.2 Pseudocode of the Improved Esau-Williams Algorithm for the CMST Problem

The pseudocode of the improved Esau-Williams algorithm is as follows. Lines 1–2 initialize the A and the collection Q to be the empty set. The **for** loop in lines 3–7 creates $|V(G)|$ connected components S_i, each containing a single node i in $V(G)$,

and initializes $C(S_i, r)$ to be the cost of a link from i to the root r, and $N(S_i)$ to be the empty set. For each S_i except for one containing the root r, the **for** loop in lines 8–17 adds feasible neighbor connected components to $N(S_i)$, computes the tradeoff for S_i, and adds S_i to Q. The link (i, j) and its cost are retained as the cheapest link from S_i to the neighbor connected component S_j in lines 14–15.

```
ESAU-WILLIAMS(G, C, W, r, B)
/* G: a given graph, G = (V, E)  W: a weight function on V                    */
/* C: a cost function on E; extended to 2^V × V and 2^V × 2^V                 */
/* r: a root node                B: a bound                                   */
1  A ← ∅                    /* A is a set of links that compose a spanning tree */
2  Q ← ∅                    /* Q is a collection of disjoint sets               */
3  for each node i ∈ V(G)
4      do MAKE-SET(i)       /* create a set, each containing a single node      */
5         S_i ← FIND-SET(i) /* return the set S_i, containing the node i        */
6         C(S_i, r) ← C(i, r) /* store a cost of the link to the root r from S_i */
7         N(S_i) ← ∅        /* N(S_i) is a collection of neighbor sets of S_i   */
   /* find feasible neighbor sets for each set                                  */
8  for each node i ∈ V(G) − {r}
9      do S_i ← FIND-SET(i)
10        for each node j ∈ V(G) − {i}
11            do S_j ← FIND-SET(j)
12               if Σ_{u∈S_i} W(u) + Σ_{v∈S_j} W(v) ≤ B /* check the weight constraint */
                    /* add S_j to N(S_i), retain the link (i, j) ∈ E(G) as a link */
                    /* for (S_i, S_j), and store the cost of the link in C(S_i, S_j) */
13                  then N(S_i) ← N(S_i) ∪ {S_j}
14                       Link(S_i, S_j) ← (i, j)
15                       C(S_i, S_j) ← C(i, j)
16        T(S_i) ← min_{S_j∈N(S_i)} C(S_i, S_j) − C(S_i, r) /* compute the tradeoff for S_i */
17        Q ← Q ∪ {S_i}                  /* add S_i to Q                        */
          /*** continued on next page ***/
```

The **while** loop in lines 18–38 repeatedly finds the most effective saving link, and adds it to A until there is no connected component in Q. Lines 19–20 find the connected component S_i that has the lowest tradeoff and a neighbor connected component S_j that is connected by the cheapest link. If the link connects S_i to the root, then S_i is

removed from Q in line 23, otherwise S_i is merged with S_j and S_j is removed from Q in lines 24–25. The **for** loop in lines 27–37 updates the relation between S_i and other feasible connected components S_u, and the tradeoff for each connected component in Q. If it is feasible to merge S_i with S_u, then update a cheaper link between them in lines 30–34, otherwise remove S_i and S_u from each feasible neighbor connected component set.

```
        /*** continued from previous page ***/
18 while |Q| > 0
19     do S_i ← MIN(Q, T)                    /* S_i has the lowest tradeoff          */
20        S_j ← EXTRACT-MIN(N(S_i), C)       /* S_j is the best neighbor of S_i      */
21        A ← A ∪ {Link(S_i, S_j)}           /* add a link (S_i, S_j) to A           */
22        if r ∈ S_j                         /* check if S_i is connected to the root */
23          then Q ← Q − {S_i}
24          else UNION(S_i, S_j)                          /* merge S_i with S_j    */
25               Q ← Q − {S_j}
26               C(S_i, r) ← min(C(S_i, r), C(S_j, r))    /* keep a cheaper link   */
                 /* check if S_i is feasible to merge with other sets in Q          */
27               for each S_u ∈ Q
28                   do N(S_u) ← N(S_u) − {S_j}
29                      if S_u ∈ N(S_i) ∩ N(S_j) and Σ_{x∈S_i} W(x) + Σ_{y∈S_u} W(y) ≤ B
                           /* update a cheaper link and its cost                  */
30                         then if C(S_i, S_u) > C(S_j, S_u)
31                                 then Link(S_i, S_u) ← Link(S_j, S_u)
32                                      Link(S_u, S_i) ← Link(S_j, S_u)
33                                      C(S_i, S_u) ← C(S_j, S_u)
34                                      C(S_u, S_i) ← C(S_j, S_u)
                           /* remove S_i and S_u from each neigbhor set           */
35                         else N(S_i) ← N(S_i) − {S_u}
36                              N(S_u) ← N(S_u) − {S_i}
                           /* update the tradeoff for S_u                         */
37                         T(S_u) ← min_{S_v∈N(S_u)} C(S_u, S_v) − C(S_u, r)
                 /* update the tradeoff for S_i                                   */
38               T(S_i) ← min_{S_v∈N(S_i)} C(S_i, S_v) − C(S_i, r)
39 return A
```

3.3 Extensions to Other Problems

As shown in Appendix B, the revised Esau-Williams algorithm gives a better solution to the CMST problem than the modified Kruskal algorithm, but it is still sub-optimal. The purpose of modifying the Esau-Williams algorithm is to use it as a base algorithm to adapt to variations of the CMST problem. In the advanced work, the base algorithm needs further modifications for accommodating additional constraints for practical network design problems, with conditions such as:

1. The local access network containing too many nodes are not designed efficiently because all of the sites will fail to access the backbone when the root is down (the order-constraint problem).
2. If a site does not have enough access ports for neighbor sites, the neighbor sites should be attached to other available sites (the degree-constraint problem).
3. If a site takes too many hops to the central site (the root), it is better to find another link to the central site with smaller number of hops (the depth-constraint problem).

These problems indicate several constraints that we should take into account in pragmatic designs. To manage the constraints, our base algorithm is modified in the next chapter.

CHAPTER IV

VARIATIONS OF THE CMST PROBLEM

4.1 Motivation

On designing local access networks, several practical constraints need to be considered. Consider single-center local access networks, which are trees rooted at a backbone network. Each local access node has a unique path to the backbone network through the root. The connectivity of a local access network is one; a node of link failure disconnects the local access network and its connection to the backbone network.

We can circumvent node or link failures by constraining the local access designs. For examples, restricting the order of each subtree, bounding the degree of each interior node, and limiting the depth of each subtree in a CMST problem.

We consider some heuristics modifying the improved Esau-Williams algorithm that give approximate solutions to these variations of the CMST problems.

4.2 The Order-Constraint Problem

The CMST problem with the order constraint is defined as follows.

Instance: Given a graph $G = (V, E)$, a root node r, a real-valued cost function C on E, and a real bound B.

Objective: Compute a minimum cost spanning tree of G in which the number of node within any rooted subtree is at most B.

As the modified Esau-Williams algorithm solves the weight-constraint problem, the algorithm can handle the order-constraint problem by checking if merging two connected components would cause the number of nodes in the merged connected component to exceed the order limit. If so, the link is discarded and a relation between the two connected components is destroyed. In the algorithm, we maintain the following inequality in any relation available for merging:

$$\text{Order}(C_i) + \text{Order}(C_j) \leq \text{Bound}_{\text{order}} \tag{3}$$

where C_i and C_j are connected components and $\text{Order}(C)$ is the number of nodes in the connected component C.

The algorithm puts a order-counter, $\text{Order}(C)$ for each connected component C. The counter starts from one, is updated when two connected components can be merged without violating the constraint.

4.3 The Degree-Constraint Problem

The CMST problem with the degree constraint is defined as follows.

Instance: Given a graph $G = (V, E)$, a root node r, a real-valued cost function C on E, and a real bound B.

Objective: Compute a minimum cost spanning tree of G in which the degree of any node is at most B.

Like the weight- and the order-constraint problems, the degree-constraint problem is also handled by checking each end-node of a link that bridges two connected com-

ponents. If accepting the link would cause the degree of one end-node to be beyond the degree limit, then the link is discarded, however unlike the weight- and the order-constraint problems, this does not imply that the two connected components are no longer merged. Since there exists at least one link between any of two connected components, it is still possibly to merge them even though the link may not be the lowest cost link that bridges them. Therefore, a set of links that bridge the two connected components needs to be maintained, and repeatedly examined until the lower cost link, which satisfies the weight, the order, and the degree constraints, is found. If an available link is found in the link set, it becomes a representative link for the pair of the two connected components. Otherwise, the relation of them is destroyed.

For each node i, the algorithm employs a degree counter, Degree(i), which starts from zero. When the link is included in a set of links to create a minimum spanning tree, the counters of end-nodes of the link are incremented by one. Let A be a set of links that are accepted by the algorithm, then Degree(i) has the following equation:

$$\text{Degree}(i) = |E'|/2 \leq \text{Bound}_{\text{degree}} \tag{4}$$

where E' is a subset of A, and contains only links whose one end-node is i.

The strategy of the component-oriented tradeoff computation is that, for each two connected components, the lowest cost link is considered as the representative link that bridges them. In the weight- and the order-constraint problems, for each pair of connected components, links except for the representative link are ignored because the representative link can save the greatest cost to bridge them, and also once the weight or the order constraints are violated, any link is unavailable to bridge them. On the other hand, in the degree-constraint problem, the set of links for each

pair of connected components needs to be maintained. If end-nodes of the current representative link would have the degree beyond the degree limit, the representative link is discarded, and then other link, whose end-nodes have the tolerable degree to connect with other node, can be replaced as the representative link.

Observe that the root of the tree formed within the connected component has possibility to be connected with the global root r. Therefore, when the link is examined. the algorithm must check if either end-node of the link has such possibility. If so, the end-node is considered as the node, which has been connected with r. If accepting the link would cause the depth of the end-node to exceed the degree limit, the link is rejected to be the representative link at this time since the root of the tree in the connected component changes as the algorithm proceeds.

The following procedure REPRESENTATIVE-LINK-DEGREE is executed when the connected components C_i and C_j are merged. Line 1 unions the set of links from C_i and C_j to their common neighbor C_u, and line 2 sorts the link set. Lines 3–17 either terminate examination for each link successfully if the degrees of both end-nodes of the link are at most a given bound, or repeat the examination after the link is rejected to be the representative link. If the link is rejected and the end-node does not have the possibility, the link is removed from the link set in line 17. If there are no links in the set, which satisfy the degree constraint, line 18 returns FALSE.

```
REPRESENTATIVE-LINK-DEGREE(C_i, C_j, C_u)
/* let Links(C_i, C_u) and Links(C_j, C_u) be sets of links that bridge C_i and C_u,  */
/* and C_j and C_u, respectively                                                      */
1  Links(C_i, C_u) ← Links(C_i, C_u) ∪ Links(C_j, C_u)
2  sort the links of Link(C_i, C_u) by non-decreasing cost
3  for each (i, u) ∈ Links(C_i, C_u) in non-decreasing order by cost Cost(i, u)
4      do P ← FALSE
5         D_i ← Degree(i) + 1
6         D_u ← Degree(u) + 1
7         if i is the root of the tree in C_i and
             Bridge-Cost(C_i, C_u) ≤ Bridge-Cost(C_u, r)
8           then D_i ← D_i + 1
9                P ← TRUE
10        elseif u is the root of the tree in C_u and
             Bridge-Cost(C_i, C_u) > Bridge-Cost(C_u, r)
11          then D_u ← D_u + 1
12               P ← TRUE
13        if D_i ≤ Bound_degree and D_u ≤ Bound_degree
14          then Bridge-Cost(C_i, C_u) ← Cost(i, u)
15               return TRUE
16        elseif P = FALSE
17          then Links(C_i, C_u) ← Links(C_i, C_u) − {(i, u)}
18 return FALSE
```

4.4 The Depth-Constraint Problem

The CMST problem with the depth constraint is defined as follows.

Instance: Given a graph $G = (V, E)$, a root node r, a real-valued cost function C on E, and a real bound B.

Objective: Compute a minimum cost spanning tree of G in which the depth of any node is at most B.

The CMST problem with the depth constraint is handled by the modified Esau-Williams algorithm, but it is more complicated to maintain the depth of each node throughout merges. Initially, each connected component contains a single node. As some connected components are merged, a new rooted tree is formed and one node,

that can be connected to the global root r by the lowest cost, becomes a root of the new tree. When two trees are bridged over the two connected components, a new rooted tree is formed such that the height of the tree is at most B. Figure 2, shows an example of two connected components and a link that is the lowest cost link available to merge them.

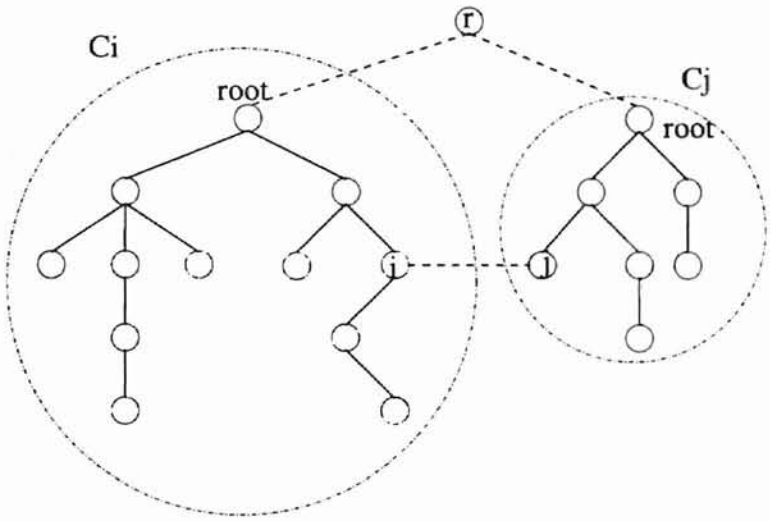


Figure 2. The two connected components C_i and C_j, and a link (i, j) that is the lowest cost link available to merge them. Two trees T_i and T_j rooted at a node that can be connected to the global root r with the lowest cost are formed within C_i and C_j. Dashed lines represent candidate links to be accepted.

Since the root of the tree changes as the tree grows, the length of the path from the root to the node is not valid until the tree is connected to r. However, if the depth of any node can be kept less than the depth limit, the result still can satisfy the depth constraint. The following procedure DEPTH-CHECK examines if accepting the link (i, j) would create a tree where any node has at most the depth limit. The procedure FIND-COMP(i) returns a pointer to the representative of the connected component

containing a node i. With any two connected components C_i and C_j, the lowest cost of a link available to bridge them is stored in Bridge-Cost(C_i, C_j). Height(i) holds the height of a tree rooted at i, and Depth(i) holds the depth of i in the rooted tree formed within the connected component. Lines 3–5 compute the maximum depth of a node in a new rooted tree, where the node is in a subtree rooted at the end-node of the link.

```
DEPTH-CHECK(i, j)
/* let i and j be end nodes of a link (i, j) */
1  C_i ← FIND-COMP(i)
2  C_j ← FIND-COMP(j)
3  if Bridge-Cost(C_i, r) > Bridge-Cost(C_j, r)
4    then D ← Height(i) + 1 + Depth(j)
5    else D ← Height(j) + 1 + Depth(i)
6  if D ≤ Bound
7    then return TRUE
8    else return FALSE
```

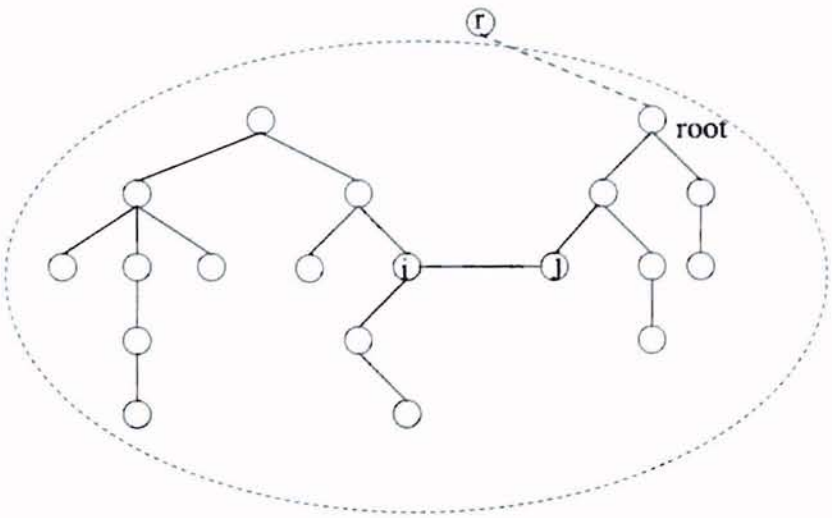


Figure 3. A connected component created by merging C_i and C_j. The two tree formed within C_i and C_j are bridged by the link (i, j).

After the link (i, j) is accepted, a merged connected component forms a new rooted tree as shown in Figure 3. For updating the depths of nodes in the two connected components, the heights of the tree rooted at nodes, and the parents of nodes, consider the trees T_i and T_j shown in Figure 4, in order to use their tree structures formed within C_i and C_j. Arrows represent pointers to the parent of a node. Let x and y be arbitrary nodes in T_i and T_j, respectively. As shown in Figure 3, the root of T_j is the root of the tree formed within the merged connected component, so the depth of y in T_j keeps the same depth and does not need the update. The depth of x in T_i is updated by calculating the sum of the depth of j, the path link (i, j), and the length of the path from x to i:

$$\text{Depth}(x) = \text{Depth}(j) + 1 + \text{the length of the path to } i, \tag{5}$$

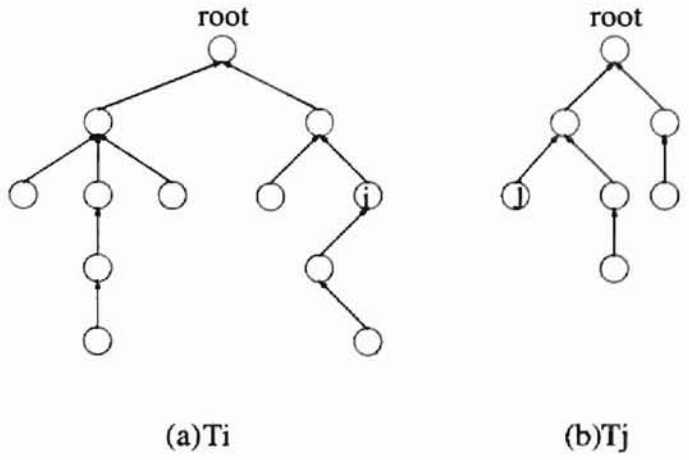


Figure 4. Rooted trees T_i and T_j from Figure 2. Arrows represent pointers to the parent of a node.

The key to getting the length of the path from x to i is to find their least common ancestor (LCA). In a rooted tree, the LCA of nodes is the ancestor node of them with

the greatest depth. Since x and i are nodes in T_i, they have the same ancestors, or either one is the ancestor of the other. Therefore, the length of the path from x to i can be obtained by the sum of the length of the paths from x and i to the LCA. Since the difference of the depth of the LCA in the tree is equal to the length of the path to the LCA, eq.(5) is rewritten to:

$$\text{Depth}(x) = \text{Depth}(j) + 1 + \text{Depth}(x) + \text{Depth}(i) - 2 \times \text{Depth}(lca) \qquad (6)$$

where lca is LCA of the nodes i and x in T_i, j is the node in T_j, and the link (i, j) bridges T_i and T_j.

```
FIND-LCA(i, x)
1  if i or x is NIL
2    then error
3  elseif i = x
4    then return i
5  elseif Depth(i) > Depth(x)
6    then return FIND-LCA(Parent(i), x)
7  elseif Depth(i) < Depth(x)
8    then return FIND-LCA(i, Parent(x))
9  else return FIND-LCA(Parent(i), Parent(x))
```

The procedure FIND-LCA(i, x) returns the LCA of two nodes i and x in a rooted tree. Parent(i) returns a pointer of the parent of i. If they are in distinct trees, their common ancestor does not exist and line 2 unsuccessfully terminates the procedure.

In the right-hand side of eq.(6), Depth(x), Depth(i) and Depth(lca) that are not updated, are not used. To avoid of using the updated data of them in the equation, an efficient method is required. Observe that, if x is i, then lca, the LCA of x and i, is also i. From eq.(6), the depth of i in the rooted tree form within the merged connected component can be obtained as follows:

$$\text{Depth}(i) = \text{Depth}(j) + 1 \qquad (7)$$

This equation is true because in the rooted tree, i and j are directly connected and the depth of i is greater than the depth of j by one as shown in Figure 3. Thus, the updating the depth of i is only dependent on the depth of j, and it can be the last among the nodes in T_i to be executed. In addition, the depth of lca is less than or equal to those of x and i in T_i.

$$\text{Depth}(lca) \leq \text{Depth}(i),\ \text{Depth}(lca) \leq \text{Depth}(x) \tag{8}$$

Since, with eq.(6), the updating the depth of lca does not use the depth of x in T_i unless lca is equal to x, the depth of each node x except for i should be updated before the depth of lca. Therefore, the updating the depth of each node x can be successfully performed by:

1. Sorting the x in order by non-increasing depth in T_i,
2. Applying eq.(6) for each node x except for i in order, and
3. Applying eq.(7) for i.

Next, the pointers to the parents of nodes in T_i are updated to point to proper parent nodes in the new rooted tree. As shown in Figure 4(a), all arrows are going toward the root of T_i. Since, for each node x in T_i, the path to the root in the new rooted tree is passing through the link (i, j), the parent pointers of nodes that are on the path from i to the root of T_i, are currently going the opposite direction. The following procedure UPDATE-PARENT-POINTER reverses the parent pointers of nodes on that path and sets the pointer of i to be j.

```
UPDATE-PARENT-POINTER(i, j)
1  if i ≠ NIL
2    then UPDATE-PARENT-POINTER(Parent(i),i)
3         Parent(i) ← j
```

For each node w within the merged connected component, Height(w) still keeps the height of the tree that is formed within in C_i or C_j and the root is w. Since accepting the link (i, j) to connect T_i and T_j, the path between each node x in T_i and each node y in T_j is newly established and so the value of Height(w) is the length of the longest path that starts from w but does not pass through the link (i, j). Note that the longest path from w passing through the link (i, j) is possibly longer than the length of the longest path from w but not passing the link (i, j). Therefore, those two longest paths need to be compared and a longer one will be retained in Height(w). The length of the longest path from w through the link (i, j) can be obtained by the following equation:

$$\text{Height}(w)' = \text{Depth}(w) + \text{Depth}(u) - 2 \times \text{Depth}(lca) + \text{Height}(v) + 1 \qquad (9)$$

where w, u and lca are nodes in either T_i or T_j, v is in the other, lca is the LCA of w and u, and u and v are end-nodes of the link (i, j). As shown in Figure 5, if w is x, the LCA of x and i is i:

$$\begin{aligned}\text{Height}(x)' &= \text{Depth}(x) + \text{Depth}(i) - 2 \times \text{Depth}(lca) + \text{Height}(j) + 1 \\ &= \text{Depth}(x) + \text{Depth}(i) - 2 \times \text{Depth}(i) + \text{Height}(j) + 1 \\ &= \text{Depth}(x) - \text{Depth}(i) + \text{Height}(j) + 1 \\ &= \text{Depth}(x) - (\text{Depth}(i) - 1) + \text{Height}(j)\end{aligned}$$

From eq.(7),

$$\text{Height}(x)' = \text{Depth}(x) - \text{Depth}(j) + \text{Height}(j). \qquad (10)$$

And if w is y, then

$$\text{Height}(y)' = \text{Depth}(y) + \text{Depth}(j) - 2 \times \text{Depth}(lca) + \text{Height}(i) + 1 \qquad (11)$$

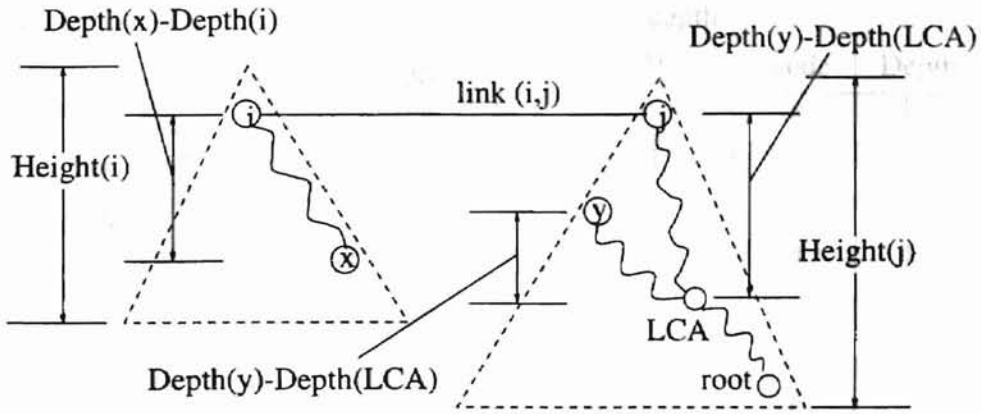
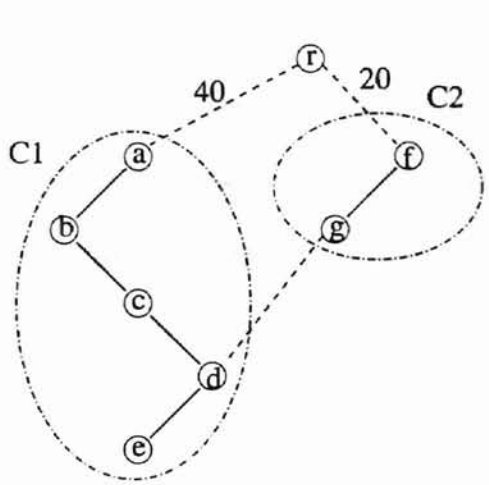


Figure 5. Showing the length of the path in a rooted tree from Figure 3. The Depth(w) currently contains the depth of w in the rooted tree formed within the merged coneected component, but Height(i) and Height(j) are the heights of the trees rooted at i and j formed within C_i and C_j shown in Figure 2.

The following procedure UPDATE-NODE is executed after each link inclusion as a part of union operation in the algorithm.

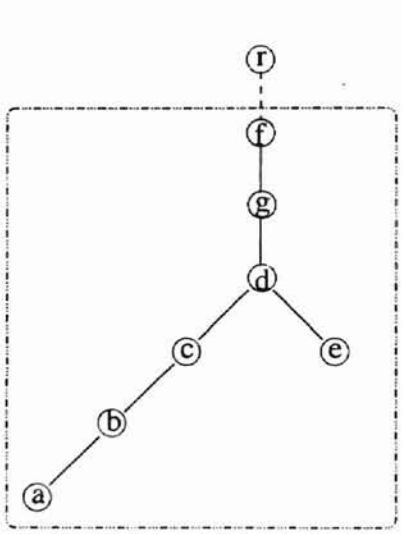
```
UPDATE-NODE(C_i, C_j, (i, j))
/* let (i, j) be a link that bridges C_i and C_j, and                    */
/* Bridge-Cost(C_i, r) > Bridge-Cost(C_j, r)                             */
1  let i and j be nodes in C_i and C_j, respectively
2  sort the nodes in C_i by non-decreasing depth
3  for each node x in C_i except for i, in order by non-decreasing depth
4      do update Depth(x)                                  /* see Eq.(6)  */
5          Height(x) = max(Height(x),Height(x)')           /* see Eq.(10) */
6  update Depth(i)                                         /* see Eq.(7)  */
7  H ← Height(i)
8  Height(i) = max(Height(i),Height(i)')                   /* see Eq.(10) */
9  UPDATE-PARENT-POINTER(i, j)
10 for each node y ∈ C_j
11     do Height(y)' ← Depth(y) + Depth(j) - 2×Depth(lca) + H + 1
12         Height(y) = max(Height(y),Height(y)')
```

Observe that how the value of the depth of the node in the rooted tree changes as the improved Esau-Williams algorithm merges the two connected components at each iteration. Figure 6(a) gives an example of merging the two connected components, C_1



node	Depth
a	1
b	2
c	3
d	4
e	5
f	1
g	2

(a) Before merging C1 and C2



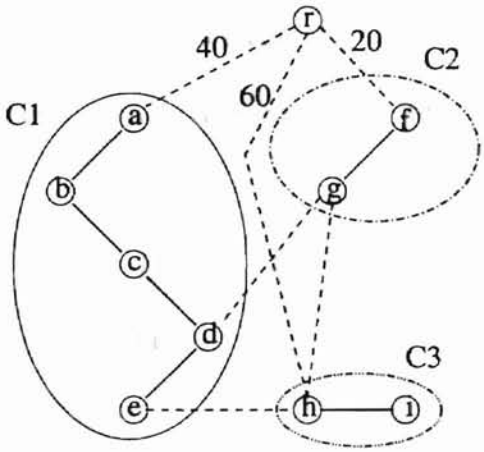
node	Depth	
a	6	greater
b	5	greater
c	4	greater
d	3	equal
e	4	less
f	1	no change
g	2	no change

(b) After merging C1 and C2

Figure 6. An example of merging two connected components and examining the depth of each node. (a) Before merging C_1 and C_2. (b) After merging C_1 and C_2.

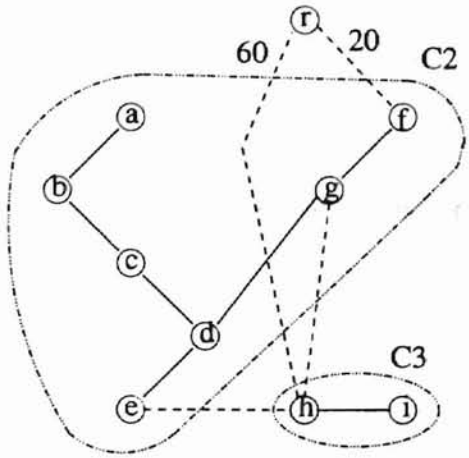
and C_2. In the figure, the link (d, g) bridges C_1 and C_2, and a and f are the roots of the trees in C_1 and C_2, respectively. Since the cost of the link from a to the global root r is greater than from f, for each node n in C_1, the structural variable Depth(n) would

be updated by eq.(6) and eq.(7). As shown in Figure 6(b), the updated Depth(n) would contain a greater, equal, or less value than the previous Depth(n). This change of the value of Depth makes effect to the result of the depth examination done by the function DEPTH-CHECK. Since the value of Depth would be used in DEPTH-CHECK, DEPTH-CHECK would not always return the same result if the value changes.



link	the maximum depth of the node	(node)	result
(d,g)	6	(a)	succeed
(e,h)	7	(i)	fail
(g,h)	4	(i)	succeed

(a) The link (e,h) fails the depth examination

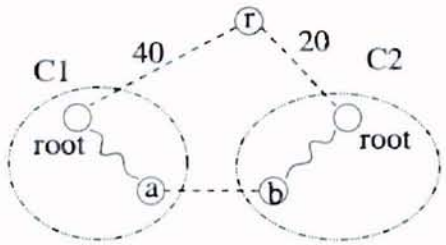


link	the maximum depth of the node	(node)	result
(e,h)	6	(a)	succeed
(g,h)	6	(a)	succeed

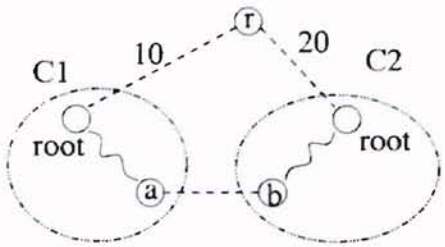
(b) The link (e,h) succeeds the depth examination

Figure 7. An example of examining the links among the connected components. The depth limit is 6. (a) The link (e, h) is not available to bridge C_1 and C_3 because accepting the link would violate the depth constraint. (b) After C_1 was merged to C_2, accepting the link (e, h) would not violate the depth constraint.

Figure 7 gives examples of the results returned by DEPTH-CHECK. Let the depth of the node limit to be 6. To analyze the relation among the connected components, the links (d, g), (e, h) and (g, h) are examined as shown in Figure 7(a). For the link (e, h), DEPTH-CHECK returns false because accepting the link (e, h) would create a rooted tree where the depth of the node i is 7. To make the maximum depth of the node at most 6, the link (e, h) is not acceptable to bridge C_1 and C_3 at this moment. Figure 7(b) shows that C_2 is merged to C_1 and Depth(e) changes. The relation between C_2 and C_3 is analyzed, and it is found that accepting the link (e, h) would create the rooted tree that satisfies the degree constraint. Thus, unlike the weight-, order-, and degree-constraint examinations, the result of the depth examination will change over iterations.



(a) Cost of the link from C1 to r is greater than C2



(b) Cost of the link from C2 to r is greater than C1

Figure 8. Examples of two connected components. (a) The cost of the link from the root in C_1 to r is greater than C_2. (b) The cost of the link from the root in C_2 to r is greater than C_1.

Next, it was analyzed what condition should be satisfied to discard the link because of violating the depth constraint. Figure 8 gives examples of checking the link in terms of the depth constraint. The link (a, b), which is available to bridge C_1 and C_2 where C_1 and C_2 contain a and b, respectively, is examined. As shown in Figure 8(a), the cost of the link from the root of the tree in C_1 to r is greater than in C_2. Suppose that accepting the link (a, b) would violate the depth constraint as the following inequality:

$$\text{Height}(a) + 1 + \text{Depth}(b) > \text{Bound}_{\text{depth}} \tag{12}$$

This inequality contains two structural variable, Height(a) and Depth(b). The value of Height(a) would be non-decreasing since it also represents the length of the longest path from a in the tree. As the improved Esau-Williams algorithm merges the two connected components in each iteration, the tree grows and the path never becomes shorter. Thus, if C_1 is merged with another neighbor, the updated height of the tree at a, Height(a)$'$, would be higher or equal to the previous height, Height(a):

$$\text{Height}(a)' \geq \text{Height}(a) \tag{13}$$

If C_2 is merged with another neighbor, however, the value of the updated depth of b is independent on the previous value. If Height(a) is greater or equal to the value of the depth limit, the link (a, b) would definitely fail the depth examination without the value of Depth(b).

If C_1 was merged with another neighbor connected component, the cost of the link from the root of the tree in C_1 to r might be lower than C_2 as shown in Figure 8(b).

If the link (a, b) is examined again, the value of following formula would be checked in DEPTH-CHECK:

$$\text{Height}(b)' + 1 + \text{Depth}(a)' \tag{14}$$

where $\text{Height}(b)'$ and $\text{Depth}(a)'$ contain the current structural data of b and a. Only if the value of $\text{Height}(b)'$ is greater or equal to the value of the depth limit, the link (a, b) would be unacceptable without the value of $\text{Depth}(a)'$.

Consequently, the result returned by DEPTH-CHECK is not enough to decide if the link is no longer possible to bridge C_1 and C_2, but if accepting the link would create a tree that satisfies the depth constraint at the moment. Thus, if and only if the following inequality is satisfied, the link (a, b) would be no longer acceptable:

$$\text{Height}(a),\ \text{Height}(b) > \text{Bound}_{\text{depth}} \tag{15}$$

In addition, if accepting any link between C_1 and C_2 would violate, but be possible to satisfy the depth constraint, set the cost of the representative link to be infinity so that C_1 and C_2 can not be merged until their structures change, however the relation between them is kept available.

The following procedure REPRESENTATIVE-LINK-DEPTH is executed when the connected components C_i and C_j are merged. Line 1 unions the set of links from C_i and C_j to their common neighbor C_u, and line 2 sorts the link set. Line 3 sets Brigde-Cost(C_i, C_u) to be infinity. Lines 4–9 either terminate examination for each link successfully if the depth constraint is satisfied, or repeat the examination after the link is rejected to be the representative link. If the link is rejected and for the

end-node, the inequality (15) is satisfied, the link is removed from the link set in line 9. If there are no links in the set, which satisfy the depth constraint, line 12 returns FALSE.

```
REPRESENTATIVE-LINK-DEPTH(C_i, C_j, C_u)
/* let Links(C_i, C_u) and Links(C_j, C_u) be sets of links that bridge C_i and C_u,  */
/* and C_j and C_u, respectively                                                      */
1  Links(C_i, C_u) ← Links(C_i, C_u) ∪ Links(C_j, C_u)
2  sort the links of Link(C_i, C_u) by non-decreasing cost
3  Bridge-Cost(C_i, C_u) ← INFINITY
4  for each (i, u) ∈ Links(C_i, C_u) in non-decreasing order by cost Cost(i, u)
5      do if DEPTH-CHECK(i, u) = TRUE
6           then Bridge-Cost(C_i, C_u) ← Cost(i, u)
7                return TRUE
8         elseif Height(i) > Bound_depth and Height(u) > Bound_depth
9           then Links(C_i, C_u) ← Links(C_i, C_u) − {(i, u)}
10 if Links(C_i, C_u) ≠ ∅
11     then return TRUE
12     else return FALSE
```

4.5 Pseudocode of the Improved Esau-Williams Algorithm for Variations of the CMST Problem

The following pseudocode for the improved Esau-Williams algorithm assumes that there are fixed number of nodes and links from each node to all others. It also assumes that any link between a node and the root is acceptable.

Lines 1–2 set A to be the empty set and Q to be the set of nodes in $V(G)$ except for the root as the representative of the connected components. The **for** loop in lines 3–18 initializes the structural fields of each node i and the connected component containing i, finds neighbors of the connected component, which is currently feasible to merge, and then computes the tradeoff of the connected component.

Because initially each connected component contains a single node, the relation between two connected components is examined by the relation between the two nodes in line 13.

```
ESAU-WILLIAMS(G, Cost, Weight, r, Bound)
/* G: a given graph, G = (V, E)      Weight: a weight function on V          */
/* Cost: a cost function on E; extended to 2^V × V and 2^V × 2^V             */
/* Bound: bounds for the weight-, order-, degree- and depth-constraints      */
/* r: a root node                                                            */
1  A ← ∅                    /* A is a set of links that compose a spanning tree */
2  Q ← V(G) − {r}           /* Q is a collection of connected components     */
   /* initialize the structual fields of each node and component            */
3  for each node i ∈ V(G)
4      do Degree(i) ← 0
5         Depth(i) ← 1
6         Height(i) ← 0
7         Parent(i) ← NIL
8         Node(i) ← {i}
9         Neighbor(i) ← {r}
10        Bridge-Cost(i, r) ← Cost(i, r)
11        W(i) ← Weight(i)
          /* find neighbor components feasible to merge                      */
12        for each node j ∈ V(G) − {i, r}
13            do if Weight(i) + Weight(j) ≤ Bound_weight
14                  then Neighbor(i) ← Neighbor(i) ∪ {j}
                         /* let (i, j) be a link that bridges i and j       */
15                       Links(i, j) ← Links(i, j) ∪ {(i, j)}
16                       Bridge-Cost(i, j) ← Cost(i, j)
17        u ← MIN(Neighbor(i), Bridge-Cost)
18        Tradeoff(i) ← Bridge-Cost(i, u)−Bridge-Cost(i, r)
19 while |Q| > 0
20    do C_i ← MIN(Q, Tradeoff)
21       C_j ← MIN(Neighbor(C_i),Bridge-Cost)
22       (i, j) ← MIN(Links(C_i, C_j), Cost)
23       A ← A ∪ {(i, j)}
24       let i and j be nodes in C_i and C_j, respectively
25       if j = r
26         then Q ← Q − {C_i}
27         else if Bridge-Cost(C_i, r) > Bridge-Cost(C_j, r)
28                 then swap C_i and C_j
29              Q ← Q − {C_j}
30              UNION(C_i, C_j, (i, j))
31 return A
```

In each iteration of the **while** loop of lines 19–28, the link (i, j) that has the lowest tradeoff is added to the set A. If j is the root, C_i is removed from the set Q in line 26. Otherwise, lines 27–29 remove either connected component, whose cost to bridge to r is greater than the other, from Q and C_i and C_j are merged. Since C_j in Neighbor(C_i) is disjointed but feasible to merge with C_i, the links in A and (i, j) that bridges C_i and C_j never form any circle on the graph. In addition, since all connected components in Q are either connected directly with the root or merged with their neighbor in lines 25–30, thus, after the **while** loop of lines 19–30, the graph with link set A becomes the MST based on the component-oriented tradeoff computation.

In the improved Esau-Williams algorithm, the following procedure UNION is an important function to maintain the structural fields. Lines 1–6 update the structural fields of nodes in C_i and C_j, and C_i merged with C_j. Since UPDATE-NODE uses the structure of the nodes in rooted trees formed within C_i and C_j, the nodes in C_j are added to the node set Node(C_i) in line 6 after structural fields of nodes are restored. Lines 7–22 recheck the relation with the neighbors of C_i and C_j, and recompute the tradeoff of the neighbors if the merging C_i and C_j makes effect the relation. In the first **for** loop of lines 7–12, for each neighbor C_u in the set Neighbor(C_j), except for one containing the root, C_j is removed from Neighbor(C_u) since C_j is merged to C_i, and the relation is examined to see if C_u is also a neighbor of C_i. If C_u is not a neighbor of C_i, C_u is removed from Neighbor(C_j) and the tradeoff of C_u is recomputed.

In the second **for** loop of lines 13–22, for each neighbor C_u in both Neighbor(C_i) and Neighbor(C_j), except for one containing the root, the relation is examined to see if the merging C_i and C_u would create a connected component that satisfies all

given constraints, and there is an available link that bridges them. If not, C_i and C_u are removed from the neighbor sets of them. The tradeoff of C_u is recomputed even though the relation between C_i and C_u is still available after the merging because REPRESENTATIVE-LINK-DEGREE and REPRESENTATIVE-LINK-DEPTH restore the lowest cost link that bridges them and then the tradeoff may change. After the relations to the neighbors of C_i are updated, lines 23–24 recompute the tradeoff of C_i.

```
UNION(C_i, C_j, (i, j))
/* let C_i and C_j be connected components, and                          */
/* Bridge-Cost(C_i, r) ≤ Bridge-Cost(C_j, r)                             */
   /* update the structural fields of nodes and the connected component C_i  */
1  let i and j be nodes in C_i and C_j, respectively
2  UPDATE-NODE(C_j, C_i, (i, j))
3  W(C_i) ← W(C_i) + W(C_j)
4  Degree(i) ← Degree(i) + 1
5  Degree(j) ← Degree(j) + 1
6  Node(C_i) ← Node(C_i) ∪ Node(C_j)
   /* recheck the relation with the neighbors                            */
7  for each C_u ∈ Neighbor(C_j) − {r}
8     do Neighbor(C_u) ← Neighbor(C_u) − {C_j}
9        if C_u ∉ Neighbor(C_i)
10         then Neighbor(C_j) ← Neighbor(C_j) − {C_u}
11              C_v ← MIN(Neighbor(C_u),Bridge-Cost)
12              Tradeoff(C_u) ← Bridge-Cost(C_u, C_v)−Bridge-Cost(C_u, r)
13 for each C_u ∈ Neighbor(C_i) − {r}
14    do if C_u ∉ Neighbor(C_j),
15          W(C_i) + W(C_u) > Bound_weight,
16          |Node(C_i)| + |Node(C_u)| > Bound_order,
17          REPRESENTATIVE-LINK-DEGREE(C_i, C_j, C_u)=FALSE, or
18          REPRESENTATIVE-LINK-DEPTH(C_i, C_j, C_u)=FALSE
19       then Neighbor(C_i) ← Neighbor(C_i) − {C_u}
20            Neighbor(C_u) ← Neighbor(C_u) − {C_i}
21       C_v ← MIN(Neighbor(C_u),Bridge-Cost)
22       Tradeoff(C_u) ← Bridge-Cost(C_u, C_v)−Bridge-Cost(C_u, r)
23 C_v ← MIN(Neighbor(C_i),Bridge-Cost)
24 Tradeoff(C_i) ← Bridge-Cost(C_i, C_v)−Bridge-Cost(C_i, r)
```

CHAPTER V

CONCLUSION AND FUTURE WORK

In improving the Esau-Williams algorithm for designing local access networks, a major approach underlying this improvement is to employ the component-oriented tradeoff computation instead of the node-oriented one. By computing the tradeoff for each connected component rather than for each node, the task of updating the structural fields of each node in the same connected component can be omitted. Since the number of the connected components decrease as the algorithm proceeds, the time complexity in finding links that have the most effective saving in the iterations, can be reduced. In addition, the relation among the connected components can be easily analyzed.

In chapter IV, to be extended to a basic algorithm for variations of the CMST problem, the improved Esau-Williams algorithm was investigated to solve the order-, degree-, and depth-constraint problems. In each problem, a common operation is to check if accepting the link could satisfy the constraint. In the weight- and the order-constraint problems, once accepting the link would fail to satisfy the constraint, the link can be discarded. In the degree- and the depth-constraint problems however, some links have possibility to be accepted later even though accepting them would violate the constraint at the moment. While the heuristic for the depth constraint

presented in this study may be overcome by other alternative approaches, decision of accepting the link can be made when the relation between the connected components is analyzed. Thus, this improved Esau-Williams algorithm can be used as the basic algorithm for designing local access networks.

In the future work, the performance of this improved Esau-Williams algorithm would be analyzed by comparing with other network design algorithms. Because of the greedy strategy, the solutions found by the algorithm are feasible, but still not guaranteed to be optimal. Thus, an approach to redesign the network found by the algorihtm might be an interesting study. In addition, the Esau-Williams algorithm can be modified for multi-center and multi-speed local access networks in a similar fashion.

REFERENCES

1. R. Cahn. *Wide Area Network Design: Concepts and Tools for Optimization.* Morgan Kaufmann, San Francisco, California, 1998.

2. T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms.* McGraw-Hill, New York, New York, 1998.

3. L. Esau and K. Williams. On teleprocessing system design: a method for approximating the optimal network. *IBM Systems Journal*, 5:142-147, 1966.

4. M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, San Francisco, California, 1979.

5. A. Kershenbaum. *Telecommunications Network Design Algorithms.* McGraw-Hill, New York, New York, 1993.

6. P. C. Prim. Shortest connection networks and some generalizations. *Bell Systems Technical Journal*, 36:1389-1401, 1957.

APPENDICES

APPENDIX A

GLOSSARY

Access Networks	The peripheral networks that connect small sites into the network.
Backbone Network	A high-capacity network which serves to connect localized networks with each other.
CMST	Capacitated Minimum Spanning Tree Problem: The total weight of each subtree at the root is at most B, where B is the weight constraint.
Connected Component	A maximal connected subgraph of a graph.
Host (Node)	A terminal on a data communications network.
Kruskal's Algorithm	A generic greedy algorithm for solving a minimum spanning tree problem. It adds edges to a forest in order by non-decreasing cost of the edge, where the edges connect two distinct tree components.
LCA	Least Common Ancestor of two nodes in a rooted tree is the ancestor node of them that has the maximum depth in the tree.
Mesh Network	A network with high connectivity.
Minimum Spanning Tree	A tree that connects all of the nodes on a given graph and whose total cost of edges is minimized.
NP	A class of decision problems that can be solved in non-deterministic polynomial time.

NP-Complete	The class of "hardest" problem in NP
NP-Hard	The class of problems that are as hard as any NP-complete problem.
Prim's Algorithm	A generic greedy algorithm for solving a minimum spanning tree problem. It adds edges to a tree in order by non-decreasing cost of the edge, where the edges connect the tree to a node not in the tree.
Site	A single or a collection of hosts.

APPENDIX B

THE TRACE OF THE ALGORITHMS FOR THE CMST PROBLEM SHOWN IN FIGURE 1

B.1 The Modified Kruskal Algorithm

- Sort the links in non-decreasing order.
  $(a,b),(b,c),(c,d),(a,r),(c,r),(b,r),(a,c),(d,r),(a,d),(b,d)$.

- Iteration 1: accepts (a,b).
  $A = \{(a,b)\}$.

- Iteration 2: accepts (b,c).
  $A = \{(a,b),(b,c)\}$.

- Iteration 3: discards (c,d) because accepting it would create a component weight of 4 more than the capacity maximum.
  $A = \{(a,b),(b,c)\}$.

- Iteration 4: accepts (a,r).
  $A = \{(a,b),(b,c),(a,r)\}$.

- Iteration 5: discards (c,r) because c and r are in the same component.
  $A = \{(a,b),(b,c),(a,r)\}$.

- Iteration 6: discards (b,r) because b and r are in the same component.
  $A = \{(a,b),(b,c),(a,r)\}$.

- Iteration 7: discards (a,c) because a and c are in the same component.
  $A = \{(a,b),(b,c),(a,r)\}$.

- Iteration 8: accepts (d,r).
  $A = \{(a,b),(b,c),(a,r),(d,r)\}$.

The total cost of the access network is

$$C_{(a,b)} + C_{(b,c)} + C_{(a,r)} + C_{(d,r)} = 2 + 3 + 5 + 10 = 20,$$

where $C_{(x,y)}$ is a cost of a link (x,y).

B.2 The Esau-Williams Algorithm

- Iteration 1:

$$\begin{aligned} t_a &= c_{a,b} - c_{a,r} = 2\text{-}5 &&= -3 \\ t_b &= c_{b,a} - c_{b,r} = 2\text{-}8 &&= -6 \\ t_c &= c_{c,b} - c_{c,r} = 3\text{-}6 &&= -3 \\ t_d &= c_{d,c} - c_{d,r} = 4\text{-}10 &&= -6 \end{aligned}$$

The link (b, a) has the most potential saving, -6, and is added to A.
$A = \{(b, a)\}$.

- Iteration 2:

$$\begin{aligned} t_a &= c_{a,r} - c_{a,r} = 5\text{-}5 &&= 0 \\ t_b &= c_{b,c} - c_{a,r} = 3\text{-}5 &&= -2 \\ t_c &= c_{c,b} - c_{c,r} = 3\text{-}6 &&= -3 \\ t_d &= c_{d,c} - c_{d,r} = 4\text{-}10 &&= -6 \end{aligned}$$

The link (d, c) has the most potential saving, -6, and is added to A.
$A = \{(b, a), (d, c)\}$.

- Iteration 3:

$$\begin{aligned} t_a &= c_{a,r} - c_{a,r} = 5\text{-}5 &&= 0 \\ t_b &= c_{b,c} - c_{a,r} = 3\text{-}5 &&= -2 \\ t_c &= c_{c,b} - c_{c,r} = 3\text{-}6 &&= -3 \\ t_d &= c_{d,r} - c_{c,r} = 10\text{-}6 &&= 4 \end{aligned}$$

The link between b and c is discarded because accepting it would create a component weight of 4, which is more than the capacity maximum 3.
$A = \{(b, a), (d, c)\}$.

- Iteration 4:

$$\begin{aligned} t_a &= c_{a,r} - c_{a,r} = 5\text{-}5 &&= 0 \\ t_b &= c_{b,r} - c_{a,r} = 8\text{-}5 &&= 3 \\ t_c &= c_{c,r} - c_{c,r} = 3\text{-}6 &&= 0 \\ t_d &= c_{d,r} - c_{c,r} = 10\text{-}6 &&= 4 \end{aligned}$$

The links (a, r) and (c, r) have the potential saving, 0, and are added to A.
$A = \{(b, a), (d, c), (a, r), (c, r)\}$.

The total cost of the access network is

$$C_{(b,a)} + C_{(d,c)} + C_{(a,r)} + C_{(c,r)} = 2 + 4 + 5 + 6 = 17.$$

B.3 The Improved Esau-Williams Algorithm

```
The Number of Nodes : 5
Root Node : 0
Weight_limit : 3
Order_limit : unconstrained
Degree_limit : unconstrained
Depth_limit : unconstrained
 *** <Cost Table> ***
Node|     0     1     2     3     4
----+-----------------------------
   0|     0     5     8     6     0
   1|     5     0     2    10    12
   2|     8     2     0     3    15
   3|     6    10     3     0     4
   4|    10    12    15     4     0

 ** Current Node Information ('*' indicates a root node) **
  Node#|Comp#|Weight|Degree|Depth
 ------+-----+------+------+-----
 *n[ 0]|c[ 0]|     1|     0|    0
  n[ 1]|c[ 1]|     1|     0|    1
  n[ 2]|c[ 2]|     1|     0|    1
  n[ 3]|c[ 3]|     1|     0|    1
  n[ 4]|c[ 4]|     1|     0|    1


 ** Current Component Information **
<Comp#|ConRt |Nearest|  Link :Cost|CostRt |Tradeoff|Weight|Nodes >
 -----+------+-------+-----------+-------+--------+------+------
 c[ 0]|  *c  | c[ 0] |( 0, 0):   0|     0 |      0 |     1| n[0]
 c[ 1]|  nc  | c[ 2] |( 1, 2):   2|     5 |     -3 |     1| n[1]
 c[ 2]|  nc  | c[ 1] |( 2, 1):   2|     8 |     -6 |     1| n[2]
 c[ 3]|  nc  | c[ 2] |( 3, 2):   3|     6 |     -3 |     1| n[3]
 c[ 4]|  nc  | c[ 3] |( 4, 3):   4|    10 |     -6 |     1| n[4]

Neighbors of Component[1]: <Component#:(link:cost)>
{<c[ 2]:( 1, 2):   2> <c[ 0]:( 1, 0):   5> <c[ 3]:( 1, 3):  10> <c[ 4]:( 1, 4):  12>}

Neighbors of Component[2]: <Component#:(link:cost)>
{<c[ 1]:( 2, 1):   2> <c[ 0]:( 2, 0):   8> <c[ 3]:( 2, 3):   3> <c[ 4]:( 2, 4):  15>}

Neighbors of Component[3]: <Component#:(link:cost)>
{<c[ 2]:( 3, 2):   3> <c[ 0]:( 3, 0):   6> <c[ 1]:( 3, 1):  10> <c[ 4]:( 3, 4):   4>}

Neighbors of Component[4]: <Component#:(link:cost)>
{<c[ 3]:( 4, 3):   4> <c[ 0]:( 4, 0):  10> <c[ 1]:( 4, 1):  12> <c[ 2]:( 4, 2):  15>}

A = {}

 *** Start EW Algorithm ***

1): The smallest Tradeoff(c[2])=-6
   Merge Comp(2) and Comp(1) by a link(2,1)

 Link(2,3) is the representative link between c[3] and c[1]
 Update Tradeoff(c[3]) = 3 - 6 = -3
 Link(1,4) is the representative link between c[4] and c[1]
 Update Tradeoff(c[4]) = 4 - 10 = -6
```

```
 Update Tradeoff(c[1]) = 3 - 5 = -2

 ** Current Component Information **
<Comp#|ConRt |Nearest|  Link :Cost|CostRt |Tradeoff|Weight|Nodes >
 -----+------+-------+-----------+-------+--------+------+------
 c[ 0]|  *c  | c[ 0] |( 0, 0):   0|     0 |      0 |     1| n[0]
 c[ 1]|  nc  | c[ 3] |( 2, 3):   3|     5 |     -2 |     2| n[1] n[2]
 c[ 3]|  nc  | c[ 1] |( 2, 3):   3|     6 |     -3 |     1| n[3]
 c[ 4]|  nc  | c[ 3] |( 4, 3):   4|    10 |     -6 |     1| n[4]

Neighbors of Component[1]: <Component#:(link:cost)>
{<c[ 3]:( 2, 3):   3> <c[ 0]:( 1, 0):   5> <c[ 4]:( 1, 4):  12>}

Neighbors of Component[3]: <Component#:(link:cost)>
{<c[ 1]:( 2, 3):   3> <c[ 0]:( 3, 0):   6> <c[ 4]:( 3, 4):   4>}

Neighbors of Component[4]: <Component#:(link:cost)>
{<c[ 3]:( 4, 3):   4> <c[ 0]:( 4, 0):  10> <c[ 1]:( 1, 4):  12>}

A = {(1,2) }

 ** Current Node Information ('*' indicates a root node) **
  Node#|Comp#|Weight|Degree|Depth
 ------+-----+------+------+-----
 *n[ 0]|c[ 0]|     1|     0|    0
  n[ 1]|c[ 1]|     1|     1|    1
  n[ 2]|c[ 1]|     1|     1|    2
  n[ 3]|c[ 3]|     1|     0|    1
  n[ 4]|c[ 4]|     1|     0|    1

2): The smallest Tradeoff(c[4])=-6
   Merge Comp(4) and Comp(3) by a link(4,3)

 Merging c[3] and c[1]) would violate the weight-constraint.
 Destory a relation between Comp(3) and Comp(1)
 Update Tradeoff(c[1]) = 5 - 5 = 0
 Update Tradeoff(c[3]) = 6 - 6 = 0

 ** Current Component Information **
<Comp#|ConRt |Nearest|  Link :Cost|CostRt |Tradeoff|Weight|Nodes >
 -----+------+-------+-----------+-------+--------+------+------
 c[ 0]|  *c  | c[ 0] |( 0, 0):   0|     0 |      0 |     1| n[0]
 c[ 1]|  nc  | c[ 0] |( 1, 0):   5|     5 |      0 |     2| n[1] n[2]
 c[ 3]|  nc  | c[ 0] |( 3, 0):   6|     6 |      0 |     2| n[3] n[4]

Neighbors of Component[1]: <Component#:(link:cost)>
{<c[ 0]:( 1, 0):   5>}

Neighbors of Component[3]: <Component#:(link:cost)>
{<c[ 0]:( 3, 0):   6>}

A = {(1,2) (3,4) }

 ** Current Node Information ('*' indicates a root node) **
  Node#|Comp#|Weight|Degree|Depth
 ------+-----+------+------+-----
 *n[ 0]|c[ 0]|     1|     0|    0
  n[ 1]|c[ 1]|     1|     1|    1
```

```
 n[ 2]|c[ 1]|     1|     1|    2
 n[ 3]|c[ 3]|     1|     1|    1
 n[ 4]|c[ 3]|     1|     1|    2

3): The smallest Tradeoff(c[1])=0
   Merge Comp(1) and Comp(0) by a link(1,0)


 ** Current Component Information **
<Comp#|ConRt |Nearest|  Link :Cost|CostRt |Tradeoff|Weight|Nodes >
 -----+------+-------+-----------+-------+--------+------+------
 c[ 0]|  *c  | c[ 0] |( 0, 0):   0|     0 |      0 |     1| n[0]
 c[ 1]|   c  | c[ 0] |( 1, 0):   5|     5 |      0 |     2| n[1] n[2]
 c[ 3]|  nc  | c[ 0] |( 3, 0):   6|     6 |      0 |     2| n[3] n[4]

Neighbors of Component[1]: <Component#:(link:cost)>
{<c[ 0]:( 1, 0):   5>}

Neighbors of Component[3]: <Component#:(link:cost)>
{<c[ 0]:( 3, 0):   6>}

A = {(1,2) (3,4) (1,0) }

 ** Current Node Information ('*' indicates a root node) **
  Node#|Comp#|Weight|Degree|Depth
 ------+-----+------+------+-----
 *n[ 0]|c[ 0]|     1|     1|    0
  n[ 1]|c[ 1]|     1|     2|    1
  n[ 2]|c[ 1]|     1|     1|    2
  n[ 3]|c[ 3]|     1|     1|    1
  n[ 4]|c[ 3]|     1|     1|    2

4): The smallest Tradeoff(c[3])=0
   Merge Comp(3) and Comp(0) by a link(3,0)


 ** Current Component Information **
<Comp#|ConRt |Nearest|  Link :Cost|CostRt |Tradeoff|Weight|Nodes >
 -----+------+-------+-----------+-------+--------+------+------
 c[ 0]|  *c  | c[ 0] |( 0, 0):   0|     0 |      0 |     1| n[0]
 c[ 1]|   c  | c[ 0] |( 1, 0):   5|     5 |      0 |     2| n[1] n[2]
 c[ 3]|   c  | c[ 0] |( 3, 0):   6|     6 |      0 |     2| n[3] n[4]

Neighbors of Component[1]: <Component#:(link:cost)>
{<c[ 0]:( 1, 0):   5>}

Neighbors of Component[3]: <Component#:(link:cost)>
{<c[ 0]:( 3, 0):   6>}

A = {(1,2) (3,4) (1,0) (3,0) }

 ** Current Node Information ('*' indicates a root node) **
  Node#|Comp#|Weight|Degree|Depth
 ------+-----+------+------+-----
 *n[ 0]|c[ 0]|     1|     2|    0
  n[ 1]|c[ 1]|     1|     2|    1
  n[ 2]|c[ 1]|     1|     1|    2
  n[ 3]|c[ 3]|     1|     2|    1
```

```
 n[ 4]|c[ 3]|     1|     1|    2
```

```
*** Complete EW Algorithm ***

The accepted links and the cost are following

   Link    | Cost
  ---------+------
  (1,2)    | 2
  (3,4)    | 4
  (1,0)    | 5
  (3,0)    | 6
----------------------------
Total Network Cost = 17
```

APPENDIX C

PROGRAM CODE OF THE IMPROVED ESAU-WILLIAMS ALGORITHM

```
/*****************************************************************************************
          IMPROVING THE ESAW-WILLIAMS ALGORITHM FOR DESINGING LOCAL ACCESS NETWORKS
                                        Shinji Fujino

1. Compile the program
   javac EW_algorithm.java

2. Create an input data file:
    The input data file must contains information about the number of nodes, the
    root node, the cost and weight tables. Information about the constraints
    (weight, order, degree, and depth) is optional. If the constraint information
    is not given, the program will solve the MST problem.
 Example:
  Node 11
  Root 3
  Cost
     200  3434  3496  3366  2730  3148  1248  1604   806  2864  3404
    3434   200  1210   432  2850  3510  2416  2034  3218  2882  3722
    3496  1210   200   988  2112  2700  2634  2252  3110  2104  2876
    3366   432   988   200  2636  2080  1304  1088  1456  1538  2350
    2730  2850  2112  2636   200  2166  2718  2372  2976  1602  2322
    3148  3510  2700  2080  2166   200  2362  2228  1946   546  1196
    1248  2416  2634  1304  2718  2362   200  2426  3266  2164  2904
    1604  2034  2252  1088  2372  2228  2426   200  1302  1294  2030
     806  3218  3110  1456  2976  1946  3266  1302   200  3296  4102
    2864  2882  2104  1538  1602   546  2164  1294  3296   200  3784
    3404  3722  2876  2350  2322  1196  2904  2030  4102  3784   200
  END
  Weight 2400  2400  2400  69600  2400  2400  2400  2400  2400  2400 2400
  Constraint
   Weight_limit 9600
   Order_limit
   Degree_limit
   Depth_limit
  END

 3. Run the program
     java EW_algorithm input_data

    If you want to create a trace file,
     java EW_algorithm input_data -trace
    The trace file, input_data.trc, would be created after the program is done.
*****************************************************************************************/
import java.io.*;
```

```
import java.util.*;

public class EW_algorithm
{
  private static Vector
    A = new Vector(0),              // A set of links to build a spanning tree
    C = new Vector(0),              // A set of components
    Q = new Vector(0),              // Priority Queue of the component-oriented tradeoff
    L = new Vector(0);              // A collection of link set between two components

  // Link Poointer Table
  private static int Link_Pointer[][];
  static int lp;                    // Link Pointer value

  // Link Class
  private static Link link;

  private static int
    nn,                             // Number of nodes
    root,                           // Root
    Cost[][],                       // Cost table
    W[];                            // Weight of node

  private static int
    Set[],                          // Pointer to the representative of the set
    Tradeoff[];                     // An array of component's tradeoff

  // Structural Fields of Node
  private static int
    Degree[],                       // Degree of each node
    Depth[],                        // Depth of each node
    Height[],                       // Height of a tree rooted at each node
    Parent[];                       // Parent of each node in a tree

  // Constraints
  private static int
    Weight_limit,                   // Weight limitation
    Order_limit,                    // Order limitaiton
    Degree_limit,                   // Degree limitation
    Depth_limit;                    // Depth limitation

  // Input File
  private static String input_file;

  // Trace Option
  static boolean trace = false;
  // Variable point indicates the place to write down in the trace file
  static long point = 0;

  /************************** Main ************************************************************/
  public static void main(String args[]) throws IOException
  {
    if(args.length > 0)
      input_file = args[0];
    else {
      System.out.print("Input file name is not given");
      System.exit(0);
    }
```

```
  if(args.length > 1)
    if(args[1].equals("-trace"))
      trace = true;

  Load_Data();

  INIT();

  EW();

  TOTAL_COST();
}

/************************** INIT ***********************************************************
 Initializes the structural fields of each node and component
*******************************************************************************************/
public static void INIT()
{
  Degree = new int[nn];
  Height = new int[nn];
  Depth = new int[nn];
  Parent = new int[nn];
  Set = new int[nn];
  Tradeoff = new int[nn];

  Link_Pointer = new int[nn][nn];
  lp = 1;

  for(int i = 0; i < nn; i++) {

    Degree[i] = 0;
    Height[i] = 0;
    Depth[i] = (i == root) ? 0 : 1;
    Parent[i] = -1;
    Set[i] = i;

    MAKE_COMP(i,root,nn);
  }

  if(trace) {
    CostTable();
    NodeInfo();  }

}

/************************** MAKE_COMP ******************************************************
 Creates a new component
*******************************************************************************************/
public static void MAKE_COMP(int i, int root, int nn)
{
  Component Comp = new Component(i);

  if(i == root)
    Comp.ConRt = true;
  else
    Q.addElement(String.valueOf(i));
```

```
  Comp.NComp.addElement(String.valueOf(root));
  Comp.BridgeCost = new int[nn];
  Comp.BridgeCost[root] = Cost[i][root];
  Comp.W = W[i];

  for(int j = 0; j < nn; j++) {
    Comp.Links.addElement(new Link(i, j, Cost[i][j]));
    Comp.BridgeCost[j] = Cost[i][j];

    if(j == i || j == root)
      continue;

    if(W[i] + W[j] <= Weight_limit) {
      Comp.NComp.addElement(String.valueOf(j));
      SET_LINKS(i, j, Cost[i][j]);
    }
  }
  int u = Minimum(Comp.NComp, Comp.BridgeCost);
  Tradeoff[i] = Cost[i][u] - Comp.BridgeCost[root];
  C.addElement(Comp);
}

/************************** SET_COMP ***************************************************
 Sets up the structural fields of a componentn and calculates the tradeoff
*****************************************************************************************/
public static void SET_LINKS(int i, int j, int cost)
{
  if(Link_Pointer[i][j] == 0) {
    Vector Links = new Vector(0);
    Links.addElement(new Link(i, j, cost));
    Link_Pointer[i][j] = lp;
    Link_Pointer[j][i] = lp++;
    L.addElement(Links);
  }
}

/************************** EW *********************************************************
 Implements the Esaw-Williams Algorithm
*****************************************************************************************/
public static void EW()
{
  int C_i, C_j, i = 0;
  Component Comp_i, Comp_j, Comp_t;

  if(trace) {
    CurrentSituation();
    StoreData( "\n *** Start EW Algorithm *** \n\n" ); }

  while(Q.size() > 0) {

    C_i = Minimum(Q, Tradeoff);
    Comp_i = (Component) C.elementAt(C_i);

    C_j = Integer.parseInt(Comp_i.NearestNComp());
    Comp_j = (Component) C.elementAt(C_j);

    link = Comp_i.LinkTo(C_j);
    A.addElement(link);
```

```
    if(trace) {
      StoreData(++i + "): " );
      StoreData("The smallest Tradeoff(c[" + String.valueOf( C_i ) +
        "])=" + Tradeoff[C_i] + "\n" );
      StoreData("   Merge Comp(" + C_i + ") and Comp(" + C_j + ") by ");
      StoreData("a link(" + link.end1 + "," + link.end2 + ") \n\n" ); }

    /* Special Case that a component will be connected to a root */
    if(link.end1 == root || link.end2 == root) {
      Comp_i.ConRt = true;
      Parent[link.end1] = root;
      Degree[link.end1]++;
      Degree[link.end2]++;
      Q.removeElement(String.valueOf(Comp_i.ID));
    }
    else {
      if(Comp_i.BridgeCost[root] > Comp_j.BridgeCost[root]) {
        Comp_t = Comp_i;
        Comp_i = Comp_j;
        Comp_j = Comp_t;
      }

      Q.removeElement(String.valueOf(Comp_j.ID));
      UNION(Comp_i, Comp_j, link);
    }

    if(trace) {
      CurrentSituation();
      NodeInfo(); }
  }

  if(trace)
    StoreData("\n *** Complete EW Algorithm *** \n\n");
}

/************************** UNION ************************************************************
 Updates the structural fields of nodes in the componennts Comp_i and Comp_j
 so that they are bridged by the link
*********************************************************************************************/
public static void UNION(Component Comp_i, Component Comp_j, Link link)
{
  /* Update CompN and ConRt */
  Set[Comp_j.ID] = Comp_i.ID;

  /* let node_i and node_j be nodes in Comp_i and Comp_j, respectively */
  if(Comp_i.Nodes.contains(String.valueOf(link.end2))) {
    int t = link.end1;
    link.end1 = link.end2;
    link.end2 = t;
  }

  Update_Node(Comp_j, Comp_i, link.end2, link.end1);

  // Nodes(Comp_i) = Nodes(Comp_i) U Nodes(Comp_j)
  for(int index = 0; index < Comp_j.Nodes.size(); index++)
    Comp_i.Nodes.addElement(Comp_j.Nodes.elementAt(index));
  // Clear Component Nodes
```

```
  Comp_j.Nodes.removeAllElements();

  /* Update Weight of component */
  Comp_i.W = Comp_i.W + Comp_j.W;
  Degree[link.end1]++;
  Degree[link.end2]++;
  Update_Relation(Comp_i, Comp_j);
}

/************************* Update_Node ***************************************************
 Updates the structural fields of nodes in the componennts Comp_i and Comp_j
 where Cost(Comp_i,root) > Cost(Comp_j,root)
*****************************************************************************************/
public static void Update_Node(Component Comp_i, Component Comp_j,
                               int node_i, int node_j)
{
  int node_x, node_y, H;

  /* sort the nodes in Comp_i by non-decreasing depth */
  SortNodes(Comp_i.Nodes, Depth);

  for(int index = Comp_i.Nodes.size() - 1 ; index >= 0 ; index--) {
    node_x = Integer.parseInt((String) Comp_i.Nodes.elementAt(index));
    if(node_x != node_i) {
      Update_Depth(node_x, node_i, Depth[node_j]);
      Update_Height_I(node_x, node_j);
    }
  }

  Depth[node_i] = 1 + Depth[node_j];
  H = Height[node_i];
  Update_Height_I(node_i, node_j);
  Update_Parent(node_i, node_j);

  for(int index = 0; index < Comp_j.Nodes.size(); index++) {
    node_y = Integer.parseInt((String) Comp_j.Nodes.elementAt(index));
    Update_Height_J(node_y, node_j, H);
  }
}

/************************* Update_Depth ***************************************************
 Updates the depth of x in the rooted tree
*****************************************************************************************/
public static void Update_Depth(int x, int i, int depth_j)
{
  int LCA = Find_LCA(x, i);

  Depth[x] = Depth[x] + Depth[i] - 2*Depth[LCA] + depth_j + 1;
}

/************************* Update_Height ***************************************************
 Updates the height of the tree rooted at x in the component
*****************************************************************************************/
public static void Update_Height_I(int x, int j)
{
  Height[x] = Math.max(Height[x], Height[j] + Depth[x] - Depth[j]);
}
```

```
/************************** Update_Height ***************************************************
 Updates the height of the tree rooted at y in the component
*********************************************************************************************/
public static void Update_Height_J(int y, int j, int height_i)
{
  int LCA;

  LCA = Find_LCA(y, j);

  Height[y] = Math.max(Height[y],
    Depth[y] + Depth[j] - 2*Depth[LCA] + height_i + 1);
}

/************************** Update_Parent ***************************************************
 Sets the parent pointers of the node  and the parent of i to be j
*********************************************************************************************/
public static void Update_Parent(int i, int j)
{
  if(i != -1) {
    Update_Parent(Parent[i], i);
    Parent[i] = j;
  }
}

/************************** Find_LCA ********************************************************
 Returns the Least Common Ancestor of two nodes x and i
*********************************************************************************************/
public static int Find_LCA(int x, int i)
{
  if(x == -1 || i == -1) {
    System.out.print("LCA of " + x + " and " + i + " is not found.\n");
    System.exit(0);
  }

  if(x == i)
    return x;
  else {
    if(Depth[x] > Depth[i])
      return Find_LCA( Parent[x], i);
    else if(Depth[x] < Depth[i])
      return Find_LCA(x, Parent[i]);
    else
      return Find_LCA(Parent[x], Parent[i]);
  }
}

/************************** Update_Relation *************************************************
 Checks if two components are available to merge
*********************************************************************************************/
public static void Update_Relation(Component Comp_i, Component Comp_j)
{
  Component Comp_u;
  int index = 0;

  while(index < Comp_j.NComp.size()) {
    int u = Integer.parseInt((String) Comp_j.NComp.elementAt(index));

    if(u == root) {
```

```
      index++;
      continue;
    }

    Comp_u = (Component) C.elementAt(u);

    // Neighbor(Comp_u) = Neighbor(Comp_u) - {Comp_j}
    Comp_u.NComp.removeElement(String.valueOf(Comp_j.ID));

    if(!Comp_i.NComp.contains(String.valueOf(u)) && u != Comp_i.ID) {
      Comp_j.NComp.removeElementAt(index);
      UpdateTradeoff(Comp_u);
      C.removeElementAt(u);
      C.insertElementAt(Comp_u ,u);
    }
    else
      index++;
  }

  index = 0;
  while(index < Comp_i.NComp.size()) {
    int u = Integer.parseInt((String) Comp_i.NComp.elementAt(index));

    if(u == root) {
      index++;
      continue;
    }

    Comp_u = (Component) C.elementAt(u);
    if(Comp_j.NComp.contains(String.valueOf(u)) == false ||
      WeightCheck(Comp_i,Comp_u) == false ||                    // Checks weight
      OrderCheck(Comp_i,Comp_u) == false ||                     // Checks order
      Representative_Link(Comp_i, Comp_j, Comp_u) == false)     // Checks degree and depth
      DestroyRelation(Comp_i, Comp_u);
    else
      index++;

    UpdateTradeoff(Comp_u);
    C.removeElementAt(u);
    C.insertElementAt(Comp_u ,u);
  }

  UpdateTradeoff(Comp_i);

  C.removeElementAt(Comp_i.ID);
  C.insertElementAt(Comp_i ,Comp_i.ID);
}

/************************** WeightCheck ***************************************************
 Checks if merging two components would satisfy the weight-constraint
*******************************************************************************************/
public static boolean WeightCheck(Component Comp_i, Component Comp_j)
{
  if(Comp_i.W + Comp_j.W <= Weight_limit)
    return true;
  else {
    if(trace)
      StoreData(" Merging c[" + Comp_i.ID + "] and c[" + Comp_j.ID + "]" +
```

```
                ") would violate the weight-constraint.\n");
    return false;
  }
}

/************************** OrderCheck ********************************************************
 Checks if merging two components would satisfy the order-constraint
**********************************************************************************************/
public static boolean OrderCheck(Component Comp_i, Component Comp_j)
{
  if(Comp_i.Nodes.size() + Comp_j.Nodes.size() <= Order_limit)
    return true;
  else {
    if(trace)
      StoreData(" Merging c[" + Comp_i.ID + "] and c[" + Comp_j.ID + "]" +
                ") would violate the order-constraint.\n");
    return false;
  }
}

/************************** Representative_Link ***********************************************
 Checks to see if there is the lowest cost link available to bridge two components
**********************************************************************************************/
public static boolean Representative_Link(Component Comp_i, Component Comp_j,
                                          Component Comp_u)
{
  int index = 0, node_i, node_j, degree_i, degree_j;
  int i_p = Link_Pointer[Comp_i.ID][Comp_u.ID] - 1;
  int j_p = Link_Pointer[Comp_j.ID][Comp_u.ID] - 1;
  Vector Links_i = (Vector) L.elementAt(i_p);
  Vector Links_j = (Vector) L.elementAt(j_p);
  boolean possible = false;

  // Insert the links of Links_j to Links_i in order to sort the links by cost
  InsertLinks(Links_i, Links_j);

  // Set the brige-cost to be the cost of the link to the root plus one besides infinity
  Comp_i.BridgeCost[Comp_u.ID] = Comp_i.BridgeCost[root] + 1;
  Comp_u.BridgeCost[Comp_i.ID] = Comp_i.BridgeCost[root] + 1;

  while(index < Links_i.size()) {
    link = (Link) Links_i.elementAt(index);

    if(Comp_i.Nodes.contains(String.valueOf(link.end1))) {
      node_i = link.end1;
      node_j = link.end2;
    }
    else {
      node_i = link.end2;
      node_j = link.end1;
    }

    degree_i = Degree[node_i];
    degree_j = Degree[node_j];
    if(Comp_i.BridgeCost[root] <= Comp_u.BridgeCost[root]) {
      if(Comp_i.ID == node_i) {        // node_i would be connected with the root
        possible = true;
        degree_i += 2;
```

```
    }
    else
      degree_i += 1;

    degree_j += 1;
  }
  else if(Comp_i.BridgeCost[root] > Comp_u.BridgeCost[root]) {
    if(Comp_j.ID == node_j) {        // node_j would be connected with the root
      possible = true;
      degree_j += 2;
    }
    else
      degree_j += 1;

    degree_i += 1;
  }

  if(degree_i > Degree_limit || degree_j > Degree_limit) {
    if(possible == false) {
      if(trace)
        StoreData(" A link (" + link.end1 + "," + link.end2 +
                  ") is discarded because of violating the degree-constraint.\n");
      Links_i.removeElement(link);
    }
    else
      index++;
  }
  else if(DepthCheck(Comp_i, Comp_u, node_i, node_j)) {
    if(trace)
      StoreData(" Link(" + node_i + "," + node_j + ") is the representative link" +
          " between c[" + Comp_u.ID + "] and c[" + Comp_i.ID + "]\n");

    Comp_i.UpdateLink(Comp_u.ID, link.cost, link);
    Comp_u.UpdateLink(Comp_i.ID, link.cost, link);
    break;
  }
  else {
    if(Height[link.end1] >= Depth_limit && Height[link.end1] >= Depth_limit) {
      if(trace)
        StoreData(" A link (" + link.end1 + "," + link.end2 +
                  ") is discarded because of violating the depth-constraint.\n");
      Links_i.removeElement(link);
    }
    else
      index++;
  }
  possible = false;
}

if(Links_i.size() > 0)
  return true;
else {
  if(trace)
    StoreData("Any link can not bridge c[" + Comp_i.ID + "] and c[" + Comp_j.ID + "]");

  return false;
}
```

```
}

/************************** DepthCheck ***************************************************
 Checks if accepting the link would satisfy the depth-constraint
*****************************************************************************************/
public static boolean DepthCheck(Component Comp_i, Component Comp_j,
                                 int node_i, int node_j)
{
  int depth, height;

  if(Comp_i.BridgeCost[root] > Comp_j.BridgeCost[root]) {
    height = Height[node_i];
    depth = Depth[node_j];
  }
  else {
    height = Height[node_j];
    depth = Depth[node_i];
  }

  if(depth + 1 + height <= Depth_limit)
    return true;
  else
    return false;
}

/************************** DestroyRelation **********************************************
 Destroys the relation between two components
*****************************************************************************************/
public static void DestroyRelation(Component Comp_i, Component Comp_u)
{
  Comp_i.NComp.removeElement(String.valueOf(Comp_u.ID));
  Comp_u.NComp.removeElement(String.valueOf(Comp_i.ID));

  if(trace) {
    StoreData(" Destory a relation between" );
    StoreData(" Comp(" + Comp_i.ID + ") and Comp(" + Comp_u.ID + ")");
    StoreData("\n"); }

}

/************************** UpdateTradeoff ***********************************************
 Recalculates the tradeoff for the component
*****************************************************************************************/
public static void UpdateTradeoff(Component Comp)
{
  int iNeigh = Minimum(Comp.NComp, Comp.BridgeCost);
  Tradeoff[Comp.ID] = Comp.BridgeCost[iNeigh] - Comp.BridgeCost[root];

  if(trace) {
    StoreData(" Update Tradeoff(c[" + Comp.ID + "]) = " +
      Comp.BridgeCost[iNeigh] + " - " + Comp.BridgeCost[root] + " = "
      + Tradeoff[Comp.ID] + "\n"); }
 }

/************************** FIND_COMP ****************************************************
 Identifies the componet that contains a node n
*****************************************************************************************/
public static int FIND_COMP(int n)
```

```
{
  if(n == Set[n])
    return n;
  else
    return FIND_COMP(Set[n]);
}


/************************* SortNodes ********************************************************
 Sorts the nodes in order by non-decreasing value of key
*********************************************************************************************/
public static void SortNodes(Vector Nodes, int[] Keys)
{
  int[] Elements = new int[Nodes.size()];

  for(int i = 0; i < Nodes.size(); i++)
    Elements[i] = Integer.parseInt((String) Nodes.elementAt(i));

  HeapSort(Elements, Keys);

  for(int i = 0; i < Elements.length; i++) {
    Nodes.removeElementAt(0);
    Nodes.addElement(String.valueOf(Elements[i]));
  }
}

/************************* InsertLinks ******************************************************
 Inserts links in the set links_j into the set links_i
*********************************************************************************************/
public static void InsertLinks(Vector Links_i, Vector Links_j)
{
  int Elements[] = new int[Links_i.size()+Links_j.size()];
  int Keys[] = new int[Links_i.size()+Links_j.size()];
  int l, parent, index;

  for(index = 0; index < Links_i.size(); index++) {
    Elements[index] = index;
    Link link_i = (Link) Links_i.elementAt(index);
    Keys[index] = link_i.cost;
  }

  for(int i = 0; i < Links_j.size(); i++) {
    l = index;
    Link link_j = (Link) Links_j.elementAt(i);
    Keys[index] = link_j.cost;

    parent = (l - 1)/2;

    while(l >= 1 && Keys[Elements[parent]] > Keys[l]) {
      Elements[l] = Elements[parent];
      Link link_i = (Link) Links_i.elementAt(parent);
      if(l >= Links_i.size()) {
        Links_i.addElement(link_i);
      }
      else {
        Links_i.removeElementAt(l);
        Links_i.insertElementAt(link_i, l);
      }
```

```
      l = parent;
      parent = (l - 1)/2;
    }

    Elements[l] = index;
    if(l < Links_i.size())
      Links_i.removeElementAt(l);

    Links_i.insertElementAt(link_j, l);
    index++;
  }
}

/************************** HeapSort ***********************************************************
 Sorts the elements in order by non-decreasing value of key
***********************************************************************************************/
public static void HeapSort(int[] Elements, int[] Keys)
{
  int size = Elements.length;

  BuildHeap(Elements, Keys);

  for(int i = size; i >= 2; i--) {
    size--;
    int buf = Elements[0];
    Elements[0] = Elements[size];
    Elements[size] = buf;
    Heapify(Elements, Keys, size, 0);
  }
}


/************************** Minimum ************************************************************
 Finds an element which has the smallest value of key
***********************************************************************************************/
public static int Minimum(Vector Q, int[] Keys)
{
  int size = Q.size();

  int[] Elements = new int[Q.size()];

  for(int i = 0; i < size; i++)
    Elements[i] = Integer.parseInt((String) Q.elementAt(i));

  BuildHeap(Elements, Keys);

  Q.removeElement(String.valueOf(Elements[0]));
  Q.insertElementAt(String.valueOf(Elements[0]), 0);

  return Elements[0];
}

/************************** BuildHeap **********************************************************
 Builds a heap structure in a given elements set
***********************************************************************************************/
public static void BuildHeap(int[] Elements, int[] Keys)
{
```

```
  for(int i = Elements.length / 2; i >= 1; i--)
    Heapify(Elements, Keys, Elements.length, i-1);
}

/************************** Heapify ***********************************************************
 Heapifies the given elements set with keys
 If the values of two elements are equal, compare their element number
**********************************************************************************************/
public static void Heapify(int[] Elements, int[] Keys, int size, int i)
{
  int l = 2 * i + 1;
  int r = 2 * i + 2;
  int smallest;

  if(l < size && Keys[Elements[l]] < Keys[Elements[i]])
    smallest = l;
  else
    smallest = i;

  if(r < size && Keys[Elements[r]] < Keys[Elements[smallest]])
    smallest = r;

  if(smallest != i) {
    int buf = Elements[i];
    Elements[i] = Elements[smallest];
    Elements[smallest] = buf;

    Heapify(Elements, Keys, size, smallest);
  }
}

/************************** TOTAL_COST ********************************************************
 Calculates the total cost of accepted links
**********************************************************************************************/
public static void TOTAL_COST()
{
  int n1, n2, index, final_cost;

  final_cost = 0;

  System.out.print(" The accepted links and the cost are following \n\n");
  System.out.print("\t Link \t| Cost \n");
  System.out.print("\t--------+------\n");
  if(trace) {
    StoreData(" The accepted links and the cost are following \n\n");
    StoreData("\t Link \t| Cost \n");
    StoreData("\t--------+------\n");
  }
  for(int i = 0; i < A.size(); i++) {
    link = (Link) A.elementAt(i);

    final_cost = final_cost + link.cost;

    System.out.print("\t(" + link.end1 + "," + link.end2 + ")\t| " + link.cost + "\n");
    if(trace)
      StoreData("\t(" + link.end1 + "," + link.end2 + ")\t| " + link.cost + "\n");
  }
  System.out.print(" --------------------------\n");
```

```
  System.out.print(" Total Network Cost = " + final_cost + "\n");
  if(trace) {
    StoreData(" ---------------------------\n");
    StoreData(" Total Network Cost = " + final_cost + "\n");
  }
}

/************************** Load_Data ***********************************************************
 Loads Network Data from an external file
*************************************************************************************************/
public static void Load_Data()
{
  try{
    RandomAccessFile NodeData = new RandomAccessFile(input_file,"r");
    String data, key;
    StringTokenizer ST;
    boolean b_cost = false, b_const = false;
    int token, line = 0, row = 0;
    int total_weight = 0;
    long f_pointer = NodeData.getFilePointer();
    long lline = NodeData.length();

    nn = -1; root = -1;
    Weight_limit = -1; Order_limit = -1; Degree_limit = -1; Depth_limit = -1;

    while((f_pointer = NodeData.getFilePointer()) < NodeData.length()) {
      data = NodeData.readLine();
      ST = new StringTokenizer(data);

      if(ST.countTokens() == 0)
        continue;

      if(b_cost) {
        key = ST.nextToken();
        if(key.equals("END"))
          b_cost = false;
        else {
          if(ST.countTokens() < nn - 1 || row > nn) {
            System.out.print("The size of the cost table is wrong.");
            System.exit(0);
          }

          Cost[row][0] = Integer.parseInt(key);
          for(int column = 1; column < nn; column++)
            Cost[row][column] = Integer.parseInt(ST.nextToken());
          row++;
        }
      }
      else if(b_const) {
        key = ST.nextToken();
        if(key.equals("END"))
          b_const = false;
        else {

          if(ST.hasMoreTokens()) {
            if(key.equals("Weight_limit"))
              Weight_limit = Integer.parseInt(ST.nextToken());
            else if(key.equals("Order_limit")) {
```

```
          Order_limit = Integer.parseInt(ST.nextToken());
        }
        else if(key.equals("Degree_limit"))
          Degree_limit = Integer.parseInt(ST.nextToken());
        else if(key.equals("Depth_limit"))
          Depth_limit = Integer.parseInt(ST.nextToken());
        else {
          System.out.print("Unrecognized Key:" + key);
          System.exit(0);
        }
      }
    }
  }
  else {
    key = ST.nextToken();
    if(key.equals("Node")) {
      if(!ST.hasMoreTokens()) {
        System.out.print("The number of nodes is not given.");
        System.exit(0);
      }
      nn = Integer.parseInt(ST.nextToken());
      Cost = new int[nn][nn];
      W = new int[nn];
    }
    else if(key.equals("Root"))
      root = Integer.parseInt(ST.nextToken());
    else if(key.equals("Constraint"))
      b_const = true;
    else if(key.equals("Weight")) {
      if(nn < 1) {
        System.out.print("The number of nodes is not given.");
        System.exit(0);
      }
      if(ST.countTokens() != nn) {
        System.out.print("The number of weight on node is not matched.");
        System.exit(0);
      }

      for(int i = 0; i < nn; i++) {
        W[i] = Integer.parseInt(ST.nextToken());
        total_weight += W[i];
      }
    }
    else if(key.equals("Cost")) {
      b_cost = true;
      row = 0;
    }
    else {
      System.out.print("Unrecognized Key:" + key);
      System.exit(0);
    }
  }
}
if(b_cost || b_const) {
  System.out.print("Format Error: END is missing");
  System.exit(0);
}
```

```
    if(trace) {
    StoreData("The Number of Nodes : " + nn + "\n" +
      "Root Node : " + root + "\n"); }

    if(Weight_limit != -1) {
      if(trace)
        StoreData("Weight_limit : " + Weight_limit + "\n");
    }
    else {
      if(trace)
        StoreData("Weight_limit : unconstrained\n");
      Weight_limit = total_weight;
    }
    if(Order_limit != -1) {
      if(trace)
        StoreData("Order_limit : " + Order_limit + "\n");
    }
    else {
      if(trace)
        StoreData("Order_limit : unconstrained\n");
      Order_limit = nn;
    }
    if(Degree_limit != -1) {
      if(trace)
        StoreData("Degree_limit : " + Degree_limit + "\n");
    }
    else {
      if(trace)
        StoreData("Degree_limit : unconstrained\n");
      Degree_limit = nn;
    }
    if(Depth_limit != -1) {
      if(trace)
        StoreData("Depth_limit : " + Depth_limit + "\n" );
    }
    else {
      if(trace)
        StoreData("Depth_limit : unconstrained\n");
      Depth_limit = nn;
    }

  }
  catch (FileNotFoundException e) {
    System.err.println(e);
  }
  catch (IOException e) {
    System.err.println(e);
  }
}

/************************** StoreData ***************************************************
 Stores a string data into the trace file
*****************************************************************************************/
public static void StoreData(String S)
{
  try
  {
    if(point == 0) {
```

```
      File file = new File(input_file + ".trc");
      if(file.exists())
        file.delete();
    }

    RandomAccessFile result = new RandomAccessFile(input_file+".trc", "rw");
    result.seek(point);
    result.writeBytes(S);

    point = result.getFilePointer();
    result.close();
  }
  catch (FileNotFoundException e)
  {
    System.err.println(e);
  }
  catch (IOException e)
  {
    System.err.println(e);
  }
}


/*****************************************************************************************
*****************************************************************************************
***                          TRACE FUNCTIONS                                          ***
*****************************************************************************************
*****************************************************************************************/


/************************** CostTable ***************************************************
 Writes Cost Table into the trace file
*****************************************************************************************/
public static void CostTable()
{
  String sCost;
  StoreData(" *** <Cost Table> ***\nNode|");
  for(int i = 0; i < nn; i++) {
    if(i >= 10)
      StoreData("    " + i);
    else
      StoreData("     " + i);
  }

  StoreData("\n----+");
  for(int i = 0; i < nn; i++)
    StoreData("------");
  StoreData("\n");

  for(int i = 0; i < nn; i++) {
    if(i >= 10)
      StoreData("  " + i + "|");
    else
      StoreData("   " + i + "|");

    for(int j = 0; j < nn; j++) {
      sCost = String.valueOf(Cost[i][j]);
      sCost = WhiteSpace(6 - sCost.length()) + sCost;
```

```
      StoreData(sCost);
    }
    StoreData("\n");
  }
}

/************************** NodeInfo ************************************************************
 Shows node information into the trace file
*************************************************************************************************/
public static void NodeInfo()
{
  int iCompN;
  String sNode, sCompN, sWeight, sCost, sDegree, sDepth;

      StoreData("\n ** Current Node Information ('*' indicates a root node) ** \n");
  StoreData("  Node#|Comp#|Weight|Degree|Depth \n");
  StoreData(" ------+-----+------+------+----- \n");
  for(int i = 0; i < nn; i++) {

    iCompN = FIND_COMP(i);

    sCompN  = String.valueOf(iCompN);
    sDegree = String.valueOf(Degree[i]);
    sDepth  = String.valueOf(Depth[i]);
    sNode   = String.valueOf(i);
    sWeight = String.valueOf(W[i]);

    sCompN  = WhiteSpace(2 - sCompN.length()) + sCompN;
    sDegree = WhiteSpace(5 - sDegree.length()) + sDegree;
    sDepth  = WhiteSpace(4 - sDepth.length()) + sDepth;
    sNode   = WhiteSpace(2 - sNode.length()) + sNode;
    sWeight = WhiteSpace(5 - sWeight.length()) + sWeight;

    if(i == root)
      StoreData(" *");
    else
      StoreData("  ");

    StoreData("n[" + sNode + "]|c[" + sCompN + "]| " + sWeight +
      "| " + sDegree + "| " + sDepth + "\n");
  }

  StoreData("\n");
}

/************************** CurrentSituation ****************************************************
 Shows current situation into the trace file
*************************************************************************************************/
public static void CurrentSituation()
{
  /**** Check Current Situation ****/
  int iNode, index, eA, eB,iNeigh, iNCompN;
  String sCompN,sNode,sCtr,sTrf,sCostRt,sWeight,
    sA,sB,sNeigh,sCost,sNCompN;

  StoreData("\n ** Current Component Information ** \n");
  StoreData("<Comp#|ConRt |Nearest|  Link :Cost|CostRt |Tradeoff|Weight|Nodes >\n");
  StoreData(" -----+------+-------+------------+-------+--------+------+------ \n");
```

```
for(int i = 0; i < C.size(); i++)  {
  if(((Component) C.elementAt(i)).Nodes.size() == 0)
    continue;

  sCompN  = String.valueOf(i);
  sCostRt = String.valueOf(((Component) C.elementAt(i)).BridgeCost[root]);
  sTrf    = String.valueOf(Tradeoff[i]);
  sWeight = String.valueOf(((Component) C.elementAt(i)).W);


  if(i == root)
    sCtr = " *c";
  else if(((Component) C.elementAt(i)).ConRt)
    sCtr = "  c";
  else
    sCtr = " nc";

  sCompN  = WhiteSpace(2 - sCompN.length()) + sCompN;
  sCostRt = WhiteSpace(4 - sCostRt.length()) + sCostRt;
  sTrf    = WhiteSpace(5 - sTrf.length()) + sTrf;
  sWeight = WhiteSpace(5 - sWeight.length()) + sWeight;

  if(i != root) {
    sNCompN = (String) ((Component) C.elementAt(i)).NComp.firstElement();
    iNCompN = Integer.parseInt(sNCompN);

    sNCompN = WhiteSpace(2 - sNCompN.length()) + sNCompN;
    link = ((Component) C.elementAt(i)).LinkTo(iNCompN);
    eA = link.end1;
    eB = link.end2;
    sCost = String.valueOf(link.cost);
    sA = String.valueOf(eA);
    sB = String.valueOf(eB);
  }
  else {
    sA = String.valueOf(i);
    sB = sA;
    sNCompN = sCompN;
    sCost = String.valueOf(Cost[i][i]);
  }

  sA = WhiteSpace(2 - sA.length()) + sA;
  sB = WhiteSpace(2 - sB.length()) + sB;
  sCost = WhiteSpace(4 - sCost.length()) + sCost;

  StoreData(" c[" + sCompN + "]| " + sCtr + "  | c[" + sNCompN + "] |(" +
    sA + "," + sB + "):" + sCost + "|  " +
    sCostRt + " |  " + sTrf + " | " + sWeight + "|");

  for(int n = 0; n < ((Component) C.elementAt(i)).Nodes.size(); n++)
    StoreData(" n[" + (String) ((Component) C.elementAt(i)).Nodes.elementAt(n) + "]");
  StoreData("\n");

}

for(int i = 0; i < C.size(); i++) {
  if(((Component) C.elementAt(i)).Nodes.size() == 0)
```

```
      continue;

    if(((Component) C.elementAt(i)).ID != root)  {
      StoreData("\nNeighbors of Component[" + i
        + "]: <Component#:(link:cost)>\n{");

      for(int c = 0; c < ((Component) C.elementAt(i)).NComp.size(); c++) {
        sNeigh = (String) ((Component) C.elementAt(i)).NComp.elementAt(c);
        iNeigh = Integer.parseInt(sNeigh);
        link = ((Component) C.elementAt(i)).LinkTo(iNeigh);
        sA = String.valueOf(link.end1);
        sB = String.valueOf(link.end2);
        sCost = String.valueOf(link.cost);

        sNeigh = WhiteSpace(2 - sNeigh.length()) + sNeigh;
        sA = WhiteSpace(2 - sA.length()) +  sA;
        sB = WhiteSpace(2 - sB.length()) + sB;
        sCost = WhiteSpace(4 - sCost.length()) + sCost;

        StoreData("<c[" + sNeigh + "]:(" + sA + "," + sB + "):" + sCost + ">");
        if(c + 1 == ((Component) C.elementAt(i)).NComp.size())
          StoreData("}\n");
        else if((c + 1)%4 == 0)
          StoreData("\n ");
        else
          StoreData(" ");
      }
    }
  }

  StoreData("\nA = {");
  for(index = 0; index < A.size(); index++) {
    eA = ((Link) A.elementAt(index)).end1;
    eB = ((Link) A.elementAt(index)).end2;
    StoreData("(" + eA + "," + eB + ") ");
    if((index + 1)%11 == 0)
      StoreData("\n     ");
  }
  StoreData("}\n");
}

/************************* WhiteSpace ********************************************************
 Returns a string consisting of the specified number of spaces.
*********************************************************************************************/
public static String WhiteSpace(int num)
{
  String space = "";
  switch(Math.abs(num)) {
  case 0: break;
  case 1: space = " "; break;
  case 2: space = "  "; break;
  case 3: space = "   "; break;
  case 4: space = "    "; break;
  case 5: space = "     "; break;
  default :
    space = WhiteSpace(Math.abs(num) - 5) + WhiteSpace(5);
  }
```

```
    return space;
  }
}

// Component Class
class Component
{
  int ID;              // Component ID
  int W;               // Weight of a component
  int BridgeCost[];    // Distance array of Neighbor Components
  boolean ConRt;       // Connection to a root
  Vector NComp,        // A set of Available Neighbor Components
    Links,             // A set of best links connecting Available Neighbor Components
    Nodes;             // A set of nodes in a component

  // Constructor
  public Component(int num) {
    ID = num;
    ConRt = false;
    Nodes = new Vector(0);
    NComp = new Vector(0);
    Links = new Vector(0);
    Nodes.addElement(String.valueOf(num));
  }

  // Returns the nearest available neighbor component
  public String NearestNComp() {
    return ((String) NComp.firstElement());
  }

  // Updates a link to other component by validating the smallest cost link
  public void UpdateLink(int cA, int cost, Link link) {
    BridgeCost[cA] = cost;
    Links.removeElementAt(cA);
    Links.insertElementAt(link, cA);
  }

  // Returns a link connetcting to a component, cT
  public Link LinkTo(int cT) {
    return ((Link) Links.elementAt(cT));
  }
}

// Link Class
class Link
{
  int end1;    // An end point of a link
  int end2;    // The other end point of the link
  int cost;    // Cost of the link

  public Link(int end1, int end2, int cost) {
    this.end1 = end1;
    this.end2 = end2;
    this.cost = cost;
  }
}
```

APPENDIX D

THE TRACE OF THE IMPROVED ESAU-WILLIAMS ALGORITHM

D.1 The CMST problem

```
The Number of Nodes : 11
Root Node : 3
Weight_limit : 9600
Order_limit : unconstrained
Degree_limit : unconstrained
Depth_limit : unconstrained
 *** <Cost Table> ***
```

Node	0	1	2	3	4	5	6	7	8	9	10
0	200	3434	3496	3366	2730	3148	1248	1604	806	2864	3404
1	3434	200	1210	432	2850	3510	2416	2034	3218	2882	3722
2	3496	1210	200	988	2112	2700	2634	2252	3110	2104	2876
3	3366	432	988	200	2636	2080	1304	1088	1456	1538	2350
4	2730	2850	2112	2636	200	2166	2718	2372	2976	1602	2322
5	3148	3510	2700	2080	2166	200	2362	2228	1946	546	1196
6	1248	2416	2634	1304	2718	2362	200	2426	3266	2164	2904
7	1604	2034	2252	1088	2372	2228	2426	200	1302	1294	2030
8	806	3218	3110	1456	2976	1946	3266	1302	200	3296	4102
9	2864	2882	2104	1538	1602	546	2164	1294	3296	200	3784
10	3404	3722	2876	2350	2322	1196	2904	2030	4102	3784	200

```
 ** Current Node Information ('*' indicates a root node) **
```

Node#	Comp#	Weight	Degree	Depth
n[0]	c[0]	2400	0	1
n[1]	c[1]	2400	0	1
n[2]	c[2]	2400	0	1
*n[3]	c[3]	69600	0	0
n[4]	c[4]	2400	0	1
n[5]	c[5]	2400	0	1
n[6]	c[6]	2400	0	1
n[7]	c[7]	2400	0	1
n[8]	c[8]	2400	0	1
n[9]	c[9]	2400	0	1
n[10]	c[10]	2400	0	1

```
 ** Current Component Information **
```

<Comp#	ConRt	Nearest	Link	:Cost	CostRt	Tradeoff	Weight	Nodes >
c[0]	nc	c[8]	(0, 8):	806	3366	-2560	2400	n[0]
c[1]	nc	c[3]	(1, 3):	432	432	0	2400	n[1]
c[2]	nc	c[3]	(2, 3):	988	988	0	2400	n[2]

```
 c[ 3]|  *c  | c[ 3] |( 3, 3): 200|   200 |      0 | 69600| n[3]
 c[ 4]|  nc  | c[ 9] |( 4, 9):1602|  2636 |  -1034 |  2400| n[4]
 c[ 5]|  nc  | c[ 9] |( 5, 9): 546|  2080 |  -1534 |  2400| n[5]
 c[ 6]|  nc  | c[ 0] |( 6, 0):1248|  1304 |    -56 |  2400| n[6]
 c[ 7]|  nc  | c[ 3] |( 7, 3):1088|  1088 |      0 |  2400| n[7]
 c[ 8]|  nc  | c[ 0] |( 8, 0): 806|  1456 |   -650 |  2400| n[8]
 c[ 9]|  nc  | c[ 5] |( 9, 5): 546|  1538 |   -992 |  2400| n[9]
 c[10]|  nc  | c[ 5] |(10, 5):1196|  2350 |  -1154 |  2400| n[10]

Neighbors of Component[0]: <Component#:(link:cost)>
{<c[ 8]:( 0, 8): 806> <c[ 3]:( 0, 3):3366> <c[ 1]:( 0, 1):3434> <c[ 2]:( 0, 2):3496>
 <c[ 4]:( 0, 4):2730> <c[ 5]:( 0, 5):3148> <c[ 6]:( 0, 6):1248> <c[ 7]:( 0, 7):1604>
 <c[ 9]:( 0, 9):2864> <c[10]:( 0,10):3404>}

Neighbors of Component[1]: <Component#:(link:cost)>
{<c[ 3]:( 1, 3): 432> <c[ 0]:( 1, 0):3434> <c[ 2]:( 1, 2):1210> <c[ 4]:( 1, 4):2850>
 <c[ 5]:( 1, 5):3510> <c[ 6]:( 1, 6):2416> <c[ 7]:( 1, 7):2034> <c[ 8]:( 1, 8):3218>
 <c[ 9]:( 1, 9):2882> <c[10]:( 1,10):3722>}

Neighbors of Component[2]: <Component#:(link:cost)>
{<c[ 3]:( 2, 3): 988> <c[ 0]:( 2, 0):3496> <c[ 1]:( 2, 1):1210> <c[ 4]:( 2, 4):2112>
 <c[ 5]:( 2, 5):2700> <c[ 6]:( 2, 6):2634> <c[ 7]:( 2, 7):2252> <c[ 8]:( 2, 8):3110>
 <c[ 9]:( 2, 9):2104> <c[10]:( 2,10):2876>}

Neighbors of Component[4]: <Component#:(link:cost)>
{<c[ 9]:( 4, 9):1602> <c[ 3]:( 4, 3):2636> <c[ 0]:( 4, 0):2730> <c[ 1]:( 4, 1):2850>
 <c[ 2]:( 4, 2):2112> <c[ 5]:( 4, 5):2166> <c[ 6]:( 4, 6):2718> <c[ 7]:( 4, 7):2372>
 <c[ 8]:( 4, 8):2976> <c[10]:( 4,10):2322>}

Neighbors of Component[5]: <Component#:(link:cost)>
{<c[ 9]:( 5, 9): 546> <c[ 3]:( 5, 3):2080> <c[ 0]:( 5, 0):3148> <c[ 1]:( 5, 1):3510>
 <c[ 2]:( 5, 2):2700> <c[ 4]:( 5, 4):2166> <c[ 6]:( 5, 6):2362> <c[ 7]:( 5, 7):2228>
 <c[ 8]:( 5, 8):1946> <c[10]:( 5,10):1196>}

Neighbors of Component[6]: <Component#:(link:cost)>
{<c[ 0]:( 6, 0):1248> <c[ 3]:( 6, 3):1304> <c[ 1]:( 6, 1):2416> <c[ 2]:( 6, 2):2634>
 <c[ 4]:( 6, 4):2718> <c[ 5]:( 6, 5):2362> <c[ 7]:( 6, 7):2426> <c[ 8]:( 6, 8):3266>
 <c[ 9]:( 6, 9):2164> <c[10]:( 6,10):2904>}

Neighbors of Component[7]: <Component#:(link:cost)>
{<c[ 3]:( 7, 3):1088> <c[ 0]:( 7, 0):1604> <c[ 1]:( 7, 1):2034> <c[ 2]:( 7, 2):2252>
 <c[ 4]:( 7, 4):2372> <c[ 5]:( 7, 5):2228> <c[ 6]:( 7, 6):2426> <c[ 8]:( 7, 8):1302>
 <c[ 9]:( 7, 9):1294> <c[10]:( 7,10):2030>}

Neighbors of Component[8]: <Component#:(link:cost)>
{<c[ 0]:( 8, 0): 806> <c[ 3]:( 8, 3):1456> <c[ 1]:( 8, 1):3218> <c[ 2]:( 8, 2):3110>
 <c[ 4]:( 8, 4):2976> <c[ 5]:( 8, 5):1946> <c[ 6]:( 8, 6):3266> <c[ 7]:( 8, 7):1302>
 <c[ 9]:( 8, 9):3296> <c[10]:( 8,10):4102>}

Neighbors of Component[9]: <Component#:(link:cost)>
{<c[ 5]:( 9, 5): 546> <c[ 3]:( 9, 3):1538> <c[ 0]:( 9, 0):2864> <c[ 1]:( 9, 1):2882>
 <c[ 2]:( 9, 2):2104> <c[ 4]:( 9, 4):1602> <c[ 6]:( 9, 6):2164> <c[ 7]:( 9, 7):1294>
 <c[ 8]:( 9, 8):3296> <c[10]:( 9,10):3784>}

Neighbors of Component[10]: <Component#:(link:cost)>
{<c[ 5]:(10, 5):1196> <c[ 3]:(10, 3):2350> <c[ 0]:(10, 0):3404> <c[ 1]:(10, 1):3722>
 <c[ 2]:(10, 2):2876> <c[ 4]:(10, 4):2322> <c[ 6]:(10, 6):2904> <c[ 7]:(10, 7):2030>
 <c[ 8]:(10, 8):4102> <c[ 9]:(10, 9):3784>}
```

```
A = {}

 *** Start EW Algorithm ***

1): The smallest Tradeoff(c[0])=-2560
   Merge Comp(0) and Comp(8) by a link(0,8)

 Link(8,1) is the representative link between c[1] and c[8]
 Update Tradeoff(c[1]) = 432 - 432 = 0
 Link(8,2) is the representative link between c[2] and c[8]
 Update Tradeoff(c[2]) = 988 - 988 = 0
 Link(0,4) is the representative link between c[4] and c[8]
 Update Tradeoff(c[4]) = 1602 - 2636 = -1034
 Link(8,5) is the representative link between c[5] and c[8]
 Update Tradeoff(c[5]) = 546 - 2080 = -1534
 Link(0,6) is the representative link between c[6] and c[8]
 Update Tradeoff(c[6]) = 1248 - 1304 = -56
 Link(8,7) is the representative link between c[7] and c[8]
 Update Tradeoff(c[7]) = 1088 - 1088 = 0
 Link(0,9) is the representative link between c[9] and c[8]
 Update Tradeoff(c[9]) = 546 - 1538 = -992
 Link(0,10) is the representative link between c[10] and c[8]
 Update Tradeoff(c[10]) = 1196 - 2350 = -1154
 Update Tradeoff(c[8]) = 1248 - 1456 = -208

 ** Current Component Information **
<Comp#|ConRt |Nearest|  Link :Cost|CostRt |Tradeoff|Weight|Nodes >
 -----+------+-------+------------+-------+--------+------+------
 c[ 1]|  nc  | c[ 3] |( 1, 3): 432|   432 |      0 |  2400| n[1]
 c[ 2]|  nc  | c[ 3] |( 2, 3): 988|   988 |      0 |  2400| n[2]
 c[ 3]|  *c  | c[ 3] |( 3, 3): 200|   200 |      0 | 69600| n[3]
 c[ 4]|  nc  | c[ 9] |( 4, 9):1602|  2636 |  -1034 |  2400| n[4]
 c[ 5]|  nc  | c[ 9] |( 5, 9): 546|  2080 |  -1534 |  2400| n[5]
 c[ 6]|  nc  | c[ 8] |( 0, 6):1248|  1304 |    -56 |  2400| n[6]
 c[ 7]|  nc  | c[ 3] |( 7, 3):1088|  1088 |      0 |  2400| n[7]
 c[ 8]|  nc  | c[ 6] |( 0, 6):1248|  1456 |   -208 |  4800| n[8] n[0]
 c[ 9]|  nc  | c[ 5] |( 9, 5): 546|  1538 |   -992 |  2400| n[9]
 c[10]|  nc  | c[ 5] |(10, 5):1196|  2350 |  -1154 |  2400| n[10]

Neighbors of Component[1]: <Component#:(link:cost)>
{<c[ 3]:( 1, 3): 432> <c[ 2]:( 1, 2):1210> <c[ 4]:( 1, 4):2850> <c[ 5]:( 1, 5):3510>
 <c[ 6]:( 1, 6):2416> <c[ 7]:( 1, 7):2034> <c[ 8]:( 1, 8):3218> <c[ 9]:( 1, 9):2882>
 <c[10]:( 1,10):3722>}

Neighbors of Component[2]: <Component#:(link:cost)>
{<c[ 3]:( 2, 3): 988> <c[ 1]:( 2, 1):1210> <c[ 4]:( 2, 4):2112> <c[ 5]:( 2, 5):2700>
 <c[ 6]:( 2, 6):2634> <c[ 7]:( 2, 7):2252> <c[ 8]:( 2, 8):3110> <c[ 9]:( 2, 9):2104>
 <c[10]:( 2,10):2876>}

Neighbors of Component[4]: <Component#:(link:cost)>
{<c[ 9]:( 4, 9):1602> <c[ 3]:( 4, 3):2636> <c[ 1]:( 4, 1):2850> <c[ 2]:( 4, 2):2112>
 <c[ 5]:( 4, 5):2166> <c[ 6]:( 4, 6):2718> <c[ 7]:( 4, 7):2372> <c[ 8]:( 0, 4):2730>
 <c[10]:( 4,10):2322>}

Neighbors of Component[5]: <Component#:(link:cost)>
{<c[ 9]:( 5, 9): 546> <c[ 3]:( 5, 3):2080> <c[ 1]:( 5, 1):3510> <c[ 2]:( 5, 2):2700>
 <c[ 4]:( 5, 4):2166> <c[ 6]:( 5, 6):2362> <c[ 7]:( 5, 7):2228> <c[ 8]:( 5, 8):1946>
```

```
 <c[10]:( 5,10):1196>}

Neighbors of Component[6]: <Component#:(link:cost)>
{<c[ 8]:( 0, 6):1248> <c[ 3]:( 6, 3):1304> <c[ 1]:( 6, 1):2416> <c[ 2]:( 6, 2):2634>
 <c[ 4]:( 6, 4):2718> <c[ 5]:( 6, 5):2362> <c[ 7]:( 6, 7):2426> <c[ 9]:( 6, 9):2164>
 <c[10]:( 6,10):2904>}

Neighbors of Component[7]: <Component#:(link:cost)>
{<c[ 3]:( 7, 3):1088> <c[ 1]:( 7, 1):2034> <c[ 2]:( 7, 2):2252> <c[ 4]:( 7, 4):2372>
 <c[ 5]:( 7, 5):2228> <c[ 6]:( 7, 6):2426> <c[ 8]:( 7, 8):1302> <c[ 9]:( 7, 9):1294>
 <c[10]:( 7,10):2030>}

Neighbors of Component[8]: <Component#:(link:cost)>
{<c[ 6]:( 0, 6):1248> <c[ 3]:( 8, 3):1456> <c[ 1]:( 1, 8):3218> <c[ 2]:( 2, 8):3110>
 <c[ 4]:( 0, 4):2730> <c[ 5]:( 5, 8):1946> <c[ 7]:( 7, 8):1302> <c[ 9]:( 0, 9):2864>
 <c[10]:( 0,10):3404>}

Neighbors of Component[9]: <Component#:(link:cost)>
{<c[ 5]:( 9, 5): 546> <c[ 3]:( 9, 3):1538> <c[ 1]:( 9, 1):2882> <c[ 2]:( 9, 2):2104>
 <c[ 4]:( 9, 4):1602> <c[ 6]:( 9, 6):2164> <c[ 7]:( 9, 7):1294> <c[ 8]:( 0, 9):2864>
 <c[10]:( 9,10):3784>}

Neighbors of Component[10]: <Component#:(link:cost)>
{<c[ 5]:(10, 5):1196> <c[ 3]:(10, 3):2350> <c[ 1]:(10, 1):3722> <c[ 2]:(10, 2):2876>
 <c[ 4]:(10, 4):2322> <c[ 6]:(10, 6):2904> <c[ 7]:(10, 7):2030> <c[ 8]:( 0,10):3404>
 <c[ 9]:(10, 9):3784>}

A = {(8,0) }

 ** Current Node Information ('*' indicates a root node) **
  Node#|Comp#|Weight|Degree|Depth
 ------+-----+------+------+-----
  n[ 0]|c[ 8]|  2400|     1|    2
  n[ 1]|c[ 1]|  2400|     0|    1
  n[ 2]|c[ 2]|  2400|     0|    1
 *n[ 3]|c[ 3]| 69600|     0|    0
  n[ 4]|c[ 4]|  2400|     0|    1
  n[ 5]|c[ 5]|  2400|     0|    1
  n[ 6]|c[ 6]|  2400|     0|    1
  n[ 7]|c[ 7]|  2400|     0|    1
  n[ 8]|c[ 8]|  2400|     1|    1
  n[ 9]|c[ 9]|  2400|     0|    1
  n[10]|c[10]|  2400|     0|    1

2): The smallest Tradeoff(c[5])=-1534
   Merge Comp(5) and Comp(9) by a link(5,9)

 Link(9,1) is the representative link between c[1] and c[9]
 Update Tradeoff(c[1]) = 432 - 432 = 0
 Link(9,2) is the representative link between c[2] and c[9]
 Update Tradeoff(c[2]) = 988 - 988 = 0
 Link(9,4) is the representative link between c[4] and c[9]
 Update Tradeoff(c[4]) = 1602 - 2636 = -1034
 Link(9,6) is the representative link between c[6] and c[9]
 Update Tradeoff(c[6]) = 1248 - 1304 = -56
 Link(9,7) is the representative link between c[7] and c[9]
 Update Tradeoff(c[7]) = 1088 - 1088 = 0
 Link(5,8) is the representative link between c[8] and c[9]
```

```
Update Tradeoff(c[8]) = 1248 - 1456 = -208
Link(5,10) is the representative link between c[10] and c[9]
Update Tradeoff(c[10]) = 1196 - 2350 = -1154
Update Tradeoff(c[9]) = 1196 - 1538 = -342

** Current Component Information **
<Comp#|ConRt |Nearest|  Link :Cost|CostRt |Tradeoff|Weight|Nodes >
-----+------+-------+------------+-------+--------+------+------
c[ 1]|  nc  | c[ 3] |( 1, 3): 432|   432 |      0 |  2400| n[1]
c[ 2]|  nc  | c[ 3] |( 2, 3): 988|   988 |      0 |  2400| n[2]
c[ 3]|  *c  | c[ 3] |( 3, 3): 200|   200 |      0 | 69600| n[3]
c[ 4]|  nc  | c[ 9] |( 4, 9):1602|  2636 |  -1034 |  2400| n[4]
c[ 6]|  nc  | c[ 8] |( 0, 6):1248|  1304 |    -56 |  2400| n[6]
c[ 7]|  nc  | c[ 3] |( 7, 3):1088|  1088 |      0 |  2400| n[7]
c[ 8]|  nc  | c[ 6] |( 0, 6):1248|  1456 |   -208 |  4800| n[8] n[0]
c[ 9]|  nc  | c[10] |( 5,10):1196|  1538 |   -342 |  4800| n[9] n[5]
c[10]|  nc  | c[ 9] |( 5,10):1196|  2350 |  -1154 |  2400| n[10]

Neighbors of Component[1]: <Component#:(link:cost)>
{<c[ 3]:( 1, 3): 432> <c[ 2]:( 1, 2):1210> <c[ 4]:( 1, 4):2850> <c[ 6]:( 1, 6):2416>
 <c[ 7]:( 1, 7):2034> <c[ 8]:( 1, 8):3218> <c[ 9]:( 1, 9):2882> <c[10]:( 1,10):3722>}

Neighbors of Component[2]: <Component#:(link:cost)>
{<c[ 3]:( 2, 3): 988> <c[ 1]:( 2, 1):1210> <c[ 4]:( 2, 4):2112> <c[ 6]:( 2, 6):2634>
 <c[ 7]:( 2, 7):2252> <c[ 8]:( 2, 8):3110> <c[ 9]:( 2, 9):2104> <c[10]:( 2,10):2876>}

Neighbors of Component[4]: <Component#:(link:cost)>
{<c[ 9]:( 4, 9):1602> <c[ 3]:( 4, 3):2636> <c[ 1]:( 4, 1):2850> <c[ 2]:( 4, 2):2112>
 <c[ 6]:( 4, 6):2718> <c[ 7]:( 4, 7):2372> <c[ 8]:( 0, 4):2730> <c[10]:( 4,10):2322>}

Neighbors of Component[6]: <Component#:(link:cost)>
{<c[ 8]:( 0, 6):1248> <c[ 3]:( 6, 3):1304> <c[ 1]:( 6, 1):2416> <c[ 2]:( 6, 2):2634>
 <c[ 4]:( 6, 4):2718> <c[ 7]:( 6, 7):2426> <c[ 9]:( 6, 9):2164> <c[10]:( 6,10):2904>}

Neighbors of Component[7]: <Component#:(link:cost)>
{<c[ 3]:( 7, 3):1088> <c[ 1]:( 7, 1):2034> <c[ 2]:( 7, 2):2252> <c[ 4]:( 7, 4):2372>
 <c[ 6]:( 7, 6):2426> <c[ 8]:( 7, 8):1302> <c[ 9]:( 7, 9):1294> <c[10]:( 7,10):2030>}

Neighbors of Component[8]: <Component#:(link:cost)>
{<c[ 6]:( 0, 6):1248> <c[ 3]:( 8, 3):1456> <c[ 1]:( 1, 8):3218> <c[ 2]:( 2, 8):3110>
 <c[ 4]:( 0, 4):2730> <c[ 7]:( 7, 8):1302> <c[ 9]:( 5, 8):1946> <c[10]:( 0,10):3404>}

Neighbors of Component[9]: <Component#:(link:cost)>
{<c[10]:( 5,10):1196> <c[ 3]:( 9, 3):1538> <c[ 1]:( 1, 9):2882> <c[ 2]:( 2, 9):2104>
 <c[ 4]:( 4, 9):1602> <c[ 6]:( 6, 9):2164> <c[ 7]:( 7, 9):1294> <c[ 8]:( 5, 8):1946>}

Neighbors of Component[10]: <Component#:(link:cost)>
{<c[ 9]:( 5,10):1196> <c[ 3]:(10, 3):2350> <c[ 1]:(10, 1):3722> <c[ 2]:(10, 2):2876>
 <c[ 4]:(10, 4):2322> <c[ 6]:(10, 6):2904> <c[ 7]:(10, 7):2030> <c[ 8]:( 0,10):3404>}

A = {(8,0) (9,5) }

 ** Current Node Information ('*' indicates a root node) **
  Node#|Comp#|Weight|Degree|Depth
 ------+-----+------+------+-----
  n[ 0]|c[ 8]|  2400|     1|    2
  n[ 1]|c[ 1]|  2400|     0|    1
  n[ 2]|c[ 2]|  2400|     0|    1
```

```
 *n[ 3]|c[ 3]| 69600|     0|    0
  n[ 4]|c[ 4]|  2400|     0|    1
  n[ 5]|c[ 9]|  2400|     1|    2
  n[ 6]|c[ 6]|  2400|     0|    1
  n[ 7]|c[ 7]|  2400|     0|    1
  n[ 8]|c[ 8]|  2400|     1|    1
  n[ 9]|c[ 9]|  2400|     1|    1
  n[10]|c[10]|  2400|     0|    1

3): The smallest Tradeoff(c[10])=-1154
   Merge Comp(10) and Comp(9) by a link(5,10)

 Link(9,1) is the representative link between c[1] and c[9]
 Update Tradeoff(c[1]) = 432 - 432 = 0
 Link(9,2) is the representative link between c[2] and c[9]
 Update Tradeoff(c[2]) = 988 - 988 = 0
 Link(9,4) is the representative link between c[4] and c[9]
 Update Tradeoff(c[4]) = 1602 - 2636 = -1034
 Link(9,6) is the representative link between c[6] and c[9]
 Update Tradeoff(c[6]) = 1248 - 1304 = -56
 Link(9,7) is the representative link between c[7] and c[9]
 Update Tradeoff(c[7]) = 1088 - 1088 = 0
 Merging c[9] and c[8]) would violate the weight-constraint.
 Destory a relation between Comp(9) and Comp(8)
 Update Tradeoff(c[8]) = 1248 - 1456 = -208
 Update Tradeoff(c[9]) = 1294 - 1538 = -244

 ** Current Component Information **
<Comp#|ConRt |Nearest|  Link :Cost|CostRt |Tradeoff|Weight|Nodes >
 -----+------+-------+------------+-------+--------+------+------
 c[ 1]|  nc  | c[ 3] |( 1, 3): 432|   432 |      0 |  2400| n[1]
 c[ 2]|  nc  | c[ 3] |( 2, 3): 988|   988 |      0 |  2400| n[2]
 c[ 3]|  *c  | c[ 3] |( 3, 3): 200|   200 |      0 | 69600| n[3]
 c[ 4]|  nc  | c[ 9] |( 4, 9):1602|  2636 |  -1034 |  2400| n[4]
 c[ 6]|  nc  | c[ 8] |( 0, 6):1248|  1304 |    -56 |  2400| n[6]
 c[ 7]|  nc  | c[ 3] |( 7, 3):1088|  1088 |      0 |  2400| n[7]
 c[ 8]|  nc  | c[ 6] |( 0, 6):1248|  1456 |   -208 |  4800| n[8] n[0]
 c[ 9]|  nc  | c[ 7] |( 7, 9):1294|  1538 |   -244 |  7200| n[9] n[5] n[10]

Neighbors of Component[1]: <Component#:(link:cost)>
{<c[ 3]:( 1, 3): 432> <c[ 2]:( 1, 2):1210> <c[ 4]:( 1, 4):2850> <c[ 6]:( 1, 6):2416>
 <c[ 7]:( 1, 7):2034> <c[ 8]:( 1, 8):3218> <c[ 9]:( 1, 9):2882>}

Neighbors of Component[2]: <Component#:(link:cost)>
{<c[ 3]:( 2, 3): 988> <c[ 1]:( 2, 1):1210> <c[ 4]:( 2, 4):2112> <c[ 6]:( 2, 6):2634>
 <c[ 7]:( 2, 7):2252> <c[ 8]:( 2, 8):3110> <c[ 9]:( 2, 9):2104>}

Neighbors of Component[4]: <Component#:(link:cost)>
{<c[ 9]:( 4, 9):1602> <c[ 3]:( 4, 3):2636> <c[ 1]:( 4, 1):2850> <c[ 2]:( 4, 2):2112>
 <c[ 6]:( 4, 6):2718> <c[ 7]:( 4, 7):2372> <c[ 8]:( 0, 4):2730>}

Neighbors of Component[6]: <Component#:(link:cost)>
{<c[ 8]:( 0, 6):1248> <c[ 3]:( 6, 3):1304> <c[ 1]:( 6, 1):2416> <c[ 2]:( 6, 2):2634>
 <c[ 4]:( 6, 4):2718> <c[ 7]:( 6, 7):2426> <c[ 9]:( 6, 9):2164>}

Neighbors of Component[7]: <Component#:(link:cost)>
{<c[ 3]:( 7, 3):1088> <c[ 1]:( 7, 1):2034> <c[ 2]:( 7, 2):2252> <c[ 4]:( 7, 4):2372>
 <c[ 6]:( 7, 6):2426> <c[ 8]:( 7, 8):1302> <c[ 9]:( 7, 9):1294>}
```

```
Neighbors of Component[8]: <Component#:(link:cost)>
{<c[ 6]:( 0, 6):1248> <c[ 3]:( 8, 3):1456> <c[ 1]:( 1, 8):3218> <c[ 2]:( 2, 8):3110>
 <c[ 4]:( 0, 4):2730> <c[ 7]:( 7, 8):1302>}

Neighbors of Component[9]: <Component#:(link:cost)>
{<c[ 7]:( 7, 9):1294> <c[ 3]:( 9, 3):1538> <c[ 1]:( 1, 9):2882> <c[ 2]:( 2, 9):2104>
 <c[ 4]:( 4, 9):1602> <c[ 6]:( 6, 9):2164>}

A = {(8,0) (9,5) (5,10) }

 ** Current Node Information ('*' indicates a root node) **
  Node#|Comp#|Weight|Degree|Depth
 ------+-----+------+------+-----
  n[ 0]|c[ 8]|  2400|     1|    2
  n[ 1]|c[ 1]|  2400|     0|    1
  n[ 2]|c[ 2]|  2400|     0|    1
 *n[ 3]|c[ 3]| 69600|     0|    0
  n[ 4]|c[ 4]|  2400|     0|    1
  n[ 5]|c[ 9]|  2400|     2|    2
  n[ 6]|c[ 6]|  2400|     0|    1
  n[ 7]|c[ 7]|  2400|     0|    1
  n[ 8]|c[ 8]|  2400|     1|    1
  n[ 9]|c[ 9]|  2400|     1|    1
  n[10]|c[ 9]|  2400|     1|    3

4): The smallest Tradeoff(c[4])=-1034
   Merge Comp(4) and Comp(9) by a link(4,9)

 Update Tradeoff(c[8]) = 1248 - 1456 = -208
 Merging c[9] and c[7]) would violate the weight-constraint.
 Destory a relation between Comp(9) and Comp(7)
 Update Tradeoff(c[7]) = 1088 - 1088 = 0
 Merging c[9] and c[1]) would violate the weight-constraint.
 Destory a relation between Comp(9) and Comp(1)
 Update Tradeoff(c[1]) = 432 - 432 = 0
 Merging c[9] and c[2]) would violate the weight-constraint.
 Destory a relation between Comp(9) and Comp(2)
 Update Tradeoff(c[2]) = 988 - 988 = 0
 Merging c[9] and c[6]) would violate the weight-constraint.
 Destory a relation between Comp(9) and Comp(6)
 Update Tradeoff(c[6]) = 1248 - 1304 = -56
 Update Tradeoff(c[9]) = 1538 - 1538 = 0

 ** Current Component Information **
<Comp#|ConRt |Nearest|  Link :Cost|CostRt |Tradeoff|Weight|Nodes >
 -----+------+-------+-----------+-------+--------+------+------
 c[ 1]|  nc  | c[ 3] |( 1, 3): 432|   432 |      0 |  2400| n[1]
 c[ 2]|  nc  | c[ 3] |( 2, 3): 988|   988 |      0 |  2400| n[2]
 c[ 3]|  *c  | c[ 3] |( 3, 3): 200|   200 |      0 | 69600| n[3]
 c[ 6]|  nc  | c[ 8] |( 0, 6):1248|  1304 |    -56 |  2400| n[6]
 c[ 7]|  nc  | c[ 3] |( 7, 3):1088|  1088 |      0 |  2400| n[7]
 c[ 8]|  nc  | c[ 6] |( 0, 6):1248|  1456 |   -208 |  4800| n[8] n[0]
 c[ 9]|  nc  | c[ 3] |( 9, 3):1538|  1538 |      0 |  9600| n[9] n[5] n[10] n[4]

Neighbors of Component[1]: <Component#:(link:cost)>
{<c[ 3]:( 1, 3): 432> <c[ 2]:( 1, 2):1210> <c[ 6]:( 1, 6):2416> <c[ 7]:( 1, 7):2034>
 <c[ 8]:( 1, 8):3218>}
```

```
Neighbors of Component[2]: <Component#:(link:cost)>
{<c[ 3]:( 2, 3): 988> <c[ 1]:( 2, 1):1210> <c[ 6]:( 2, 6):2634> <c[ 7]:( 2, 7):2252>
 <c[ 8]:( 2, 8):3110>}

Neighbors of Component[6]: <Component#:(link:cost)>
{<c[ 8]:( 0, 6):1248> <c[ 3]:( 6, 3):1304> <c[ 1]:( 6, 1):2416> <c[ 2]:( 6, 2):2634>
 <c[ 7]:( 6, 7):2426>}

Neighbors of Component[7]: <Component#:(link:cost)>
{<c[ 3]:( 7, 3):1088> <c[ 1]:( 7, 1):2034> <c[ 2]:( 7, 2):2252> <c[ 6]:( 7, 6):2426>
 <c[ 8]:( 7, 8):1302>}

Neighbors of Component[8]: <Component#:(link:cost)>
{<c[ 6]:( 0, 6):1248> <c[ 3]:( 8, 3):1456> <c[ 1]:( 1, 8):3218> <c[ 2]:( 2, 8):3110>
 <c[ 7]:( 7, 8):1302>}

Neighbors of Component[9]: <Component#:(link:cost)>
{<c[ 3]:( 9, 3):1538>}

A = {(8,0) (9,5) (5,10) (9,4) }

 ** Current Node Information ('*' indicates a root node) **
  Node#|Comp#|Weight|Degree|Depth
 ------+-----+------+------+-----
  n[ 0]|c[ 8]|  2400|     1|    2
  n[ 1]|c[ 1]|  2400|     0|    1
  n[ 2]|c[ 2]|  2400|     0|    1
 *n[ 3]|c[ 3]| 69600|     0|    0
  n[ 4]|c[ 9]|  2400|     1|    2
  n[ 5]|c[ 9]|  2400|     2|    2
  n[ 6]|c[ 6]|  2400|     0|    1
  n[ 7]|c[ 7]|  2400|     0|    1
  n[ 8]|c[ 8]|  2400|     1|    1
  n[ 9]|c[ 9]|  2400|     2|    1
  n[10]|c[ 9]|  2400|     1|    3

5): The smallest Tradeoff(c[8])=-208
   Merge Comp(8) and Comp(6) by a link(0,6)

 Link(6,1) is the representative link between c[1] and c[6]
 Update Tradeoff(c[1]) = 432 - 432 = 0
 Link(6,2) is the representative link between c[2] and c[6]
 Update Tradeoff(c[2]) = 988 - 988 = 0
 Link(8,7) is the representative link between c[7] and c[6]
 Update Tradeoff(c[7]) = 1088 - 1088 = 0
 Update Tradeoff(c[6]) = 1302 - 1304 = -2

 ** Current Component Information **
<Comp#|ConRt |Nearest|  Link :Cost|CostRt |Tradeoff|Weight|Nodes >
 -----+------+-------+-----------+-------+--------+------+-----
 c[ 1]|  nc  | c[ 3] |( 1, 3): 432|   432 |      0 |  2400| n[1]
 c[ 2]|  nc  | c[ 3] |( 2, 3): 988|   988 |      0 |  2400| n[2]
 c[ 3]|  *c  | c[ 3] |( 3, 3): 200|   200 |      0 | 69600| n[3]
 c[ 6]|  nc  | c[ 7] |( 7, 8):1302|  1304 |     -2 |  7200| n[6] n[0] n[8]
 c[ 7]|  nc  | c[ 3] |( 7, 3):1088|  1088 |      0 |  2400| n[7]
 c[ 9]|  nc  | c[ 3] |( 9, 3):1538|  1538 |      0 |  9600| n[9] n[5] n[10] n[4]
```

```
Neighbors of Component[1]: <Component#:(link:cost)>
{<c[ 3]:( 1, 3): 432> <c[ 2]:( 1, 2):1210> <c[ 6]:( 1, 6):2416> <c[ 7]:( 1, 7):2034>}

Neighbors of Component[2]: <Component#:(link:cost)>
{<c[ 3]:( 2, 3): 988> <c[ 1]:( 2, 1):1210> <c[ 6]:( 2, 6):2634> <c[ 7]:( 2, 7):2252>}

Neighbors of Component[6]: <Component#:(link:cost)>
{<c[ 7]:( 7, 8):1302> <c[ 3]:( 6, 3):1304> <c[ 1]:( 1, 6):2416> <c[ 2]:( 2, 6):2634>}

Neighbors of Component[7]: <Component#:(link:cost)>
{<c[ 3]:( 7, 3):1088> <c[ 1]:( 7, 1):2034> <c[ 2]:( 7, 2):2252> <c[ 6]:( 7, 8):1302>}

Neighbors of Component[9]: <Component#:(link:cost)>
{<c[ 3]:( 9, 3):1538>}

A = {(8,0) (9,5) (5,10) (9,4) (6,0) }

 ** Current Node Information ('*' indicates a root node) **
  Node#|Comp#|Weight|Degree|Depth
 ------+-----+------+------+-----
  n[ 0]|c[ 6]|  2400|     2|    2
  n[ 1]|c[ 1]|  2400|     0|    1
  n[ 2]|c[ 2]|  2400|     0|    1
 *n[ 3]|c[ 3]| 69600|     0|    0
  n[ 4]|c[ 9]|  2400|     1|    2
  n[ 5]|c[ 9]|  2400|     2|    2
  n[ 6]|c[ 6]|  2400|     1|    1
  n[ 7]|c[ 7]|  2400|     0|    1
  n[ 8]|c[ 6]|  2400|     1|    3
  n[ 9]|c[ 9]|  2400|     2|    1
  n[10]|c[ 9]|  2400|     1|    3

6): The smallest Tradeoff(c[6])=-2
   Merge Comp(6) and Comp(7) by a link(7,8)

 Merging c[7] and c[1]) would violate the weight-constraint.
 Destory a relation between Comp(7) and Comp(1)
 Update Tradeoff(c[1]) = 432 - 432 = 0
 Merging c[7] and c[2]) would violate the weight-constraint.
 Destory a relation between Comp(7) and Comp(2)
 Update Tradeoff(c[2]) = 988 - 988 = 0
 Update Tradeoff(c[7]) = 1088 - 1088 = 0

 ** Current Component Information **
<Comp#|ConRt |Nearest|  Link :Cost|CostRt |Tradeoff|Weight|Nodes >
 -----+------+-------+------------+-------+--------+------+------
 c[ 1]|  nc  | c[ 3] |( 1, 3): 432|   432 |      0 |  2400| n[1]
 c[ 2]|  nc  | c[ 3] |( 2, 3): 988|   988 |      0 |  2400| n[2]
 c[ 3]|  *c  | c[ 3] |( 3, 3): 200|   200 |      0 | 69600| n[3]
 c[ 7]|  nc  | c[ 3] |( 7, 3):1088|  1088 |      0 |  9600| n[7] n[8] n[0] n[6]
 c[ 9]|  nc  | c[ 3] |( 9, 3):1538|  1538 |      0 |  9600| n[9] n[5] n[10] n[4]

Neighbors of Component[1]: <Component#:(link:cost)>
{<c[ 3]:( 1, 3): 432> <c[ 2]:( 1, 2):1210>}

Neighbors of Component[2]: <Component#:(link:cost)>
{<c[ 3]:( 2, 3): 988> <c[ 1]:( 2, 1):1210>}
```

```
Neighbors of Component[7]: <Component#:(link:cost)>
{<c[ 3]:( 7, 3):1088>}

Neighbors of Component[9]: <Component#:(link:cost)>
{<c[ 3]:( 9, 3):1538>}

A = {(8,0) (9,5) (5,10) (9,4) (6,0) (7,8) }

 ** Current Node Information ('*' indicates a root node) **
  Node#|Comp#|Weight|Degree|Depth
 ------+-----+------+------+-----
  n[ 0]|c[ 7]|  2400|     2|    3
  n[ 1]|c[ 1]|  2400|     0|    1
  n[ 2]|c[ 2]|  2400|     0|    1
 *n[ 3]|c[ 3]| 69600|     0|    0
  n[ 4]|c[ 9]|  2400|     1|    2
  n[ 5]|c[ 9]|  2400|     2|    2
  n[ 6]|c[ 7]|  2400|     1|    4
  n[ 7]|c[ 7]|  2400|     1|    1
  n[ 8]|c[ 7]|  2400|     2|    2
  n[ 9]|c[ 9]|  2400|     2|    1
  n[10]|c[ 9]|  2400|     1|    3

7): The smallest Tradeoff(c[1])=0
   Merge Comp(1) and Comp(3) by a link(1,3)


 ** Current Component Information **
<Comp#|ConRt |Nearest|  Link :Cost|CostRt |Tradeoff|Weight|Nodes >
 -----+------+-------+-----------+-------+--------+------+------
 c[ 1]|   c  | c[ 3] |( 1, 3): 432|   432 |      0 |  2400| n[1]
 c[ 2]|  nc  | c[ 3] |( 2, 3): 988|   988 |      0 |  2400| n[2]
 c[ 3]|  *c  | c[ 3] |( 3, 3): 200|   200 |      0 | 69600| n[3]
 c[ 7]|  nc  | c[ 3] |( 7, 3):1088|  1088 |      0 |  9600| n[7] n[8] n[0] n[6]
 c[ 9]|  nc  | c[ 3] |( 9, 3):1538|  1538 |      0 |  9600| n[9] n[5] n[10] n[4]

Neighbors of Component[1]: <Component#:(link:cost)>
{<c[ 3]:( 1, 3): 432> <c[ 2]:( 1, 2):1210>}

Neighbors of Component[2]: <Component#:(link:cost)>
{<c[ 3]:( 2, 3): 988> <c[ 1]:( 2, 1):1210>}

Neighbors of Component[7]: <Component#:(link:cost)>
{<c[ 3]:( 7, 3):1088>}

Neighbors of Component[9]: <Component#:(link:cost)>
{<c[ 3]:( 9, 3):1538>}

A = {(8,0) (9,5) (5,10) (9,4) (6,0) (7,8) (1,3) }

 ** Current Node Information ('*' indicates a root node) **
  Node#|Comp#|Weight|Degree|Depth
 ------+-----+------+------+-----
  n[ 0]|c[ 7]|  2400|     2|    3
  n[ 1]|c[ 1]|  2400|     1|    1
  n[ 2]|c[ 2]|  2400|     0|    1
 *n[ 3]|c[ 3]| 69600|     1|    0
  n[ 4]|c[ 9]|  2400|     1|    2
```

```
 n[ 5]|c[ 9]|  2400|     2|    2
 n[ 6]|c[ 7]|  2400|     1|    4
 n[ 7]|c[ 7]|  2400|     1|    1
 n[ 8]|c[ 7]|  2400|     2|    2
 n[ 9]|c[ 9]|  2400|     2|    1
 n[10]|c[ 9]|  2400|     1|    3

8): The smallest Tradeoff(c[2])=0
   Merge Comp(2) and Comp(3) by a link(2,3)


 ** Current Component Information **
<Comp#|ConRt |Nearest|  Link :Cost|CostRt |Tradeoff|Weight|Nodes >
 -----+------+-------+-----------+-------+--------+------+------
 c[ 1]|   c  | c[ 3] |( 1, 3): 432|   432 |      0 |  2400| n[1]
 c[ 2]|   c  | c[ 3] |( 2, 3): 988|   988 |      0 |  2400| n[2]
 c[ 3]|  *c  | c[ 3] |( 3, 3): 200|   200 |      0 | 69600| n[3]
 c[ 7]|  nc  | c[ 3] |( 7, 3):1088|  1088 |      0 |  9600| n[7] n[8] n[0] n[6]
 c[ 9]|  nc  | c[ 3] |( 9, 3):1538|  1538 |      0 |  9600| n[9] n[5] n[10] n[4]

Neighbors of Component[1]: <Component#:(link:cost)>
{<c[ 3]:( 1, 3): 432> <c[ 2]:( 1, 2):1210>}

Neighbors of Component[2]: <Component#:(link:cost)>
{<c[ 3]:( 2, 3): 988> <c[ 1]:( 2, 1):1210>}

Neighbors of Component[7]: <Component#:(link:cost)>
{<c[ 3]:( 7, 3):1088>}

Neighbors of Component[9]: <Component#:(link:cost)>
{<c[ 3]:( 9, 3):1538>}

A = {(8,0) (9,5) (5,10) (9,4) (6,0) (7,8) (1,3) (2,3) }

 ** Current Node Information ('*' indicates a root node) **
  Node#|Comp#|Weight|Degree|Depth
 ------+-----+------+------+-----
  n[ 0]|c[ 7]|  2400|     2|    3
  n[ 1]|c[ 1]|  2400|     1|    1
  n[ 2]|c[ 2]|  2400|     1|    1
 *n[ 3]|c[ 3]| 69600|     2|    0
  n[ 4]|c[ 9]|  2400|     1|    2
  n[ 5]|c[ 9]|  2400|     2|    2
  n[ 6]|c[ 7]|  2400|     1|    4
  n[ 7]|c[ 7]|  2400|     1|    1
  n[ 8]|c[ 7]|  2400|     2|    2
  n[ 9]|c[ 9]|  2400|     2|    1
  n[10]|c[ 9]|  2400|     1|    3

9): The smallest Tradeoff(c[7])=0
   Merge Comp(7) and Comp(3) by a link(7,3)


 ** Current Component Information **
<Comp#|ConRt |Nearest|  Link :Cost|CostRt |Tradeoff|Weight|Nodes >
 -----+------+-------+-----------+-------+--------+------+------
 c[ 1]|   c  | c[ 3] |( 1, 3): 432|   432 |      0 |  2400| n[1]
 c[ 2]|   c  | c[ 3] |( 2, 3): 988|   988 |      0 |  2400| n[2]
```

```
 c[ 3]|  *c  | c[ 3] |( 3, 3): 200|   200 |      0 | 69600| n[3]
 c[ 7]|   c  | c[ 3] |( 7, 3):1088|  1088 |      0 |  9600| n[7] n[8] n[0] n[6]
 c[ 9]|  nc  | c[ 3] |( 9, 3):1538|  1538 |      0 |  9600| n[9] n[5] n[10] n[4]

Neighbors of Component[1]: <Component#:(link:cost)>
{<c[ 3]:( 1, 3): 432> <c[ 2]:( 1, 2):1210>}

Neighbors of Component[2]: <Component#:(link:cost)>
{<c[ 3]:( 2, 3): 988> <c[ 1]:( 2, 1):1210>}

Neighbors of Component[7]: <Component#:(link:cost)>
{<c[ 3]:( 7, 3):1088>}

Neighbors of Component[9]: <Component#:(link:cost)>
{<c[ 3]:( 9, 3):1538>}

A = {(8,0) (9,5) (5,10) (9,4) (6,0) (7,8) (1,3) (2,3) (7,3) }

 ** Current Node Information ('*' indicates a root node) **
  Node#|Comp#|Weight|Degree|Depth
 ------+-----+------+------+-----
  n[ 0]|c[ 7]|  2400|     2|    3
  n[ 1]|c[ 1]|  2400|     1|    1
  n[ 2]|c[ 2]|  2400|     1|    1
 *n[ 3]|c[ 3]| 69600|     3|    0
  n[ 4]|c[ 9]|  2400|     1|    2
  n[ 5]|c[ 9]|  2400|     2|    2
  n[ 6]|c[ 7]|  2400|     1|    4
  n[ 7]|c[ 7]|  2400|     2|    1
  n[ 8]|c[ 7]|  2400|     2|    2
  n[ 9]|c[ 9]|  2400|     2|    1
  n[10]|c[ 9]|  2400|     1|    3

10): The smallest Tradeoff(c[9])=0
   Merge Comp(9) and Comp(3) by a link(9,3)


 ** Current Component Information **
<Comp#|ConRt |Nearest|  Link :Cost|CostRt |Tradeoff|Weight|Nodes >
 -----+------+-------+-----------+-------+--------+------+------
 c[ 1]|   c  | c[ 3] |( 1, 3): 432|   432 |      0 |  2400| n[1]
 c[ 2]|   c  | c[ 3] |( 2, 3): 988|   988 |      0 |  2400| n[2]
 c[ 3]|  *c  | c[ 3] |( 3, 3): 200|   200 |      0 | 69600| n[3]
 c[ 7]|   c  | c[ 3] |( 7, 3):1088|  1088 |      0 |  9600| n[7] n[8] n[0] n[6]
 c[ 9]|   c  | c[ 3] |( 9, 3):1538|  1538 |      0 |  9600| n[9] n[5] n[10] n[4]

Neighbors of Component[1]: <Component#:(link:cost)>
{<c[ 3]:( 1, 3): 432> <c[ 2]:( 1, 2):1210>}

Neighbors of Component[2]: <Component#:(link:cost)>
{<c[ 3]:( 2, 3): 988> <c[ 1]:( 2, 1):1210>}

Neighbors of Component[7]: <Component#:(link:cost)>
{<c[ 3]:( 7, 3):1088>}

Neighbors of Component[9]: <Component#:(link:cost)>
{<c[ 3]:( 9, 3):1538>}
```

```
A = {(8,0) (9,5) (5,10) (9,4) (6,0) (7,8) (1,3) (2,3) (7,3) (9,3) }

 ** Current Node Information ('*' indicates a root node) **
  Node#|Comp#|Weight|Degree|Depth
 ------+-----+------+------+-----
  n[ 0]|c[ 7]|  2400|     2|    3
  n[ 1]|c[ 1]|  2400|     1|    1
  n[ 2]|c[ 2]|  2400|     1|    1
 *n[ 3]|c[ 3]| 69600|     4|    0
  n[ 4]|c[ 9]|  2400|     1|    2
  n[ 5]|c[ 9]|  2400|     2|    2
  n[ 6]|c[ 7]|  2400|     1|    4
  n[ 7]|c[ 7]|  2400|     2|    1
  n[ 8]|c[ 7]|  2400|     2|    2
  n[ 9]|c[ 9]|  2400|     3|    1
  n[10]|c[ 9]|  2400|     1|    3


 *** Complete EW Algorithm ***

 The accepted links and the cost are following

    Link    | Cost
   ---------+------
   (8,0)    | 806
   (9,5)    | 546
   (5,10)   | 1196
   (9,4)    | 1602
   (6,0)    | 1248
   (7,8)    | 1302
   (1,3)    | 432
   (2,3)    | 988
   (7,3)    | 1088
   (9,3)    | 1538
 ----------------------------
 Total Network Cost = 10746
```

D.2 The Order-Constraint Problem

```
The Number of Nodes : 11
Root Node : 3
Weight_limit : 9600
Order_limit : 3
Degree_limit : unconstrained
Depth_limit : unconstrained
 *** <Cost Table> ***
```

Node	0	1	2	3	4	5	6	7	8	9	10
0	200	3434	3496	3366	2730	3148	1248	1604	806	2864	3404
1	3434	200	1210	432	2850	3510	2416	2034	3218	2882	3722
2	3496	1210	200	988	2112	2700	2634	2252	3110	2104	2876
3	3366	432	988	200	2636	2080	1304	1088	1456	1538	2350
4	2730	2850	2112	2636	200	2166	2718	2372	2976	1602	2322
5	3148	3510	2700	2080	2166	200	2362	2228	1946	546	1196
6	1248	2416	2634	1304	2718	2362	200	2426	3266	2164	2904
7	1604	2034	2252	1088	2372	2228	2426	200	1302	1294	2030
8	806	3218	3110	1456	2976	1946	3266	1302	200	3296	4102
9	2864	2882	2104	1538	1602	546	2164	1294	3296	200	3784
10	3404	3722	2876	2350	2322	1196	2904	2030	4102	3784	200

```
 ** Current Node Information ('*' indicates a root node) **
```

Node#	Comp#	Weight	Degree	Depth
n[0]	c[0]	2400	0	1
n[1]	c[1]	2400	0	1
n[2]	c[2]	2400	0	1
*n[3]	c[3]	69600	0	0
n[4]	c[4]	2400	0	1
n[5]	c[5]	2400	0	1
n[6]	c[6]	2400	0	1
n[7]	c[7]	2400	0	1
n[8]	c[8]	2400	0	1
n[9]	c[9]	2400	0	1
n[10]	c[10]	2400	0	1

```
 ** Current Component Information **
```

<Comp#	ConRt	Nearest	Link :Cost	CostRt	Tradeoff	Weight	Nodes >
c[0]	nc	c[8]	(0, 8): 806	3366	-2560	2400	n[0]
c[1]	nc	c[3]	(1, 3): 432	432	0	2400	n[1]
c[2]	nc	c[3]	(2, 3): 988	988	0	2400	n[2]
c[3]	*c	c[3]	(3, 3): 200	200	0	69600	n[3]
c[4]	nc	c[9]	(4, 9):1602	2636	-1034	2400	n[4]
c[5]	nc	c[9]	(5, 9): 546	2080	-1534	2400	n[5]
c[6]	nc	c[0]	(6, 0):1248	1304	-56	2400	n[6]
c[7]	nc	c[3]	(7, 3):1088	1088	0	2400	n[7]
c[8]	nc	c[0]	(8, 0): 806	1456	-650	2400	n[8]
c[9]	nc	c[5]	(9, 5): 546	1538	-992	2400	n[9]
c[10]	nc	c[5]	(10, 5):1196	2350	-1154	2400	n[10]

```
Neighbors of Component[0]: <Component#:(link:cost)>
{<c[ 8]:( 0, 8): 806> <c[ 3]:( 0, 3):3366> <c[ 1]:( 0, 1):3434> <c[ 2]:( 0, 2):3496>
 <c[ 4]:( 0, 4):2730> <c[ 5]:( 0, 5):3148> <c[ 6]:( 0, 6):1248> <c[ 7]:( 0, 7):1604>
 <c[ 9]:( 0, 9):2864> <c[10]:( 0,10):3404>}
```

```
Neighbors of Component[1]: <Component#:(link:cost)>
{<c[ 3]:( 1, 3): 432> <c[ 0]:( 1, 0):3434> <c[ 2]:( 1, 2):1210> <c[ 4]:( 1, 4):2850>
 <c[ 5]:( 1, 5):3510> <c[ 6]:( 1, 6):2416> <c[ 7]:( 1, 7):2034> <c[ 8]:( 1, 8):3218>
 <c[ 9]:( 1, 9):2882> <c[10]:( 1,10):3722>}

Neighbors of Component[2]: <Component#:(link:cost)>
{<c[ 3]:( 2, 3): 988> <c[ 0]:( 2, 0):3496> <c[ 1]:( 2, 1):1210> <c[ 4]:( 2, 4):2112>
 <c[ 5]:( 2, 5):2700> <c[ 6]:( 2, 6):2634> <c[ 7]:( 2, 7):2252> <c[ 8]:( 2, 8):3110>
 <c[ 9]:( 2, 9):2104> <c[10]:( 2,10):2876>}

Neighbors of Component[4]: <Component#:(link:cost)>
{<c[ 9]:( 4, 9):1602> <c[ 3]:( 4, 3):2636> <c[ 0]:( 4, 0):2730> <c[ 1]:( 4, 1):2850>
 <c[ 2]:( 4, 2):2112> <c[ 5]:( 4, 5):2166> <c[ 6]:( 4, 6):2718> <c[ 7]:( 4, 7):2372>
 <c[ 8]:( 4, 8):2976> <c[10]:( 4,10):2322>}

Neighbors of Component[5]: <Component#:(link:cost)>
{<c[ 9]:( 5, 9): 546> <c[ 3]:( 5, 3):2080> <c[ 0]:( 5, 0):3148> <c[ 1]:( 5, 1):3510>
 <c[ 2]:( 5, 2):2700> <c[ 4]:( 5, 4):2166> <c[ 6]:( 5, 6):2362> <c[ 7]:( 5, 7):2228>
 <c[ 8]:( 5, 8):1946> <c[10]:( 5,10):1196>}

Neighbors of Component[6]: <Component#:(link:cost)>
{<c[ 0]:( 6, 0):1248> <c[ 3]:( 6, 3):1304> <c[ 1]:( 6, 1):2416> <c[ 2]:( 6, 2):2634>
 <c[ 4]:( 6, 4):2718> <c[ 5]:( 6, 5):2362> <c[ 7]:( 6, 7):2426> <c[ 8]:( 6, 8):3266>
 <c[ 9]:( 6, 9):2164> <c[10]:( 6,10):2904>}

Neighbors of Component[7]: <Component#:(link:cost)>
{<c[ 3]:( 7, 3):1088> <c[ 0]:( 7, 0):1604> <c[ 1]:( 7, 1):2034> <c[ 2]:( 7, 2):2252>
 <c[ 4]:( 7, 4):2372> <c[ 5]:( 7, 5):2228> <c[ 6]:( 7, 6):2426> <c[ 8]:( 7, 8):1302>
 <c[ 9]:( 7, 9):1294> <c[10]:( 7,10):2030>}

Neighbors of Component[8]: <Component#:(link:cost)>
{<c[ 0]:( 8, 0): 806> <c[ 3]:( 8, 3):1456> <c[ 1]:( 8, 1):3218> <c[ 2]:( 8, 2):3110>
 <c[ 4]:( 8, 4):2976> <c[ 5]:( 8, 5):1946> <c[ 6]:( 8, 6):3266> <c[ 7]:( 8, 7):1302>
 <c[ 9]:( 8, 9):3296> <c[10]:( 8,10):4102>}

Neighbors of Component[9]: <Component#:(link:cost)>
{<c[ 5]:( 9, 5): 546> <c[ 3]:( 9, 3):1538> <c[ 0]:( 9, 0):2864> <c[ 1]:( 9, 1):2882>
 <c[ 2]:( 9, 2):2104> <c[ 4]:( 9, 4):1602> <c[ 6]:( 9, 6):2164> <c[ 7]:( 9, 7):1294>
 <c[ 8]:( 9, 8):3296> <c[10]:( 9,10):3784>}

Neighbors of Component[10]: <Component#:(link:cost)>
{<c[ 5]:(10, 5):1196> <c[ 3]:(10, 3):2350> <c[ 0]:(10, 0):3404> <c[ 1]:(10, 1):3722>
 <c[ 2]:(10, 2):2876> <c[ 4]:(10, 4):2322> <c[ 6]:(10, 6):2904> <c[ 7]:(10, 7):2030>
 <c[ 8]:(10, 8):4102> <c[ 9]:(10, 9):3784>}

A = {}

 *** Start EW Algorithm ***

1): The smallest Tradeoff(c[0])=-2560
   Merge Comp(0) and Comp(8) by a link(0,8)

 Link(8,1) is the representative link between c[1] and c[8]
 Update Tradeoff(c[1]) = 432 - 432 = 0
 Link(8,2) is the representative link between c[2] and c[8]
 Update Tradeoff(c[2]) = 988 - 988 = 0
 Link(0,4) is the representative link between c[4] and c[8]
 Update Tradeoff(c[4]) = 1602 - 2636 = -1034
```

```
 Link(8,5) is the representative link between c[5] and c[8]
 Update Tradeoff(c[5]) = 546 - 2080 = -1534
 Link(0,6) is the representative link between c[6] and c[8]
 Update Tradeoff(c[6]) = 1248 - 1304 = -56
 Link(8,7) is the representative link between c[7] and c[8]
 Update Tradeoff(c[7]) = 1088 - 1088 = 0
 Link(0,9) is the representative link between c[9] and c[8]
 Update Tradeoff(c[9]) = 546 - 1538 = -992
 Link(0,10) is the representative link between c[10] and c[8]
 Update Tradeoff(c[10]) = 1196 - 2350 = -1154
 Update Tradeoff(c[8]) = 1248 - 1456 = -208

 ** Current Component Information **
<Comp#|ConRt |Nearest|  Link :Cost|CostRt |Tradeoff|Weight|Nodes >
 -----+------+-------+------------+-------+--------+------+------
 c[ 1]|  nc  | c[ 3] |( 1, 3): 432|   432 |      0 |  2400| n[1]
 c[ 2]|  nc  | c[ 3] |( 2, 3): 988|   988 |      0 |  2400| n[2]
 c[ 3]|  *c  | c[ 3] |( 3, 3): 200|   200 |      0 | 69600| n[3]
 c[ 4]|  nc  | c[ 9] |( 4, 9):1602|  2636 |  -1034 |  2400| n[4]
 c[ 5]|  nc  | c[ 9] |( 5, 9): 546|  2080 |  -1534 |  2400| n[5]
 c[ 6]|  nc  | c[ 8] |( 0, 6):1248|  1304 |    -56 |  2400| n[6]
 c[ 7]|  nc  | c[ 3] |( 7, 3):1088|  1088 |      0 |  2400| n[7]
 c[ 8]|  nc  | c[ 6] |( 0, 6):1248|  1456 |   -208 |  4800| n[8] n[0]
 c[ 9]|  nc  | c[ 5] |( 9, 5): 546|  1538 |   -992 |  2400| n[9]
 c[10]|  nc  | c[ 5] |(10, 5):1196|  2350 |  -1154 |  2400| n[10]

Neighbors of Component[1]: <Component#:(link:cost)>
{<c[ 3]:( 1, 3): 432> <c[ 2]:( 1, 2):1210> <c[ 4]:( 1, 4):2850> <c[ 5]:( 1, 5):3510>
 <c[ 6]:( 1, 6):2416> <c[ 7]:( 1, 7):2034> <c[ 8]:( 1, 8):3218> <c[ 9]:( 1, 9):2882>
 <c[10]:( 1,10):3722>}

Neighbors of Component[2]: <Component#:(link:cost)>
{<c[ 3]:( 2, 3): 988> <c[ 1]:( 2, 1):1210> <c[ 4]:( 2, 4):2112> <c[ 5]:( 2, 5):2700>
 <c[ 6]:( 2, 6):2634> <c[ 7]:( 2, 7):2252> <c[ 8]:( 2, 8):3110> <c[ 9]:( 2, 9):2104>
 <c[10]:( 2,10):2876>}

Neighbors of Component[4]: <Component#:(link:cost)>
{<c[ 9]:( 4, 9):1602> <c[ 3]:( 4, 3):2636> <c[ 1]:( 4, 1):2850> <c[ 2]:( 4, 2):2112>
 <c[ 5]:( 4, 5):2166> <c[ 6]:( 4, 6):2718> <c[ 7]:( 4, 7):2372> <c[ 8]:( 0, 4):2730>
 <c[10]:( 4,10):2322>}

Neighbors of Component[5]: <Component#:(link:cost)>
{<c[ 9]:( 5, 9): 546> <c[ 3]:( 5, 3):2080> <c[ 1]:( 5, 1):3510> <c[ 2]:( 5, 2):2700>
 <c[ 4]:( 5, 4):2166> <c[ 6]:( 5, 6):2362> <c[ 7]:( 5, 7):2228> <c[ 8]:( 5, 8):1946>
 <c[10]:( 5,10):1196>}

Neighbors of Component[6]: <Component#:(link:cost)>
{<c[ 8]:( 0, 6):1248> <c[ 3]:( 6, 3):1304> <c[ 1]:( 6, 1):2416> <c[ 2]:( 6, 2):2634>
 <c[ 4]:( 6, 4):2718> <c[ 5]:( 6, 5):2362> <c[ 7]:( 6, 7):2426> <c[ 9]:( 6, 9):2164>
 <c[10]:( 6,10):2904>}

Neighbors of Component[7]: <Component#:(link:cost)>
{<c[ 3]:( 7, 3):1088> <c[ 1]:( 7, 1):2034> <c[ 2]:( 7, 2):2252> <c[ 4]:( 7, 4):2372>
 <c[ 5]:( 7, 5):2228> <c[ 6]:( 7, 6):2426> <c[ 8]:( 7, 8):1302> <c[ 9]:( 7, 9):1294>
 <c[10]:( 7,10):2030>}

Neighbors of Component[8]: <Component#:(link:cost)>
{<c[ 6]:( 0, 6):1248> <c[ 3]:( 8, 3):1456> <c[ 1]:( 1, 8):3218> <c[ 2]:( 2, 8):3110>
```

```
 <c[ 4]:( 0, 4):2730> <c[ 5]:( 5, 8):1946> <c[ 7]:( 7, 8):1302> <c[ 9]:( 0, 9):2864>
 <c[10]:( 0,10):3404>}

Neighbors of Component[9]: <Component#:(link:cost)>
{<c[ 5]:( 9, 5): 546> <c[ 3]:( 9, 3):1538> <c[ 1]:( 9, 1):2882> <c[ 2]:( 9, 2):2104>
 <c[ 4]:( 9, 4):1602> <c[ 6]:( 9, 6):2164> <c[ 7]:( 9, 7):1294> <c[ 8]:( 0, 9):2864>
 <c[10]:( 9,10):3784>}

Neighbors of Component[10]: <Component#:(link:cost)>
{<c[ 5]:(10, 5):1196> <c[ 3]:(10, 3):2350> <c[ 1]:(10, 1):3722> <c[ 2]:(10, 2):2876>
 <c[ 4]:(10, 4):2322> <c[ 6]:(10, 6):2904> <c[ 7]:(10, 7):2030> <c[ 8]:( 0,10):3404>
 <c[ 9]:(10, 9):3784>}

A = {(8,0) }

 ** Current Node Information ('*' indicates a root node) **
  Node#|Comp#|Weight|Degree|Depth
 ------+-----+------+------+-----
  n[ 0]|c[ 8]|  2400|     1|    2
  n[ 1]|c[ 1]|  2400|     0|    1
  n[ 2]|c[ 2]|  2400|     0|    1
 *n[ 3]|c[ 3]| 69600|     0|    0
  n[ 4]|c[ 4]|  2400|     0|    1
  n[ 5]|c[ 5]|  2400|     0|    1
  n[ 6]|c[ 6]|  2400|     0|    1
  n[ 7]|c[ 7]|  2400|     0|    1
  n[ 8]|c[ 8]|  2400|     1|    1
  n[ 9]|c[ 9]|  2400|     0|    1
  n[10]|c[10]|  2400|     0|    1

2): The smallest Tradeoff(c[5])=-1534
   Merge Comp(5) and Comp(9) by a link(5,9)

 Link(9,1) is the representative link between c[1] and c[9]
 Update Tradeoff(c[1]) = 432 - 432 = 0
 Link(9,2) is the representative link between c[2] and c[9]
 Update Tradeoff(c[2]) = 988 - 988 = 0
 Link(9,4) is the representative link between c[4] and c[9]
 Update Tradeoff(c[4]) = 1602 - 2636 = -1034
 Link(9,6) is the representative link between c[6] and c[9]
 Update Tradeoff(c[6]) = 1248 - 1304 = -56
 Link(9,7) is the representative link between c[7] and c[9]
 Update Tradeoff(c[7]) = 1088 - 1088 = 0
 Merging c[9] and c[8]) would violate the order-constraint.
 Destory a relation between Comp(9) and Comp(8)
 Update Tradeoff(c[8]) = 1248 - 1456 = -208
 Link(5,10) is the representative link between c[10] and c[9]
 Update Tradeoff(c[10]) = 1196 - 2350 = -1154
 Update Tradeoff(c[9]) = 1196 - 1538 = -342

 ** Current Component Information **
<Comp#|ConRt |Nearest|  Link :Cost|CostRt |Tradeoff|Weight|Nodes >
 -----+------+-------+-----------+-------+--------+------+------
 c[ 1]|  nc  | c[ 3] |( 1, 3): 432|   432 |      0 |  2400| n[1]
 c[ 2]|  nc  | c[ 3] |( 2, 3): 988|   988 |      0 |  2400| n[2]
 c[ 3]|  *c  | c[ 3] |( 3, 3): 200|   200 |      0 | 69600| n[3]
 c[ 4]|  nc  | c[ 9] |( 4, 9):1602|  2636 |  -1034 |  2400| n[4]
 c[ 6]|  nc  | c[ 8] |( 0, 6):1248|  1304 |    -56 |  2400| n[6]
```

```
 c[ 7]|  nc  | c[ 3] |( 7, 3):1088|  1088 |    0 | 2400| n[7]
 c[ 8]|  nc  | c[ 6] |( 0, 6):1248|  1456 |  -208 | 4800| n[8] n[0]
 c[ 9]|  nc  | c[10] |( 5,10):1196|  1538 |  -342 | 4800| n[9] n[5]
 c[10]|  nc  | c[ 9] |( 5,10):1196|  2350 | -1154 | 2400| n[10]

Neighbors of Component[1]: <Component#:(link:cost)>
{<c[ 3]:( 1, 3): 432> <c[ 2]:( 1, 2):1210> <c[ 4]:( 1, 4):2850> <c[ 6]:( 1, 6):2416>
 <c[ 7]:( 1, 7):2034> <c[ 8]:( 1, 8):3218> <c[ 9]:( 1, 9):2882> <c[10]:( 1,10):3722>}

Neighbors of Component[2]: <Component#:(link:cost)>
{<c[ 3]:( 2, 3): 988> <c[ 1]:( 2, 1):1210> <c[ 4]:( 2, 4):2112> <c[ 6]:( 2, 6):2634>
 <c[ 7]:( 2, 7):2252> <c[ 8]:( 2, 8):3110> <c[ 9]:( 2, 9):2104> <c[10]:( 2,10):2876>}

Neighbors of Component[4]: <Component#:(link:cost)>
{<c[ 9]:( 4, 9):1602> <c[ 3]:( 4, 3):2636> <c[ 1]:( 4, 1):2850> <c[ 2]:( 4, 2):2112>
 <c[ 6]:( 4, 6):2718> <c[ 7]:( 4, 7):2372> <c[ 8]:( 0, 4):2730> <c[10]:( 4,10):2322>}

Neighbors of Component[6]: <Component#:(link:cost)>
{<c[ 8]:( 0, 6):1248> <c[ 3]:( 6, 3):1304> <c[ 1]:( 6, 1):2416> <c[ 2]:( 6, 2):2634>
 <c[ 4]:( 6, 4):2718> <c[ 7]:( 6, 7):2426> <c[ 9]:( 6, 9):2164> <c[10]:( 6,10):2904>}

Neighbors of Component[7]: <Component#:(link:cost)>
{<c[ 3]:( 7, 3):1088> <c[ 1]:( 7, 1):2034> <c[ 2]:( 7, 2):2252> <c[ 4]:( 7, 4):2372>
 <c[ 6]:( 7, 6):2426> <c[ 8]:( 7, 8):1302> <c[ 9]:( 7, 9):1294> <c[10]:( 7,10):2030>}

Neighbors of Component[8]: <Component#:(link:cost)>
{<c[ 6]:( 0, 6):1248> <c[ 3]:( 8, 3):1456> <c[ 1]:( 1, 8):3218> <c[ 2]:( 2, 8):3110>
 <c[ 4]:( 0, 4):2730> <c[ 7]:( 7, 8):1302> <c[10]:( 0,10):3404>}

Neighbors of Component[9]: <Component#:(link:cost)>
{<c[10]:( 5,10):1196> <c[ 3]:( 9, 3):1538> <c[ 1]:( 1, 9):2882> <c[ 2]:( 2, 9):2104>
 <c[ 4]:( 4, 9):1602> <c[ 6]:( 6, 9):2164> <c[ 7]:( 7, 9):1294>}

Neighbors of Component[10]: <Component#:(link:cost)>
{<c[ 9]:( 5,10):1196> <c[ 3]:(10, 3):2350> <c[ 1]:(10, 1):3722> <c[ 2]:(10, 2):2876>
 <c[ 4]:(10, 4):2322> <c[ 6]:(10, 6):2904> <c[ 7]:(10, 7):2030> <c[ 8]:( 0,10):3404>}

A = {(8,0) (9,5) }

 ** Current Node Information ('*' indicates a root node) **
  Node#|Comp#|Weight|Degree|Depth
 ------+-----+------+------+-----
  n[ 0]|c[ 8]|  2400|     1|    2
  n[ 1]|c[ 1]|  2400|     0|    1
  n[ 2]|c[ 2]|  2400|     0|    1
 *n[ 3]|c[ 3]| 69600|     0|    0
  n[ 4]|c[ 4]|  2400|     0|    1
  n[ 5]|c[ 9]|  2400|     1|    2
  n[ 6]|c[ 6]|  2400|     0|    1
  n[ 7]|c[ 7]|  2400|     0|    1
  n[ 8]|c[ 8]|  2400|     1|    1
  n[ 9]|c[ 9]|  2400|     1|    1
  n[10]|c[10]|  2400|     0|    1

3): The smallest Tradeoff(c[10])=-1154
   Merge Comp(10) and Comp(9) by a link(5,10)

 Update Tradeoff(c[8]) = 1248 - 1456 = -208
```

```
Merging c[9] and c[1]) would violate the order-constraint.
Destory a relation between Comp(9) and Comp(1)
Update Tradeoff(c[1]) = 432 - 432 = 0
Merging c[9] and c[2]) would violate the order-constraint.
Destory a relation between Comp(9) and Comp(2)
Update Tradeoff(c[2]) = 988 - 988 = 0
Merging c[9] and c[4]) would violate the order-constraint.
Destory a relation between Comp(9) and Comp(4)
Update Tradeoff(c[4]) = 2112 - 2636 = -524
Merging c[9] and c[6]) would violate the order-constraint.
Destory a relation between Comp(9) and Comp(6)
Update Tradeoff(c[6]) = 1248 - 1304 = -56
Merging c[9] and c[7]) would violate the order-constraint.
Destory a relation between Comp(9) and Comp(7)
Update Tradeoff(c[7]) = 1088 - 1088 = 0
Update Tradeoff(c[9]) = 1538 - 1538 = 0

** Current Component Information **
<Comp#|ConRt |Nearest|  Link :Cost|CostRt |Tradeoff|Weight|Nodes >
-----+------+-------+------------+-------+--------+------+------
c[ 1]|  nc  | c[ 3] |( 1, 3): 432|   432 |      0 |  2400| n[1]
c[ 2]|  nc  | c[ 3] |( 2, 3): 988|   988 |      0 |  2400| n[2]
c[ 3]|  *c  | c[ 3] |( 3, 3): 200|   200 |      0 | 69600| n[3]
c[ 4]|  nc  | c[ 2] |( 4, 2):2112|  2636 |   -524 |  2400| n[4]
c[ 6]|  nc  | c[ 8] |( 0, 6):1248|  1304 |    -56 |  2400| n[6]
c[ 7]|  nc  | c[ 3] |( 7, 3):1088|  1088 |      0 |  2400| n[7]
c[ 8]|  nc  | c[ 6] |( 0, 6):1248|  1456 |   -208 |  4800| n[8] n[0]
c[ 9]|  nc  | c[ 3] |( 9, 3):1538|  1538 |      0 |  7200| n[9] n[5] n[10]

Neighbors of Component[1]: <Component#:(link:cost)>
{<c[ 3]:( 1, 3): 432> <c[ 2]:( 1, 2):1210> <c[ 4]:( 1, 4):2850> <c[ 6]:( 1, 6):2416>
 <c[ 7]:( 1, 7):2034> <c[ 8]:( 1, 8):3218>}

Neighbors of Component[2]: <Component#:(link:cost)>
{<c[ 3]:( 2, 3): 988> <c[ 1]:( 2, 1):1210> <c[ 4]:( 2, 4):2112> <c[ 6]:( 2, 6):2634>
 <c[ 7]:( 2, 7):2252> <c[ 8]:( 2, 8):3110>}

Neighbors of Component[4]: <Component#:(link:cost)>
{<c[ 2]:( 4, 2):2112> <c[ 3]:( 4, 3):2636> <c[ 1]:( 4, 1):2850> <c[ 6]:( 4, 6):2718>
 <c[ 7]:( 4, 7):2372> <c[ 8]:( 0, 4):2730>}

Neighbors of Component[6]: <Component#:(link:cost)>
{<c[ 8]:( 0, 6):1248> <c[ 3]:( 6, 3):1304> <c[ 1]:( 6, 1):2416> <c[ 2]:( 6, 2):2634>
 <c[ 4]:( 6, 4):2718> <c[ 7]:( 6, 7):2426>}

Neighbors of Component[7]: <Component#:(link:cost)>
{<c[ 3]:( 7, 3):1088> <c[ 1]:( 7, 1):2034> <c[ 2]:( 7, 2):2252> <c[ 4]:( 7, 4):2372>
 <c[ 6]:( 7, 6):2426> <c[ 8]:( 7, 8):1302>}

Neighbors of Component[8]: <Component#:(link:cost)>
{<c[ 6]:( 0, 6):1248> <c[ 3]:( 8, 3):1456> <c[ 1]:( 1, 8):3218> <c[ 2]:( 2, 8):3110>
 <c[ 4]:( 0, 4):2730> <c[ 7]:( 7, 8):1302>}

Neighbors of Component[9]: <Component#:(link:cost)>
{<c[ 3]:( 9, 3):1538>}

A = {(8,0) (9,5) (5,10) }
```

```
** Current Node Information ('*' indicates a root node) **
 Node#|Comp#|Weight|Degree|Depth
------+-----+------+------+-----
 n[ 0]|c[ 8]|  2400|     1|    2
 n[ 1]|c[ 1]|  2400|     0|    1
 n[ 2]|c[ 2]|  2400|     0|    1
*n[ 3]|c[ 3]| 69600|     0|    0
 n[ 4]|c[ 4]|  2400|     0|    1
 n[ 5]|c[ 9]|  2400|     2|    2
 n[ 6]|c[ 6]|  2400|     0|    1
 n[ 7]|c[ 7]|  2400|     0|    1
 n[ 8]|c[ 8]|  2400|     1|    1
 n[ 9]|c[ 9]|  2400|     1|    1
 n[10]|c[ 9]|  2400|     1|    3

4): The smallest Tradeoff(c[4])=-524
   Merge Comp(4) and Comp(2) by a link(4,2)

 Link(2,1) is the representative link between c[1] and c[2]
 Update Tradeoff(c[1]) = 432 - 432 = 0
 Link(2,6) is the representative link between c[6] and c[2]
 Update Tradeoff(c[6]) = 1248 - 1304 = -56
 Link(2,7) is the representative link between c[7] and c[2]
 Update Tradeoff(c[7]) = 1088 - 1088 = 0
 Merging c[2] and c[8]) would violate the order-constraint.
 Destory a relation between Comp(2) and Comp(8)
 Update Tradeoff(c[8]) = 1248 - 1456 = -208
 Update Tradeoff(c[2]) = 988 - 988 = 0

 ** Current Component Information **
<Comp#|ConRt |Nearest|  Link :Cost|CostRt |Tradeoff|Weight|Nodes >
 -----+------+-------+-----------+-------+--------+------+------
 c[ 1]|  nc  | c[ 3] |( 1, 3): 432|   432 |      0 |  2400| n[1]
 c[ 2]|  nc  | c[ 3] |( 2, 3): 988|   988 |      0 |  4800| n[2] n[4]
 c[ 3]|  *c  | c[ 3] |( 3, 3): 200|   200 |      0 | 69600| n[3]
 c[ 6]|  nc  | c[ 8] |( 0, 6):1248|  1304 |    -56 |  2400| n[6]
 c[ 7]|  nc  | c[ 3] |( 7, 3):1088|  1088 |      0 |  2400| n[7]
 c[ 8]|  nc  | c[ 6] |( 0, 6):1248|  1456 |   -208 |  4800| n[8] n[0]
 c[ 9]|  nc  | c[ 3] |( 9, 3):1538|  1538 |      0 |  7200| n[9] n[5] n[10]

Neighbors of Component[1]: <Component#:(link:cost)>
{<c[ 3]:( 1, 3): 432> <c[ 2]:( 1, 2):1210> <c[ 6]:( 1, 6):2416> <c[ 7]:( 1, 7):2034>
 <c[ 8]:( 1, 8):3218>}

Neighbors of Component[2]: <Component#:(link:cost)>
{<c[ 3]:( 2, 3): 988> <c[ 1]:( 1, 2):1210> <c[ 6]:( 2, 6):2634> <c[ 7]:( 2, 7):2252>}

Neighbors of Component[6]: <Component#:(link:cost)>
{<c[ 8]:( 0, 6):1248> <c[ 3]:( 6, 3):1304> <c[ 1]:( 6, 1):2416> <c[ 2]:( 2, 6):2634>
 <c[ 7]:( 6, 7):2426>}

Neighbors of Component[7]: <Component#:(link:cost)>
{<c[ 3]:( 7, 3):1088> <c[ 1]:( 7, 1):2034> <c[ 2]:( 2, 7):2252> <c[ 6]:( 7, 6):2426>
 <c[ 8]:( 7, 8):1302>}

Neighbors of Component[8]: <Component#:(link:cost)>
{<c[ 6]:( 0, 6):1248> <c[ 3]:( 8, 3):1456> <c[ 1]:( 1, 8):3218> <c[ 7]:( 7, 8):1302>}
```

```
Neighbors of Component[9]: <Component#:(link:cost)>
{<c[ 3]:( 9, 3):1538>}

A = {(8,0) (9,5) (5,10) (2,4) }

 ** Current Node Information ('*' indicates a root node) **
  Node#|Comp#|Weight|Degree|Depth
 ------+-----+------+------+-----
  n[ 0]|c[ 8]|  2400|     1|    2
  n[ 1]|c[ 1]|  2400|     0|    1
  n[ 2]|c[ 2]|  2400|     1|    1
 *n[ 3]|c[ 3]| 69600|     0|    0
  n[ 4]|c[ 2]|  2400|     1|    2
  n[ 5]|c[ 9]|  2400|     2|    2
  n[ 6]|c[ 6]|  2400|     0|    1
  n[ 7]|c[ 7]|  2400|     0|    1
  n[ 8]|c[ 8]|  2400|     1|    1
  n[ 9]|c[ 9]|  2400|     1|    1
  n[10]|c[ 9]|  2400|     1|    3

5): The smallest Tradeoff(c[8])=-208
   Merge Comp(8) and Comp(6) by a link(0,6)

 Merging c[6] and c[1]) would violate the order-constraint.
 Destory a relation between Comp(6) and Comp(1)
 Update Tradeoff(c[1]) = 432 - 432 = 0
 Destory a relation between Comp(6) and Comp(2)
 Update Tradeoff(c[2]) = 988 - 988 = 0
 Merging c[6] and c[7]) would violate the order-constraint.
 Destory a relation between Comp(6) and Comp(7)
 Update Tradeoff(c[7]) = 1088 - 1088 = 0
 Update Tradeoff(c[6]) = 1304 - 1304 = 0

 ** Current Component Information **
<Comp#|ConRt |Nearest|  Link :Cost|CostRt |Tradeoff|Weight|Nodes >
 -----+------+-------+------------+-------+--------+------+------
 c[ 1]|  nc  | c[ 3] |( 1, 3): 432|   432 |      0 |  2400| n[1]
 c[ 2]|  nc  | c[ 3] |( 2, 3): 988|   988 |      0 |  4800| n[2] n[4]
 c[ 3]|  *c  | c[ 3] |( 3, 3): 200|   200 |      0 | 69600| n[3]
 c[ 6]|  nc  | c[ 3] |( 6, 3):1304|  1304 |      0 |  7200| n[6] n[0] n[8]
 c[ 7]|  nc  | c[ 3] |( 7, 3):1088|  1088 |      0 |  2400| n[7]
 c[ 9]|  nc  | c[ 3] |( 9, 3):1538|  1538 |      0 |  7200| n[9] n[5] n[10]

Neighbors of Component[1]: <Component#:(link:cost)>
{<c[ 3]:( 1, 3): 432> <c[ 2]:( 1, 2):1210> <c[ 7]:( 1, 7):2034>}

Neighbors of Component[2]: <Component#:(link:cost)>
{<c[ 3]:( 2, 3): 988> <c[ 1]:( 1, 2):1210> <c[ 7]:( 2, 7):2252>}

Neighbors of Component[6]: <Component#:(link:cost)>
{<c[ 3]:( 6, 3):1304>}

Neighbors of Component[7]: <Component#:(link:cost)>
{<c[ 3]:( 7, 3):1088> <c[ 1]:( 7, 1):2034> <c[ 2]:( 2, 7):2252>}

Neighbors of Component[9]: <Component#:(link:cost)>
{<c[ 3]:( 9, 3):1538>}
```

```
A = {(8,0) (9,5) (5,10) (2,4) (6,0) }

 ** Current Node Information ('*' indicates a root node) **
  Node#|Comp#|Weight|Degree|Depth
 ------+-----+------+------+-----
  n[ 0]|c[ 6]|  2400|     2|    2
  n[ 1]|c[ 1]|  2400|     0|    1
  n[ 2]|c[ 2]|  2400|     1|    1
 *n[ 3]|c[ 3]| 69600|     0|    0
  n[ 4]|c[ 2]|  2400|     1|    2
  n[ 5]|c[ 9]|  2400|     2|    2
  n[ 6]|c[ 6]|  2400|     1|    1
  n[ 7]|c[ 7]|  2400|     0|    1
  n[ 8]|c[ 6]|  2400|     1|    3
  n[ 9]|c[ 9]|  2400|     1|    1
  n[10]|c[ 9]|  2400|     1|    3

6): The smallest Tradeoff(c[1])=0
   Merge Comp(1) and Comp(3) by a link(1,3)


 ** Current Component Information **
<Comp#|ConRt |Nearest|  Link :Cost|CostRt |Tradeoff|Weight|Nodes >
 -----+------+-------+-----------+-------+--------+------+------
 c[ 1]|   c  | c[ 3] |( 1, 3): 432|   432 |      0 |  2400| n[1]
 c[ 2]|  nc  | c[ 3] |( 2, 3): 988|   988 |      0 |  4800| n[2] n[4]
 c[ 3]|  *c  | c[ 3] |( 3, 3): 200|   200 |      0 | 69600| n[3]
 c[ 6]|  nc  | c[ 3] |( 6, 3):1304|  1304 |      0 |  7200| n[6] n[0] n[8]
 c[ 7]|  nc  | c[ 3] |( 7, 3):1088|  1088 |      0 |  2400| n[7]
 c[ 9]|  nc  | c[ 3] |( 9, 3):1538|  1538 |      0 |  7200| n[9] n[5] n[10]

Neighbors of Component[1]: <Component#:(link:cost)>
{<c[ 3]:( 1, 3): 432> <c[ 2]:( 1, 2):1210> <c[ 7]:( 1, 7):2034>}

Neighbors of Component[2]: <Component#:(link:cost)>
{<c[ 3]:( 2, 3): 988> <c[ 1]:( 1, 2):1210> <c[ 7]:( 2, 7):2252>}

Neighbors of Component[6]: <Component#:(link:cost)>
{<c[ 3]:( 6, 3):1304>}

Neighbors of Component[7]: <Component#:(link:cost)>
{<c[ 3]:( 7, 3):1088> <c[ 1]:( 7, 1):2034> <c[ 2]:( 2, 7):2252>}

Neighbors of Component[9]: <Component#:(link:cost)>
{<c[ 3]:( 9, 3):1538>}

A = {(8,0) (9,5) (5,10) (2,4) (6,0) (1,3) }

 ** Current Node Information ('*' indicates a root node) **
  Node#|Comp#|Weight|Degree|Depth
 ------+-----+------+------+-----
  n[ 0]|c[ 6]|  2400|     2|    2
  n[ 1]|c[ 1]|  2400|     1|    1
  n[ 2]|c[ 2]|  2400|     1|    1
 *n[ 3]|c[ 3]| 69600|     1|    0
  n[ 4]|c[ 2]|  2400|     1|    2
  n[ 5]|c[ 9]|  2400|     2|    2
  n[ 6]|c[ 6]|  2400|     1|    1
```

```
  n[ 7]|c[ 7]|  2400|     0|    1
  n[ 8]|c[ 6]|  2400|     1|    3
  n[ 9]|c[ 9]|  2400|     1|    1
  n[10]|c[ 9]|  2400|     1|    3

7): The smallest Tradeoff(c[2])=0
   Merge Comp(2) and Comp(3) by a link(2,3)


 ** Current Component Information **
<Comp#|ConRt |Nearest|  Link :Cost|CostRt |Tradeoff|Weight|Nodes >
 -----+------+-------+-----------+-------+--------+------+-----
 c[ 1]|   c  | c[ 3] |( 1, 3): 432|   432 |      0 |  2400| n[1]
 c[ 2]|   c  | c[ 3] |( 2, 3): 988|   988 |      0 |  4800| n[2] n[4]
 c[ 3]|  *c  | c[ 3] |( 3, 3): 200|   200 |      0 | 69600| n[3]
 c[ 6]|  nc  | c[ 3] |( 6, 3):1304|  1304 |      0 |  7200| n[6] n[0] n[8]
 c[ 7]|  nc  | c[ 3] |( 7, 3):1088|  1088 |      0 |  2400| n[7]
 c[ 9]|  nc  | c[ 3] |( 9, 3):1538|  1538 |      0 |  7200| n[9] n[5] n[10]

Neighbors of Component[1]: <Component#:(link:cost)>
{<c[ 3]:( 1, 3): 432> <c[ 2]:( 1, 2):1210> <c[ 7]:( 1, 7):2034>}

Neighbors of Component[2]: <Component#:(link:cost)>
{<c[ 3]:( 2, 3): 988> <c[ 1]:( 1, 2):1210> <c[ 7]:( 2, 7):2252>}

Neighbors of Component[6]: <Component#:(link:cost)>
{<c[ 3]:( 6, 3):1304>}

Neighbors of Component[7]: <Component#:(link:cost)>
{<c[ 3]:( 7, 3):1088> <c[ 1]:( 7, 1):2034> <c[ 2]:( 2, 7):2252>}

Neighbors of Component[9]: <Component#:(link:cost)>
{<c[ 3]:( 9, 3):1538>}

A = {(8,0) (9,5) (5,10) (2,4) (6,0) (1,3) (2,3) }

 ** Current Node Information ('*' indicates a root node) **
  Node#|Comp#|Weight|Degree|Depth
 ------+-----+------+------+-----
  n[ 0]|c[ 6]|  2400|     2|    2
  n[ 1]|c[ 1]|  2400|     1|    1
  n[ 2]|c[ 2]|  2400|     2|    1
 *n[ 3]|c[ 3]| 69600|     2|    0
  n[ 4]|c[ 2]|  2400|     1|    2
  n[ 5]|c[ 9]|  2400|     2|    2
  n[ 6]|c[ 6]|  2400|     1|    1
  n[ 7]|c[ 7]|  2400|     0|    1
  n[ 8]|c[ 6]|  2400|     1|    3
  n[ 9]|c[ 9]|  2400|     1|    1
  n[10]|c[ 9]|  2400|     1|    3

8): The smallest Tradeoff(c[6])=0
   Merge Comp(6) and Comp(3) by a link(6,3)


 ** Current Component Information **
<Comp#|ConRt |Nearest|  Link :Cost|CostRt |Tradeoff|Weight|Nodes >
 -----+------+-------+-----------+-------+--------+------+------
```

```
 c[ 1]|   c  | c[ 3] |( 1, 3): 432|   432 |      0 |  2400| n[1]
 c[ 2]|   c  | c[ 3] |( 2, 3): 988|   988 |      0 |  4800| n[2] n[4]
 c[ 3]|  *c  | c[ 3] |( 3, 3): 200|   200 |      0 | 69600| n[3]
 c[ 6]|   c  | c[ 3] |( 6, 3):1304|  1304 |      0 |  7200| n[6] n[0] n[8]
 c[ 7]|  nc  | c[ 3] |( 7, 3):1088|  1088 |      0 |  2400| n[7]
 c[ 9]|  nc  | c[ 3] |( 9, 3):1538|  1538 |      0 |  7200| n[9] n[5] n[10]

Neighbors of Component[1]: <Component#:(link:cost)>
{<c[ 3]:( 1, 3): 432> <c[ 2]:( 1, 2):1210> <c[ 7]:( 1, 7):2034>}

Neighbors of Component[2]: <Component#:(link:cost)>
{<c[ 3]:( 2, 3): 988> <c[ 1]:( 1, 2):1210> <c[ 7]:( 2, 7):2252>}

Neighbors of Component[6]: <Component#:(link:cost)>
{<c[ 3]:( 6, 3):1304>}

Neighbors of Component[7]: <Component#:(link:cost)>
{<c[ 3]:( 7, 3):1088> <c[ 1]:( 7, 1):2034> <c[ 2]:( 2, 7):2252>}

Neighbors of Component[9]: <Component#:(link:cost)>
{<c[ 3]:( 9, 3):1538>}

A = {(8,0) (9,5) (5,10) (2,4) (6,0) (1,3) (2,3) (6,3) }

 ** Current Node Information ('*' indicates a root node) **
  Node#|Comp#|Weight|Degree|Depth
 ------+-----+------+------+-----
  n[ 0]|c[ 6]|  2400|     2|    2
  n[ 1]|c[ 1]|  2400|     1|    1
  n[ 2]|c[ 2]|  2400|     2|    1
 *n[ 3]|c[ 3]| 69600|     3|    0
  n[ 4]|c[ 2]|  2400|     1|    2
  n[ 5]|c[ 9]|  2400|     2|    2
  n[ 6]|c[ 6]|  2400|     2|    1
  n[ 7]|c[ 7]|  2400|     0|    1
  n[ 8]|c[ 6]|  2400|     1|    3
  n[ 9]|c[ 9]|  2400|     1|    1
  n[10]|c[ 9]|  2400|     1|    3

9): The smallest Tradeoff(c[7])=0
   Merge Comp(7) and Comp(3) by a link(7,3)


 ** Current Component Information **
<Comp#|ConRt |Nearest|  Link :Cost|CostRt |Tradeoff|Weight|Nodes >
 -----+------+-------+------------+-------+--------+------+------
 c[ 1]|   c  | c[ 3] |( 1, 3): 432|   432 |      0 |  2400| n[1]
 c[ 2]|   c  | c[ 3] |( 2, 3): 988|   988 |      0 |  4800| n[2] n[4]
 c[ 3]|  *c  | c[ 3] |( 3, 3): 200|   200 |      0 | 69600| n[3]
 c[ 6]|   c  | c[ 3] |( 6, 3):1304|  1304 |      0 |  7200| n[6] n[0] n[8]
 c[ 7]|   c  | c[ 3] |( 7, 3):1088|  1088 |      0 |  2400| n[7]
 c[ 9]|  nc  | c[ 3] |( 9, 3):1538|  1538 |      0 |  7200| n[9] n[5] n[10]

Neighbors of Component[1]: <Component#:(link:cost)>
{<c[ 3]:( 1, 3): 432> <c[ 2]:( 1, 2):1210> <c[ 7]:( 1, 7):2034>}

Neighbors of Component[2]: <Component#:(link:cost)>
{<c[ 3]:( 2, 3): 988> <c[ 1]:( 1, 2):1210> <c[ 7]:( 2, 7):2252>}
```

```
Neighbors of Component[6]: <Component#:(link:cost)>
{<c[ 3]:( 6, 3):1304>}

Neighbors of Component[7]: <Component#:(link:cost)>
{<c[ 3]:( 7, 3):1088> <c[ 1]:( 7, 1):2034> <c[ 2]:( 2, 7):2252>}

Neighbors of Component[9]: <Component#:(link:cost)>
{<c[ 3]:( 9, 3):1538>}

A = {(8,0) (9,5) (5,10) (2,4) (6,0) (1,3) (2,3) (6,3) (7,3) }

 ** Current Node Information ('*' indicates a root node) **
  Node#|Comp#|Weight|Degree|Depth
 ------+-----+------+------+-----
  n[ 0]|c[ 6]|  2400|     2|    2
  n[ 1]|c[ 1]|  2400|     1|    1
  n[ 2]|c[ 2]|  2400|     2|    1
 *n[ 3]|c[ 3]| 69600|     4|    0
  n[ 4]|c[ 2]|  2400|     1|    2
  n[ 5]|c[ 9]|  2400|     2|    2
  n[ 6]|c[ 6]|  2400|     2|    1
  n[ 7]|c[ 7]|  2400|     1|    1
  n[ 8]|c[ 6]|  2400|     1|    3
  n[ 9]|c[ 9]|  2400|     1|    1
  n[10]|c[ 9]|  2400|     1|    3

10): The smallest Tradeoff(c[9])=0
   Merge Comp(9) and Comp(3) by a link(9,3)


 ** Current Component Information **
<Comp#|ConRt |Nearest|  Link :Cost|CostRt |Tradeoff|Weight|Nodes >
 -----+------+-------+-----------+-------+--------+------+------
 c[ 1]|   c  | c[ 3] |( 1, 3): 432|   432 |      0 |  2400| n[1]
 c[ 2]|   c  | c[ 3] |( 2, 3): 988|   988 |      0 |  4800| n[2] n[4]
 c[ 3]|  *c  | c[ 3] |( 3, 3): 200|   200 |      0 | 69600| n[3]
 c[ 6]|   c  | c[ 3] |( 6, 3):1304|  1304 |      0 |  7200| n[6] n[0] n[8]
 c[ 7]|   c  | c[ 3] |( 7, 3):1088|  1088 |      0 |  2400| n[7]
 c[ 9]|   c  | c[ 3] |( 9, 3):1538|  1538 |      0 |  7200| n[9] n[5] n[10]

Neighbors of Component[1]: <Component#:(link:cost)>
{<c[ 3]:( 1, 3): 432> <c[ 2]:( 1, 2):1210> <c[ 7]:( 1, 7):2034>}

Neighbors of Component[2]: <Component#:(link:cost)>
{<c[ 3]:( 2, 3): 988> <c[ 1]:( 1, 2):1210> <c[ 7]:( 2, 7):2252>}

Neighbors of Component[6]: <Component#:(link:cost)>
{<c[ 3]:( 6, 3):1304>}

Neighbors of Component[7]: <Component#:(link:cost)>
{<c[ 3]:( 7, 3):1088> <c[ 1]:( 7, 1):2034> <c[ 2]:( 2, 7):2252>}

Neighbors of Component[9]: <Component#:(link:cost)>
{<c[ 3]:( 9, 3):1538>}

A = {(8,0) (9,5) (5,10) (2,4) (6,0) (1,3) (2,3) (6,3) (7,3) (9,3) }
```

```
** Current Node Information ('*' indicates a root node) **
 Node#|Comp#|Weight|Degree|Depth
------+-----+------+------+-----
 n[ 0]|c[ 6]|  2400|     2|    2
 n[ 1]|c[ 1]|  2400|     1|    1
 n[ 2]|c[ 2]|  2400|     2|    1
*n[ 3]|c[ 3]| 69600|     5|    0
 n[ 4]|c[ 2]|  2400|     1|    2
 n[ 5]|c[ 9]|  2400|     2|    2
 n[ 6]|c[ 6]|  2400|     2|    1
 n[ 7]|c[ 7]|  2400|     1|    1
 n[ 8]|c[ 6]|  2400|     1|    3
 n[ 9]|c[ 9]|  2400|     2|    1
 n[10]|c[ 9]|  2400|     1|    3


*** Complete EW Algorithm ***

The accepted links and the cost are following

   Link    | Cost
  ---------+------
  (8,0)    | 806
  (9,5)    | 546
  (5,10)   | 1196
  (2,4)    | 2112
  (6,0)    | 1248
  (1,3)    | 432
  (2,3)    | 988
  (6,3)    | 1304
  (7,3)    | 1088
  (9,3)    | 1538
----------------------------
Total Network Cost = 11258
```

D.3 The Degree-Constraint Problem

```
The Number of Nodes : 11
Root Node : 3
Weight_limit : 9600
Order_limit : unconstrained
Degree_limit : 2
Depth_limit : unconstrained
 *** <Cost Table> ***
```

Node	0	1	2	3	4	5	6	7	8	9	10
0	200	3434	3496	3366	2730	3148	1248	1604	806	2864	3404
1	3434	200	1210	432	2850	3510	2416	2034	3218	2882	3722
2	3496	1210	200	988	2112	2700	2634	2252	3110	2104	2876
3	3366	432	988	200	2636	2080	1304	1088	1456	1538	2350
4	2730	2850	2112	2636	200	2166	2718	2372	2976	1602	2322
5	3148	3510	2700	2080	2166	200	2362	2228	1946	546	1196
6	1248	2416	2634	1304	2718	2362	200	2426	3266	2164	2904
7	1604	2034	2252	1088	2372	2228	2426	200	1302	1294	2030
8	806	3218	3110	1456	2976	1946	3266	1302	200	3296	4102
9	2864	2882	2104	1538	1602	546	2164	1294	3296	200	3784
10	3404	3722	2876	2350	2322	1196	2904	2030	4102	3784	200

```
 ** Current Node Information ('*' indicates a root node) **
```

Node#	Comp#	Weight	Degree	Depth
n[0]	c[0]	2400	0	1
n[1]	c[1]	2400	0	1
n[2]	c[2]	2400	0	1
*n[3]	c[3]	69600	0	0
n[4]	c[4]	2400	0	1
n[5]	c[5]	2400	0	1
n[6]	c[6]	2400	0	1
n[7]	c[7]	2400	0	1
n[8]	c[8]	2400	0	1
n[9]	c[9]	2400	0	1
n[10]	c[10]	2400	0	1

```
 ** Current Component Information **
```

<Comp#	ConRt	Nearest	Link	:Cost	CostRt	Tradeoff	Weight	Nodes >
c[0]	nc	c[8]	(0, 8): 806	3366	-2560	2400	n[0]	
c[1]	nc	c[3]	(1, 3): 432	432	0	2400	n[1]	
c[2]	nc	c[3]	(2, 3): 988	988	0	2400	n[2]	
c[3]	*c	c[3]	(3, 3): 200	200	0	69600	n[3]	
c[4]	nc	c[9]	(4, 9):1602	2636	-1034	2400	n[4]	
c[5]	nc	c[9]	(5, 9): 546	2080	-1534	2400	n[5]	
c[6]	nc	c[0]	(6, 0):1248	1304	-56	2400	n[6]	
c[7]	nc	c[3]	(7, 3):1088	1088	0	2400	n[7]	
c[8]	nc	c[0]	(8, 0): 806	1456	-650	2400	n[8]	
c[9]	nc	c[5]	(9, 5): 546	1538	-992	2400	n[9]	
c[10]	nc	c[5]	(10, 5):1196	2350	-1154	2400	n[10]	

```
Neighbors of Component[0]: <Component#:(link:cost)>
{<c[ 8]:( 0, 8): 806> <c[ 3]:( 0, 3):3366> <c[ 1]:( 0, 1):3434> <c[ 2]:( 0, 2):3496>
 <c[ 4]:( 0, 4):2730> <c[ 5]:( 0, 5):3148> <c[ 6]:( 0, 6):1248> <c[ 7]:( 0, 7):1604>
 <c[ 9]:( 0, 9):2864> <c[10]:( 0,10):3404>}
```

```
Neighbors of Component[1]: <Component#:(link:cost)>
{<c[ 3]:( 1, 3): 432> <c[ 0]:( 1, 0):3434> <c[ 2]:( 1, 2):1210> <c[ 4]:( 1, 4):2850>
 <c[ 5]:( 1, 5):3510> <c[ 6]:( 1, 6):2416> <c[ 7]:( 1, 7):2034> <c[ 8]:( 1, 8):3218>
 <c[ 9]:( 1, 9):2882> <c[10]:( 1,10):3722>}

Neighbors of Component[2]: <Component#:(link:cost)>
{<c[ 3]:( 2, 3): 988> <c[ 0]:( 2, 0):3496> <c[ 1]:( 2, 1):1210> <c[ 4]:( 2, 4):2112>
 <c[ 5]:( 2, 5):2700> <c[ 6]:( 2, 6):2634> <c[ 7]:( 2, 7):2252> <c[ 8]:( 2, 8):3110>
 <c[ 9]:( 2, 9):2104> <c[10]:( 2,10):2876>}

Neighbors of Component[4]: <Component#:(link:cost)>
{<c[ 9]:( 4, 9):1602> <c[ 3]:( 4, 3):2636> <c[ 0]:( 4, 0):2730> <c[ 1]:( 4, 1):2850>
 <c[ 2]:( 4, 2):2112> <c[ 5]:( 4, 5):2166> <c[ 6]:( 4, 6):2718> <c[ 7]:( 4, 7):2372>
 <c[ 8]:( 4, 8):2976> <c[10]:( 4,10):2322>}

Neighbors of Component[5]: <Component#:(link:cost)>
{<c[ 9]:( 5, 9): 546> <c[ 3]:( 5, 3):2080> <c[ 0]:( 5, 0):3148> <c[ 1]:( 5, 1):3510>
 <c[ 2]:( 5, 2):2700> <c[ 4]:( 5, 4):2166> <c[ 6]:( 5, 6):2362> <c[ 7]:( 5, 7):2228>
 <c[ 8]:( 5, 8):1946> <c[10]:( 5,10):1196>}

Neighbors of Component[6]: <Component#:(link:cost)>
{<c[ 0]:( 6, 0):1248> <c[ 3]:( 6, 3):1304> <c[ 1]:( 6, 1):2416> <c[ 2]:( 6, 2):2634>
 <c[ 4]:( 6, 4):2718> <c[ 5]:( 6, 5):2362> <c[ 7]:( 6, 7):2426> <c[ 8]:( 6, 8):3266>
 <c[ 9]:( 6, 9):2164> <c[10]:( 6,10):2904>}

Neighbors of Component[7]: <Component#:(link:cost)>
{<c[ 3]:( 7, 3):1088> <c[ 0]:( 7, 0):1604> <c[ 1]:( 7, 1):2034> <c[ 2]:( 7, 2):2252>
 <c[ 4]:( 7, 4):2372> <c[ 5]:( 7, 5):2228> <c[ 6]:( 7, 6):2426> <c[ 8]:( 7, 8):1302>
 <c[ 9]:( 7, 9):1294> <c[10]:( 7,10):2030>}

Neighbors of Component[8]: <Component#:(link:cost)>
{<c[ 0]:( 8, 0): 806> <c[ 3]:( 8, 3):1456> <c[ 1]:( 8, 1):3218> <c[ 2]:( 8, 2):3110>
 <c[ 4]:( 8, 4):2976> <c[ 5]:( 8, 5):1946> <c[ 6]:( 8, 6):3266> <c[ 7]:( 8, 7):1302>
 <c[ 9]:( 8, 9):3296> <c[10]:( 8,10):4102>}

Neighbors of Component[9]: <Component#:(link:cost)>
{<c[ 5]:( 9, 5): 546> <c[ 3]:( 9, 3):1538> <c[ 0]:( 9, 0):2864> <c[ 1]:( 9, 1):2882>
 <c[ 2]:( 9, 2):2104> <c[ 4]:( 9, 4):1602> <c[ 6]:( 9, 6):2164> <c[ 7]:( 9, 7):1294>
 <c[ 8]:( 9, 8):3296> <c[10]:( 9,10):3784>}

Neighbors of Component[10]: <Component#:(link:cost)>
{<c[ 5]:(10, 5):1196> <c[ 3]:(10, 3):2350> <c[ 0]:(10, 0):3404> <c[ 1]:(10, 1):3722>
 <c[ 2]:(10, 2):2876> <c[ 4]:(10, 4):2322> <c[ 6]:(10, 6):2904> <c[ 7]:(10, 7):2030>
 <c[ 8]:(10, 8):4102> <c[ 9]:(10, 9):3784>}

A = {}

 *** Start EW Algorithm ***

1): The smallest Tradeoff(c[0])=-2560
   Merge Comp(0) and Comp(8) by a link(0,8)

 Link(8,1) is the representative link between c[1] and c[8]
 Update Tradeoff(c[1]) = 432 - 432 = 0
 Link(8,2) is the representative link between c[2] and c[8]
 Update Tradeoff(c[2]) = 988 - 988 = 0
 Link(0,4) is the representative link between c[4] and c[8]
 Update Tradeoff(c[4]) = 1602 - 2636 = -1034
```

```
Link(0,5) is the representative link between c[5] and c[8]
Update Tradeoff(c[5]) = 546 - 2080 = -1534
Link(0,6) is the representative link between c[6] and c[8]
Update Tradeoff(c[6]) = 1248 - 1304 = -56
Link(8,7) is the representative link between c[7] and c[8]
Update Tradeoff(c[7]) = 1088 - 1088 = 0
Link(0,9) is the representative link between c[9] and c[8]
Update Tradeoff(c[9]) = 546 - 1538 = -992
Link(0,10) is the representative link between c[10] and c[8]
Update Tradeoff(c[10]) = 1196 - 2350 = -1154
Update Tradeoff(c[8]) = 1248 - 1456 = -208

 ** Current Component Information **
<Comp#|ConRt |Nearest|  Link :Cost|CostRt |Tradeoff|Weight|Nodes >
 -----+------+-------+-----------+-------+--------+------+------
 c[ 1]|  nc  | c[ 3] |( 1, 3): 432|   432 |      0 |  2400| n[1]
 c[ 2]|  nc  | c[ 3] |( 2, 3): 988|   988 |      0 |  2400| n[2]
 c[ 3]|  *c  | c[ 3] |( 3, 3): 200|   200 |      0 | 69600| n[3]
 c[ 4]|  nc  | c[ 9] |( 4, 9):1602|  2636 |  -1034 |  2400| n[4]
 c[ 5]|  nc  | c[ 9] |( 5, 9): 546|  2080 |  -1534 |  2400| n[5]
 c[ 6]|  nc  | c[ 8] |( 0, 6):1248|  1304 |    -56 |  2400| n[6]
 c[ 7]|  nc  | c[ 3] |( 7, 3):1088|  1088 |      0 |  2400| n[7]
 c[ 8]|  nc  | c[ 6] |( 0, 6):1248|  1456 |   -208 |  4800| n[8] n[0]
 c[ 9]|  nc  | c[ 5] |( 9, 5): 546|  1538 |   -992 |  2400| n[9]
 c[10]|  nc  | c[ 5] |(10, 5):1196|  2350 |  -1154 |  2400| n[10]

Neighbors of Component[1]: <Component#:(link:cost)>
{<c[ 3]:( 1, 3): 432> <c[ 2]:( 1, 2):1210> <c[ 4]:( 1, 4):2850> <c[ 5]:( 1, 5):3510>
 <c[ 6]:( 1, 6):2416> <c[ 7]:( 1, 7):2034> <c[ 8]:( 1, 8):3218> <c[ 9]:( 1, 9):2882>
 <c[10]:( 1,10):3722>}

Neighbors of Component[2]: <Component#:(link:cost)>
{<c[ 3]:( 2, 3): 988> <c[ 1]:( 2, 1):1210> <c[ 4]:( 2, 4):2112> <c[ 5]:( 2, 5):2700>
 <c[ 6]:( 2, 6):2634> <c[ 7]:( 2, 7):2252> <c[ 8]:( 2, 8):3110> <c[ 9]:( 2, 9):2104>
 <c[10]:( 2,10):2876>}

Neighbors of Component[4]: <Component#:(link:cost)>
{<c[ 9]:( 4, 9):1602> <c[ 3]:( 4, 3):2636> <c[ 1]:( 4, 1):2850> <c[ 2]:( 4, 2):2112>
 <c[ 5]:( 4, 5):2166> <c[ 6]:( 4, 6):2718> <c[ 7]:( 4, 7):2372> <c[ 8]:( 0, 4):2730>
 <c[10]:( 4,10):2322>}

Neighbors of Component[5]: <Component#:(link:cost)>
{<c[ 9]:( 5, 9): 546> <c[ 3]:( 5, 3):2080> <c[ 1]:( 5, 1):3510> <c[ 2]:( 5, 2):2700>
 <c[ 4]:( 5, 4):2166> <c[ 6]:( 5, 6):2362> <c[ 7]:( 5, 7):2228> <c[ 8]:( 0, 5):3148>
 <c[10]:( 5,10):1196>}

Neighbors of Component[6]: <Component#:(link:cost)>
{<c[ 8]:( 0, 6):1248> <c[ 3]:( 6, 3):1304> <c[ 1]:( 6, 1):2416> <c[ 2]:( 6, 2):2634>
 <c[ 4]:( 6, 4):2718> <c[ 5]:( 6, 5):2362> <c[ 7]:( 6, 7):2426> <c[ 9]:( 6, 9):2164>
 <c[10]:( 6,10):2904>}

Neighbors of Component[7]: <Component#:(link:cost)>
{<c[ 3]:( 7, 3):1088> <c[ 1]:( 7, 1):2034> <c[ 2]:( 7, 2):2252> <c[ 4]:( 7, 4):2372>
 <c[ 5]:( 7, 5):2228> <c[ 6]:( 7, 6):2426> <c[ 8]:( 7, 8):1302> <c[ 9]:( 7, 9):1294>
 <c[10]:( 7,10):2030>}

Neighbors of Component[8]: <Component#:(link:cost)>
{<c[ 6]:( 0, 6):1248> <c[ 3]:( 8, 3):1456> <c[ 1]:( 1, 8):3218> <c[ 2]:( 2, 8):3110>
```

```
 <c[ 4]:( 0, 4):2730> <c[ 5]:( 0, 5):3148> <c[ 7]:( 7, 8):1302> <c[ 9]:( 0, 9):2864>
 <c[10]:( 0,10):3404>}

Neighbors of Component[9]: <Component#:(link:cost)>
{<c[ 5]:( 9, 5): 546> <c[ 3]:( 9, 3):1538> <c[ 1]:( 9, 1):2882> <c[ 2]:( 9, 2):2104>
 <c[ 4]:( 9, 4):1602> <c[ 6]:( 9, 6):2164> <c[ 7]:( 9, 7):1294> <c[ 8]:( 0, 9):2864>
 <c[10]:( 9,10):3784>}

Neighbors of Component[10]: <Component#:(link:cost)>
{<c[ 5]:(10, 5):1196> <c[ 3]:(10, 3):2350> <c[ 1]:(10, 1):3722> <c[ 2]:(10, 2):2876>
 <c[ 4]:(10, 4):2322> <c[ 6]:(10, 6):2904> <c[ 7]:(10, 7):2030> <c[ 8]:( 0,10):3404>
 <c[ 9]:(10, 9):3784>}

A = {(8,0) }

 ** Current Node Information ('*' indicates a root node) **
  Node#|Comp#|Weight|Degree|Depth
 ------+-----+------+------+-----
  n[ 0]|c[ 8]|  2400|     1|    2
  n[ 1]|c[ 1]|  2400|     0|    1
  n[ 2]|c[ 2]|  2400|     0|    1
 *n[ 3]|c[ 3]| 69600|     0|    0
  n[ 4]|c[ 4]|  2400|     0|    1
  n[ 5]|c[ 5]|  2400|     0|    1
  n[ 6]|c[ 6]|  2400|     0|    1
  n[ 7]|c[ 7]|  2400|     0|    1
  n[ 8]|c[ 8]|  2400|     1|    1
  n[ 9]|c[ 9]|  2400|     0|    1
  n[10]|c[10]|  2400|     0|    1

2): The smallest Tradeoff(c[5])=-1534
   Merge Comp(5) and Comp(9) by a link(5,9)

 Link(9,1) is the representative link between c[1] and c[9]
 Update Tradeoff(c[1]) = 432 - 432 = 0
 Link(9,2) is the representative link between c[2] and c[9]
 Update Tradeoff(c[2]) = 988 - 988 = 0
 Link(5,4) is the representative link between c[4] and c[9]
 Update Tradeoff(c[4]) = 2112 - 2636 = -524
 Link(9,6) is the representative link between c[6] and c[9]
 Update Tradeoff(c[6]) = 1248 - 1304 = -56
 Link(9,7) is the representative link between c[7] and c[9]
 Update Tradeoff(c[7]) = 1088 - 1088 = 0
 Link(5,8) is the representative link between c[8] and c[9]
 Update Tradeoff(c[8]) = 1248 - 1456 = -208
 Link(5,10) is the representative link between c[10] and c[9]
 Update Tradeoff(c[10]) = 1196 - 2350 = -1154
 Update Tradeoff(c[9]) = 1196 - 1538 = -342

 ** Current Component Information **
<Comp#|ConRt |Nearest|  Link :Cost|CostRt |Tradeoff|Weight|Nodes >
 -----+------+-------+-----------+-------+--------+------+------
 c[ 1]|  nc  | c[ 3] |( 1, 3): 432|   432 |      0 |  2400| n[1]
 c[ 2]|  nc  | c[ 3] |( 2, 3): 988|   988 |      0 |  2400| n[2]
 c[ 3]|  *c  | c[ 3] |( 3, 3): 200|   200 |      0 | 69600| n[3]
 c[ 4]|  nc  | c[ 2] |( 4, 2):2112|  2636 |   -524 |  2400| n[4]
 c[ 6]|  nc  | c[ 8] |( 0, 6):1248|  1304 |    -56 |  2400| n[6]
 c[ 7]|  nc  | c[ 3] |( 7, 3):1088|  1088 |      0 |  2400| n[7]
```

```
 c[ 8]|  nc  | c[ 6] |( 0, 6):1248|  1456 |  -208 |  4800| n[8] n[0]
 c[ 9]|  nc  | c[10] |( 5,10):1196|  1538 |  -342 |  4800| n[9] n[5]
 c[10]|  nc  | c[ 9] |( 5,10):1196|  2350 | -1154 |  2400| n[10]

Neighbors of Component[1]: <Component#:(link:cost)>
{<c[ 3]:( 1, 3): 432> <c[ 2]:( 1, 2):1210> <c[ 4]:( 1, 4):2850> <c[ 6]:( 1, 6):2416>
 <c[ 7]:( 1, 7):2034> <c[ 8]:( 1, 8):3218> <c[ 9]:( 1, 9):2882> <c[10]:( 1,10):3722>}

Neighbors of Component[2]: <Component#:(link:cost)>
{<c[ 3]:( 2, 3): 988> <c[ 1]:( 2, 1):1210> <c[ 4]:( 2, 4):2112> <c[ 6]:( 2, 6):2634>
 <c[ 7]:( 2, 7):2252> <c[ 8]:( 2, 8):3110> <c[ 9]:( 2, 9):2104> <c[10]:( 2,10):2876>}

Neighbors of Component[4]: <Component#:(link:cost)>
{<c[ 2]:( 4, 2):2112> <c[ 9]:( 4, 5):2166> <c[ 3]:( 4, 3):2636> <c[ 1]:( 4, 1):2850>
 <c[ 6]:( 4, 6):2718> <c[ 7]:( 4, 7):2372> <c[ 8]:( 0, 4):2730> <c[10]:( 4,10):2322>}

Neighbors of Component[6]: <Component#:(link:cost)>
{<c[ 8]:( 0, 6):1248> <c[ 3]:( 6, 3):1304> <c[ 1]:( 6, 1):2416> <c[ 2]:( 6, 2):2634>
 <c[ 4]:( 6, 4):2718> <c[ 7]:( 6, 7):2426> <c[ 9]:( 6, 9):2164> <c[10]:( 6,10):2904>}

Neighbors of Component[7]: <Component#:(link:cost)>
{<c[ 3]:( 7, 3):1088> <c[ 1]:( 7, 1):2034> <c[ 2]:( 7, 2):2252> <c[ 4]:( 7, 4):2372>
 <c[ 6]:( 7, 6):2426> <c[ 8]:( 7, 8):1302> <c[ 9]:( 7, 9):1294> <c[10]:( 7,10):2030>}

Neighbors of Component[8]: <Component#:(link:cost)>
{<c[ 6]:( 0, 6):1248> <c[ 3]:( 8, 3):1456> <c[ 1]:( 1, 8):3218> <c[ 2]:( 2, 8):3110>
 <c[ 4]:( 0, 4):2730> <c[ 7]:( 7, 8):1302> <c[ 9]:( 5, 8):1946> <c[10]:( 0,10):3404>}

Neighbors of Component[9]: <Component#:(link:cost)>
{<c[10]:( 5,10):1196> <c[ 3]:( 9, 3):1538> <c[ 1]:( 1, 9):2882> <c[ 2]:( 2, 9):2104>
 <c[ 4]:( 4, 5):2166> <c[ 6]:( 6, 9):2164> <c[ 7]:( 7, 9):1294> <c[ 8]:( 5, 8):1946>}

Neighbors of Component[10]: <Component#:(link:cost)>
{<c[ 9]:( 5,10):1196> <c[ 3]:(10, 3):2350> <c[ 1]:(10, 1):3722> <c[ 2]:(10, 2):2876>
 <c[ 4]:(10, 4):2322> <c[ 6]:(10, 6):2904> <c[ 7]:(10, 7):2030> <c[ 8]:( 0,10):3404>}

A = {(8,0) (9,5) }

 ** Current Node Information ('*' indicates a root node) **
  Node#|Comp#|Weight|Degree|Depth
 ------+-----+------+------+-----
  n[ 0]|c[ 8]|  2400|     1|    2
  n[ 1]|c[ 1]|  2400|     0|    1
  n[ 2]|c[ 2]|  2400|     0|    1
 *n[ 3]|c[ 3]| 69600|     0|    0
  n[ 4]|c[ 4]|  2400|     0|    1
  n[ 5]|c[ 9]|  2400|     1|    2
  n[ 6]|c[ 6]|  2400|     0|    1
  n[ 7]|c[ 7]|  2400|     0|    1
  n[ 8]|c[ 8]|  2400|     1|    1
  n[ 9]|c[ 9]|  2400|     1|    1
  n[10]|c[10]|  2400|     0|    1

3): The smallest Tradeoff(c[10])=-1154
   Merge Comp(10) and Comp(9) by a link(5,10)

 Link(9,1) is the representative link between c[1] and c[9]
 Update Tradeoff(c[1]) = 432 - 432 = 0
```

```
Link(9,2) is the representative link between c[2] and c[9]
Update Tradeoff(c[2]) = 988 - 988 = 0
A link (4,5) is discarded because of violating the degree-constraint.
Link(10,4) is the representative link between c[4] and c[9]
Update Tradeoff(c[4]) = 2112 - 2636 = -524
Link(9,6) is the representative link between c[6] and c[9]
Update Tradeoff(c[6]) = 1248 - 1304 = -56
Link(9,7) is the representative link between c[7] and c[9]
Update Tradeoff(c[7]) = 1088 - 1088 = 0
Merging c[9] and c[8]) would violate the weight-constraint.
Destory a relation between Comp(9) and Comp(8)
Update Tradeoff(c[8]) = 1248 - 1456 = -208
Update Tradeoff(c[9]) = 1294 - 1538 = -244

 ** Current Component Information **
<Comp#|ConRt |Nearest|  Link :Cost|CostRt |Tradeoff|Weight|Nodes >
 -----+------+-------+------------+-------+--------+------+------
 c[ 1]|  nc  | c[ 3] |( 1, 3): 432|   432 |      0 |  2400| n[1]
 c[ 2]|  nc  | c[ 3] |( 2, 3): 988|   988 |      0 |  2400| n[2]
 c[ 3]|  *c  | c[ 3] |( 3, 3): 200|   200 |      0 | 69600| n[3]
 c[ 4]|  nc  | c[ 2] |( 4, 2):2112|  2636 |   -524 |  2400| n[4]
 c[ 6]|  nc  | c[ 8] |( 0, 6):1248|  1304 |    -56 |  2400| n[6]
 c[ 7]|  nc  | c[ 3] |( 7, 3):1088|  1088 |      0 |  2400| n[7]
 c[ 8]|  nc  | c[ 6] |( 0, 6):1248|  1456 |   -208 |  4800| n[8] n[0]
 c[ 9]|  nc  | c[ 7] |( 7, 9):1294|  1538 |   -244 |  7200| n[9] n[5] n[10]

Neighbors of Component[1]: <Component#:(link:cost)>
{<c[ 3]:( 1, 3): 432> <c[ 2]:( 1, 2):1210> <c[ 4]:( 1, 4):2850> <c[ 6]:( 1, 6):2416>
 <c[ 7]:( 1, 7):2034> <c[ 8]:( 1, 8):3218> <c[ 9]:( 1, 9):2882>}

Neighbors of Component[2]: <Component#:(link:cost)>
{<c[ 3]:( 2, 3): 988> <c[ 1]:( 2, 1):1210> <c[ 4]:( 2, 4):2112> <c[ 6]:( 2, 6):2634>
 <c[ 7]:( 2, 7):2252> <c[ 8]:( 2, 8):3110> <c[ 9]:( 2, 9):2104>}

Neighbors of Component[4]: <Component#:(link:cost)>
{<c[ 2]:( 4, 2):2112> <c[ 9]:( 4,10):2322> <c[ 3]:( 4, 3):2636> <c[ 1]:( 4, 1):2850>
 <c[ 6]:( 4, 6):2718> <c[ 7]:( 4, 7):2372> <c[ 8]:( 0, 4):2730>}

Neighbors of Component[6]: <Component#:(link:cost)>
{<c[ 8]:( 0, 6):1248> <c[ 3]:( 6, 3):1304> <c[ 1]:( 6, 1):2416> <c[ 2]:( 6, 2):2634>
 <c[ 4]:( 6, 4):2718> <c[ 7]:( 6, 7):2426> <c[ 9]:( 6, 9):2164>}

Neighbors of Component[7]: <Component#:(link:cost)>
{<c[ 3]:( 7, 3):1088> <c[ 1]:( 7, 1):2034> <c[ 2]:( 7, 2):2252> <c[ 4]:( 7, 4):2372>
 <c[ 6]:( 7, 6):2426> <c[ 8]:( 7, 8):1302> <c[ 9]:( 7, 9):1294>}

Neighbors of Component[8]: <Component#:(link:cost)>
{<c[ 6]:( 0, 6):1248> <c[ 3]:( 8, 3):1456> <c[ 1]:( 1, 8):3218> <c[ 2]:( 2, 8):3110>
 <c[ 4]:( 0, 4):2730> <c[ 7]:( 7, 8):1302>}

Neighbors of Component[9]: <Component#:(link:cost)>
{<c[ 7]:( 7, 9):1294> <c[ 3]:( 9, 3):1538> <c[ 1]:( 1, 9):2882> <c[ 2]:( 2, 9):2104>
 <c[ 4]:( 4,10):2322> <c[ 6]:( 6, 9):2164>}

A = {(8,0) (9,5) (5,10) }

 ** Current Node Information ('*' indicates a root node) **
  Node#|Comp#|Weight|Degree|Depth
```

```
------+-----+------+------+-----
 n[ 0]|c[ 8]|  2400|     1|    2
 n[ 1]|c[ 1]|  2400|     0|    1
 n[ 2]|c[ 2]|  2400|     0|    1
*n[ 3]|c[ 3]| 69600|     0|    0
 n[ 4]|c[ 4]|  2400|     0|    1
 n[ 5]|c[ 9]|  2400|     2|    2
 n[ 6]|c[ 6]|  2400|     0|    1
 n[ 7]|c[ 7]|  2400|     0|    1
 n[ 8]|c[ 8]|  2400|     1|    1
 n[ 9]|c[ 9]|  2400|     1|    1
 n[10]|c[ 9]|  2400|     1|    3

4): The smallest Tradeoff(c[4])=-524
   Merge Comp(4) and Comp(2) by a link(4,2)

 Link(2,1) is the representative link between c[1] and c[2]
 Update Tradeoff(c[1]) = 432 - 432 = 0
 Link(4,6) is the representative link between c[6] and c[2]
 Update Tradeoff(c[6]) = 1248 - 1304 = -56
 Link(4,7) is the representative link between c[7] and c[2]
 Update Tradeoff(c[7]) = 1088 - 1088 = 0
 Link(4,0) is the representative link between c[8] and c[2]
 Update Tradeoff(c[8]) = 1248 - 1456 = -208
 Merging c[2] and c[9]) would violate the weight-constraint.
 Destory a relation between Comp(2) and Comp(9)
 Update Tradeoff(c[9]) = 1294 - 1538 = -244
 Update Tradeoff(c[2]) = 988 - 988 = 0

 ** Current Component Information **
<Comp#|ConRt |Nearest|  Link :Cost|CostRt |Tradeoff|Weight|Nodes >
 -----+------+-------+-----------+-------+--------+------+------
 c[ 1]|  nc  | c[ 3] |( 1, 3): 432|   432 |      0 |  2400| n[1]
 c[ 2]|  nc  | c[ 3] |( 2, 3): 988|   988 |      0 |  4800| n[2] n[4]
 c[ 3]|  *c  | c[ 3] |( 3, 3): 200|   200 |      0 | 69600| n[3]
 c[ 6]|  nc  | c[ 8] |( 0, 6):1248|  1304 |    -56 |  2400| n[6]
 c[ 7]|  nc  | c[ 3] |( 7, 3):1088|  1088 |      0 |  2400| n[7]
 c[ 8]|  nc  | c[ 6] |( 0, 6):1248|  1456 |   -208 |  4800| n[8] n[0]
 c[ 9]|  nc  | c[ 7] |( 7, 9):1294|  1538 |   -244 |  7200| n[9] n[5] n[10]

Neighbors of Component[1]: <Component#:(link:cost)>
{<c[ 3]:( 1, 3): 432> <c[ 2]:( 1, 2):1210> <c[ 6]:( 1, 6):2416> <c[ 7]:( 1, 7):2034>
 <c[ 8]:( 1, 8):3218> <c[ 9]:( 1, 9):2882>}

Neighbors of Component[2]: <Component#:(link:cost)>
{<c[ 3]:( 2, 3): 988> <c[ 1]:( 1, 2):1210> <c[ 6]:( 4, 6):2718> <c[ 7]:( 4, 7):2372>
 <c[ 8]:( 0, 4):2730>}

Neighbors of Component[6]: <Component#:(link:cost)>
{<c[ 8]:( 0, 6):1248> <c[ 3]:( 6, 3):1304> <c[ 1]:( 6, 1):2416> <c[ 2]:( 4, 6):2718>
 <c[ 7]:( 6, 7):2426> <c[ 9]:( 6, 9):2164>}

Neighbors of Component[7]: <Component#:(link:cost)>
{<c[ 3]:( 7, 3):1088> <c[ 1]:( 7, 1):2034> <c[ 2]:( 4, 7):2372> <c[ 6]:( 7, 6):2426>
 <c[ 8]:( 7, 8):1302> <c[ 9]:( 7, 9):1294>}

Neighbors of Component[8]: <Component#:(link:cost)>
{<c[ 6]:( 0, 6):1248> <c[ 3]:( 8, 3):1456> <c[ 1]:( 1, 8):3218> <c[ 2]:( 0, 4):2730>
```

```
 <c[ 7]:( 7, 8):1302>}

Neighbors of Component[9]: <Component#:(link:cost)>
{<c[ 7]:( 7, 9):1294> <c[ 3]:( 9, 3):1538> <c[ 1]:( 1, 9):2882> <c[ 6]:( 6, 9):2164>}

A = {(8,0) (9,5) (5,10) (2,4) }

 ** Current Node Information ('*' indicates a root node) **
  Node#|Comp#|Weight|Degree|Depth
 ------+-----+------+------+-----
  n[ 0]|c[ 8]|  2400|     1|    2
  n[ 1]|c[ 1]|  2400|     0|    1
  n[ 2]|c[ 2]|  2400|     1|    1
 *n[ 3]|c[ 3]| 69600|     0|    0
  n[ 4]|c[ 2]|  2400|     1|    2
  n[ 5]|c[ 9]|  2400|     2|    2
  n[ 6]|c[ 6]|  2400|     0|    1
  n[ 7]|c[ 7]|  2400|     0|    1
  n[ 8]|c[ 8]|  2400|     1|    1
  n[ 9]|c[ 9]|  2400|     1|    1
  n[10]|c[ 9]|  2400|     1|    3

5): The smallest Tradeoff(c[9])=-244
   Merge Comp(9) and Comp(7) by a link(7,9)

 Merging c[7] and c[1]) would violate the weight-constraint.
 Destory a relation between Comp(7) and Comp(1)
 Update Tradeoff(c[1]) = 432 - 432 = 0
 Destory a relation between Comp(7) and Comp(2)
 Update Tradeoff(c[2]) = 988 - 988 = 0
 Merging c[7] and c[6]) would violate the weight-constraint.
 Destory a relation between Comp(7) and Comp(6)
 Update Tradeoff(c[6]) = 1248 - 1304 = -56
 Destory a relation between Comp(7) and Comp(8)
 Update Tradeoff(c[8]) = 1248 - 1456 = -208
 Update Tradeoff(c[7]) = 1088 - 1088 = 0

 ** Current Component Information **
<Comp#|ConRt |Nearest|  Link :Cost|CostRt |Tradeoff|Weight|Nodes >
 -----+------+-------+-----------+-------+--------+------+------
 c[ 1]|  nc  | c[ 3] |( 1, 3): 432|  432  |     0  |  2400| n[1]
 c[ 2]|  nc  | c[ 3] |( 2, 3): 988|  988  |     0  |  4800| n[2] n[4]
 c[ 3]|  *c  | c[ 3] |( 3, 3): 200|  200  |     0  | 69600| n[3]
 c[ 6]|  nc  | c[ 8] |( 0, 6):1248|  1304 |   -56  |  2400| n[6]
 c[ 7]|  nc  | c[ 3] |( 7, 3):1088|  1088 |     0  |  9600| n[7] n[10] n[5] n[9]
 c[ 8]|  nc  | c[ 6] |( 0, 6):1248|  1456 |  -208  |  4800| n[8] n[0]

Neighbors of Component[1]: <Component#:(link:cost)>
{<c[ 3]:( 1, 3): 432> <c[ 2]:( 1, 2):1210> <c[ 6]:( 1, 6):2416> <c[ 8]:( 1, 8):3218>}

Neighbors of Component[2]: <Component#:(link:cost)>
{<c[ 3]:( 2, 3): 988> <c[ 1]:( 1, 2):1210> <c[ 6]:( 4, 6):2718> <c[ 8]:( 0, 4):2730>}

Neighbors of Component[6]: <Component#:(link:cost)>
{<c[ 8]:( 0, 6):1248> <c[ 3]:( 6, 3):1304> <c[ 1]:( 6, 1):2416> <c[ 2]:( 4, 6):2718>}

Neighbors of Component[7]: <Component#:(link:cost)>
{<c[ 3]:( 7, 3):1088>}
```

```
Neighbors of Component[8]: <Component#:(link:cost)>
{<c[ 6]:( 0, 6):1248> <c[ 3]:( 8, 3):1456> <c[ 1]:( 1, 8):3218> <c[ 2]:( 0, 4):2730>}

A = {(8,0) (9,5) (5,10) (2,4) (7,9) }

 ** Current Node Information ('*' indicates a root node) **
  Node#|Comp#|Weight|Degree|Depth
 ------+-----+------+------+-----
  n[ 0]|c[ 8]|  2400|     1|    2
  n[ 1]|c[ 1]|  2400|     0|    1
  n[ 2]|c[ 2]|  2400|     1|    1
 *n[ 3]|c[ 3]| 69600|     0|    0
  n[ 4]|c[ 2]|  2400|     1|    2
  n[ 5]|c[ 7]|  2400|     2|    3
  n[ 6]|c[ 6]|  2400|     0|    1
  n[ 7]|c[ 7]|  2400|     1|    1
  n[ 8]|c[ 8]|  2400|     1|    1
  n[ 9]|c[ 7]|  2400|     2|    2
  n[10]|c[ 7]|  2400|     1|    4

6): The smallest Tradeoff(c[8])=-208
   Merge Comp(8) and Comp(6) by a link(0,6)

 Link(6,1) is the representative link between c[1] and c[6]
 Update Tradeoff(c[1]) = 432 - 432 = 0
 Merging c[6] and c[2]) would violate the weight-constraint.
 Destory a relation between Comp(6) and Comp(2)
 Update Tradeoff(c[2]) = 988 - 988 = 0
 Update Tradeoff(c[6]) = 1304 - 1304 = 0

 ** Current Component Information **
<Comp#|ConRt |Nearest|  Link :Cost|CostRt |Tradeoff|Weight|Nodes >
 -----+------+-------+------------+-------+--------+------+------
 c[ 1]|  nc  | c[ 3] |( 1, 3): 432|   432 |      0 |  2400| n[1]
 c[ 2]|  nc  | c[ 3] |( 2, 3): 988|   988 |      0 |  4800| n[2] n[4]
 c[ 3]|  *c  | c[ 3] |( 3, 3): 200|   200 |      0 | 69600| n[3]
 c[ 6]|  nc  | c[ 3] |( 6, 3):1304|  1304 |      0 |  7200| n[6] n[0] n[8]
 c[ 7]|  nc  | c[ 3] |( 7, 3):1088|  1088 |      0 |  9600| n[7] n[10] n[5] n[9]

Neighbors of Component[1]: <Component#:(link:cost)>
{<c[ 3]:( 1, 3): 432> <c[ 2]:( 1, 2):1210> <c[ 6]:( 1, 6):2416>}

Neighbors of Component[2]: <Component#:(link:cost)>
{<c[ 3]:( 2, 3): 988> <c[ 1]:( 1, 2):1210>}

Neighbors of Component[6]: <Component#:(link:cost)>
{<c[ 3]:( 6, 3):1304> <c[ 1]:( 1, 6):2416>}

Neighbors of Component[7]: <Component#:(link:cost)>
{<c[ 3]:( 7, 3):1088>}

A = {(8,0) (9,5) (5,10) (2,4) (7,9) (6,0) }

 ** Current Node Information ('*' indicates a root node) **
  Node#|Comp#|Weight|Degree|Depth
 ------+-----+------+------+-----
  n[ 0]|c[ 6]|  2400|     2|    2
```

```
 n[ 1]|c[ 1]|  2400|     0|    1
 n[ 2]|c[ 2]|  2400|     1|    1
*n[ 3]|c[ 3]| 69600|     0|    0
 n[ 4]|c[ 2]|  2400|     1|    2
 n[ 5]|c[ 7]|  2400|     2|    3
 n[ 6]|c[ 6]|  2400|     1|    1
 n[ 7]|c[ 7]|  2400|     1|    1
 n[ 8]|c[ 6]|  2400|     1|    3
 n[ 9]|c[ 7]|  2400|     2|    2
 n[10]|c[ 7]|  2400|     1|    4

7): The smallest Tradeoff(c[1])=0
   Merge Comp(1) and Comp(3) by a link(1,3)


 ** Current Component Information **
<Comp#|ConRt |Nearest|  Link :Cost|CostRt |Tradeoff|Weight|Nodes >
 -----+------+-------+-----------+-------+--------+------+------
 c[ 1]|   c  | c[ 3] |( 1, 3): 432|   432 |      0 |  2400| n[1]
 c[ 2]|  nc  | c[ 3] |( 2, 3): 988|   988 |      0 |  4800| n[2] n[4]
 c[ 3]|  *c  | c[ 3] |( 3, 3): 200|   200 |      0 | 69600| n[3]
 c[ 6]|  nc  | c[ 3] |( 6, 3):1304|  1304 |      0 |  7200| n[6] n[0] n[8]
 c[ 7]|  nc  | c[ 3] |( 7, 3):1088|  1088 |      0 |  9600| n[7] n[10] n[5] n[9]

Neighbors of Component[1]: <Component#:(link:cost)>
{<c[ 3]:( 1, 3): 432> <c[ 2]:( 1, 2):1210> <c[ 6]:( 1, 6):2416>}

Neighbors of Component[2]: <Component#:(link:cost)>
{<c[ 3]:( 2, 3): 988> <c[ 1]:( 1, 2):1210>}

Neighbors of Component[6]: <Component#:(link:cost)>
{<c[ 3]:( 6, 3):1304> <c[ 1]:( 1, 6):2416>}

Neighbors of Component[7]: <Component#:(link:cost)>
{<c[ 3]:( 7, 3):1088>}

A = {(8,0) (9,5) (5,10) (2,4) (7,9) (6,0) (1,3) }

 ** Current Node Information ('*' indicates a root node) **
  Node#|Comp#|Weight|Degree|Depth
 ------+-----+------+------+-----
  n[ 0]|c[ 6]|  2400|     2|    2
  n[ 1]|c[ 1]|  2400|     1|    1
  n[ 2]|c[ 2]|  2400|     1|    1
 *n[ 3]|c[ 3]| 69600|     1|    0
  n[ 4]|c[ 2]|  2400|     1|    2
  n[ 5]|c[ 7]|  2400|     2|    3
  n[ 6]|c[ 6]|  2400|     1|    1
  n[ 7]|c[ 7]|  2400|     1|    1
  n[ 8]|c[ 6]|  2400|     1|    3
  n[ 9]|c[ 7]|  2400|     2|    2
  n[10]|c[ 7]|  2400|     1|    4

8): The smallest Tradeoff(c[2])=0
   Merge Comp(2) and Comp(3) by a link(2,3)


 ** Current Component Information **
```

```
<Comp#|ConRt |Nearest|  Link :Cost|CostRt |Tradeoff|Weight|Nodes >
-----+------+-------+-----------+-------+--------+------+------
 c[ 1]|   c  | c[ 3] |( 1, 3): 432|   432 |      0 |  2400| n[1]
 c[ 2]|   c  | c[ 3] |( 2, 3): 988|   988 |      0 |  4800| n[2] n[4]
 c[ 3]|  *c  | c[ 3] |( 3, 3): 200|   200 |      0 | 69600| n[3]
 c[ 6]|  nc  | c[ 3] |( 6, 3):1304|  1304 |      0 |  7200| n[6] n[0] n[8]
 c[ 7]|  nc  | c[ 3] |( 7, 3):1088|  1088 |      0 |  9600| n[7] n[10] n[5] n[9]

Neighbors of Component[1]: <Component#:(link:cost)>
{<c[ 3]:( 1, 3): 432> <c[ 2]:( 1, 2):1210> <c[ 6]:( 1, 6):2416>}

Neighbors of Component[2]: <Component#:(link:cost)>
{<c[ 3]:( 2, 3): 988> <c[ 1]:( 1, 2):1210>}

Neighbors of Component[6]: <Component#:(link:cost)>
{<c[ 3]:( 6, 3):1304> <c[ 1]:( 1, 6):2416>}

Neighbors of Component[7]: <Component#:(link:cost)>
{<c[ 3]:( 7, 3):1088>}

A = {(8,0) (9,5) (5,10) (2,4) (7,9) (6,0) (1,3) (2,3) }

 ** Current Node Information ('*' indicates a root node) **
  Node#|Comp#|Weight|Degree|Depth
 ------+-----+------+------+-----
  n[ 0]|c[ 6]|  2400|     2|    2
  n[ 1]|c[ 1]|  2400|     1|    1
  n[ 2]|c[ 2]|  2400|     2|    1
 *n[ 3]|c[ 3]| 69600|     2|    0
  n[ 4]|c[ 2]|  2400|     1|    2
  n[ 5]|c[ 7]|  2400|     2|    3
  n[ 6]|c[ 6]|  2400|     1|    1
  n[ 7]|c[ 7]|  2400|     1|    1
  n[ 8]|c[ 6]|  2400|     1|    3
  n[ 9]|c[ 7]|  2400|     2|    2
  n[10]|c[ 7]|  2400|     1|    4

9): The smallest Tradeoff(c[6])=0
   Merge Comp(6) and Comp(3) by a link(6,3)


 ** Current Component Information **
<Comp#|ConRt |Nearest|  Link :Cost|CostRt |Tradeoff|Weight|Nodes >
 -----+------+-------+-----------+-------+--------+------+------
 c[ 1]|   c  | c[ 3] |( 1, 3): 432|   432 |      0 |  2400| n[1]
 c[ 2]|   c  | c[ 3] |( 2, 3): 988|   988 |      0 |  4800| n[2] n[4]
 c[ 3]|  *c  | c[ 3] |( 3, 3): 200|   200 |      0 | 69600| n[3]
 c[ 6]|   c  | c[ 3] |( 6, 3):1304|  1304 |      0 |  7200| n[6] n[0] n[8]
 c[ 7]|  nc  | c[ 3] |( 7, 3):1088|  1088 |      0 |  9600| n[7] n[10] n[5] n[9]

Neighbors of Component[1]: <Component#:(link:cost)>
{<c[ 3]:( 1, 3): 432> <c[ 2]:( 1, 2):1210> <c[ 6]:( 1, 6):2416>}

Neighbors of Component[2]: <Component#:(link:cost)>
{<c[ 3]:( 2, 3): 988> <c[ 1]:( 1, 2):1210>}

Neighbors of Component[6]: <Component#:(link:cost)>
{<c[ 3]:( 6, 3):1304> <c[ 1]:( 1, 6):2416>}
```

```
Neighbors of Component[7]: <Component#:(link:cost)>
{<c[ 3]:( 7, 3):1088>}

A = {(8,0) (9,5) (5,10) (2,4) (7,9) (6,0) (1,3) (2,3) (6,3) }

 ** Current Node Information ('*' indicates a root node) **
  Node#|Comp#|Weight|Degree|Depth
 ------+-----+------+------+-----
  n[ 0]|c[ 6]|  2400|     2|    2
  n[ 1]|c[ 1]|  2400|     1|    1
  n[ 2]|c[ 2]|  2400|     2|    1
 *n[ 3]|c[ 3]| 69600|     3|    0
  n[ 4]|c[ 2]|  2400|     1|    2
  n[ 5]|c[ 7]|  2400|     2|    3
  n[ 6]|c[ 6]|  2400|     2|    1
  n[ 7]|c[ 7]|  2400|     1|    1
  n[ 8]|c[ 6]|  2400|     1|    3
  n[ 9]|c[ 7]|  2400|     2|    2
  n[10]|c[ 7]|  2400|     1|    4

10): The smallest Tradeoff(c[7])=0
   Merge Comp(7) and Comp(3) by a link(7,3)


 ** Current Component Information **
<Comp#|ConRt |Nearest|  Link :Cost|CostRt |Tradeoff|Weight|Nodes >
 -----+------+-------+-----------+-------+--------+------+------
 c[ 1]|   c  | c[ 3] |( 1, 3): 432|   432 |      0 |  2400| n[1]
 c[ 2]|   c  | c[ 3] |( 2, 3): 988|   988 |      0 |  4800| n[2] n[4]
 c[ 3]|  *c  | c[ 3] |( 3, 3): 200|   200 |      0 | 69600| n[3]
 c[ 6]|   c  | c[ 3] |( 6, 3):1304|  1304 |      0 |  7200| n[6] n[0] n[8]
 c[ 7]|   c  | c[ 3] |( 7, 3):1088|  1088 |      0 |  9600| n[7] n[10] n[5] n[9]

Neighbors of Component[1]: <Component#:(link:cost)>
{<c[ 3]:( 1, 3): 432> <c[ 2]:( 1, 2):1210> <c[ 6]:( 1, 6):2416>}

Neighbors of Component[2]: <Component#:(link:cost)>
{<c[ 3]:( 2, 3): 988> <c[ 1]:( 1, 2):1210>}

Neighbors of Component[6]: <Component#:(link:cost)>
{<c[ 3]:( 6, 3):1304> <c[ 1]:( 1, 6):2416>}

Neighbors of Component[7]: <Component#:(link:cost)>
{<c[ 3]:( 7, 3):1088>}

A = {(8,0) (9,5) (5,10) (2,4) (7,9) (6,0) (1,3) (2,3) (6,3) (7,3) }

 ** Current Node Information ('*' indicates a root node) **
  Node#|Comp#|Weight|Degree|Depth
 ------+-----+------+------+-----
  n[ 0]|c[ 6]|  2400|     2|    2
  n[ 1]|c[ 1]|  2400|     1|    1
  n[ 2]|c[ 2]|  2400|     2|    1
 *n[ 3]|c[ 3]| 69600|     4|    0
  n[ 4]|c[ 2]|  2400|     1|    2
  n[ 5]|c[ 7]|  2400|     2|    3
  n[ 6]|c[ 6]|  2400|     2|    1
```

```
 n[ 7]|c[ 7]|  2400|     2|    1
 n[ 8]|c[ 6]|  2400|     1|    3
 n[ 9]|c[ 7]|  2400|     2|    2
 n[10]|c[ 7]|  2400|     1|    4
```

*** Complete EW Algorithm ***

The accepted links and the cost are following

Link	Cost
(8,0)	806
(9,5)	546
(5,10)	1196
(2,4)	2112
(7,9)	1294
(6,0)	1248
(1,3)	432
(2,3)	988
(6,3)	1304
(7,3)	1088

Total Network Cost = 11014

D.4 The Depth-Constraint Problem

```
The Number of Nodes : 11
Root Node : 3
Weight_limit : 9600
Order_limit : unconstrained
Degree_limit : unconstrained
Depth_limit : 3
 *** <Cost Table> ***
```

Node	0	1	2	3	4	5	6	7	8	9	10
0	200	3434	3496	3366	2730	3148	1248	1604	806	2864	3404
1	3434	200	1210	432	2850	3510	2416	2034	3218	2882	3722
2	3496	1210	200	988	2112	2700	2634	2252	3110	2104	2876
3	3366	432	988	200	2636	2080	1304	1088	1456	1538	2350
4	2730	2850	2112	2636	200	2166	2718	2372	2976	1602	2322
5	3148	3510	2700	2080	2166	200	2362	2228	1946	546	1196
6	1248	2416	2634	1304	2718	2362	200	2426	3266	2164	2904
7	1604	2034	2252	1088	2372	2228	2426	200	1302	1294	2030
8	806	3218	3110	1456	2976	1946	3266	1302	200	3296	4102
9	2864	2882	2104	1538	1602	546	2164	1294	3296	200	3784
10	3404	3722	2876	2350	2322	1196	2904	2030	4102	3784	200

** Current Node Information ('*' indicates a root node) **

Node#	Comp#	Weight	Degree	Depth
n[0]	c[0]	2400	0	1
n[1]	c[1]	2400	0	1
n[2]	c[2]	2400	0	1
*n[3]	c[3]	69600	0	0
n[4]	c[4]	2400	0	1
n[5]	c[5]	2400	0	1
n[6]	c[6]	2400	0	1
n[7]	c[7]	2400	0	1
n[8]	c[8]	2400	0	1
n[9]	c[9]	2400	0	1
n[10]	c[10]	2400	0	1

** Current Component Information **

<Comp#	ConRt	Nearest	Link :Cost	CostRt	Tradeoff	Weight	Nodes >
c[0]	nc	c[8]	(0, 8): 806	3366	-2560	2400	n[0]
c[1]	nc	c[3]	(1, 3): 432	432	0	2400	n[1]
c[2]	nc	c[3]	(2, 3): 988	988	0	2400	n[2]
c[3]	*c	c[3]	(3, 3): 200	200	0	69600	n[3]
c[4]	nc	c[9]	(4, 9):1602	2636	-1034	2400	n[4]
c[5]	nc	c[9]	(5, 9): 546	2080	-1534	2400	n[5]
c[6]	nc	c[0]	(6, 0):1248	1304	-56	2400	n[6]
c[7]	nc	c[3]	(7, 3):1088	1088	0	2400	n[7]
c[8]	nc	c[0]	(8, 0): 806	1456	-650	2400	n[8]
c[9]	nc	c[5]	(9, 5): 546	1538	-992	2400	n[9]
c[10]	nc	c[5]	(10, 5):1196	2350	-1154	2400	n[10]

```
Neighbors of Component[0]: <Component#:(link:cost)>
{<c[ 8]:( 0, 8): 806> <c[ 3]:( 0, 3):3366> <c[ 1]:( 0, 1):3434> <c[ 2]:( 0, 2):3496>
 <c[ 4]:( 0, 4):2730> <c[ 5]:( 0, 5):3148> <c[ 6]:( 0, 6):1248> <c[ 7]:( 0, 7):1604>
 <c[ 9]:( 0, 9):2864> <c[10]:( 0,10):3404>}
```

```
Neighbors of Component[1]: <Component#:(link:cost)>
{<c[ 3]:( 1, 3): 432> <c[ 0]:( 1, 0):3434> <c[ 2]:( 1, 2):1210> <c[ 4]:( 1, 4):2850>
 <c[ 5]:( 1, 5):3510> <c[ 6]:( 1, 6):2416> <c[ 7]:( 1, 7):2034> <c[ 8]:( 1, 8):3218>
 <c[ 9]:( 1, 9):2882> <c[10]:( 1,10):3722>}

Neighbors of Component[2]: <Component#:(link:cost)>
{<c[ 3]:( 2, 3): 988> <c[ 0]:( 2, 0):3496> <c[ 1]:( 2, 1):1210> <c[ 4]:( 2, 4):2112>
 <c[ 5]:( 2, 5):2700> <c[ 6]:( 2, 6):2634> <c[ 7]:( 2, 7):2252> <c[ 8]:( 2, 8):3110>
 <c[ 9]:( 2, 9):2104> <c[10]:( 2,10):2876>}

Neighbors of Component[4]: <Component#:(link:cost)>
{<c[ 9]:( 4, 9):1602> <c[ 3]:( 4, 3):2636> <c[ 0]:( 4, 0):2730> <c[ 1]:( 4, 1):2850>
 <c[ 2]:( 4, 2):2112> <c[ 5]:( 4, 5):2166> <c[ 6]:( 4, 6):2718> <c[ 7]:( 4, 7):2372>
 <c[ 8]:( 4, 8):2976> <c[10]:( 4,10):2322>}

Neighbors of Component[5]: <Component#:(link:cost)>
{<c[ 9]:( 5, 9): 546> <c[ 3]:( 5, 3):2080> <c[ 0]:( 5, 0):3148> <c[ 1]:( 5, 1):3510>
 <c[ 2]:( 5, 2):2700> <c[ 4]:( 5, 4):2166> <c[ 6]:( 5, 6):2362> <c[ 7]:( 5, 7):2228>
 <c[ 8]:( 5, 8):1946> <c[10]:( 5,10):1196>}

Neighbors of Component[6]: <Component#:(link:cost)>
{<c[ 0]:( 6, 0):1248> <c[ 3]:( 6, 3):1304> <c[ 1]:( 6, 1):2416> <c[ 2]:( 6, 2):2634>
 <c[ 4]:( 6, 4):2718> <c[ 5]:( 6, 5):2362> <c[ 7]:( 6, 7):2426> <c[ 8]:( 6, 8):3266>
 <c[ 9]:( 6, 9):2164> <c[10]:( 6,10):2904>}

Neighbors of Component[7]: <Component#:(link:cost)>
{<c[ 3]:( 7, 3):1088> <c[ 0]:( 7, 0):1604> <c[ 1]:( 7, 1):2034> <c[ 2]:( 7, 2):2252>
 <c[ 4]:( 7, 4):2372> <c[ 5]:( 7, 5):2228> <c[ 6]:( 7, 6):2426> <c[ 8]:( 7, 8):1302>
 <c[ 9]:( 7, 9):1294> <c[10]:( 7,10):2030>}

Neighbors of Component[8]: <Component#:(link:cost)>
{<c[ 0]:( 8, 0): 806> <c[ 3]:( 8, 3):1456> <c[ 1]:( 8, 1):3218> <c[ 2]:( 8, 2):3110>
 <c[ 4]:( 8, 4):2976> <c[ 5]:( 8, 5):1946> <c[ 6]:( 8, 6):3266> <c[ 7]:( 8, 7):1302>
 <c[ 9]:( 8, 9):3296> <c[10]:( 8,10):4102>}

Neighbors of Component[9]: <Component#:(link:cost)>
{<c[ 5]:( 9, 5): 546> <c[ 3]:( 9, 3):1538> <c[ 0]:( 9, 0):2864> <c[ 1]:( 9, 1):2882>
 <c[ 2]:( 9, 2):2104> <c[ 4]:( 9, 4):1602> <c[ 6]:( 9, 6):2164> <c[ 7]:( 9, 7):1294>
 <c[ 8]:( 9, 8):3296> <c[10]:( 9,10):3784>}

Neighbors of Component[10]: <Component#:(link:cost)>
{<c[ 5]:(10, 5):1196> <c[ 3]:(10, 3):2350> <c[ 0]:(10, 0):3404> <c[ 1]:(10, 1):3722>
 <c[ 2]:(10, 2):2876> <c[ 4]:(10, 4):2322> <c[ 6]:(10, 6):2904> <c[ 7]:(10, 7):2030>
 <c[ 8]:(10, 8):4102> <c[ 9]:(10, 9):3784>}

A = {}

 *** Start EW Algorithm ***

1): The smallest Tradeoff(c[0])=-2560
   Merge Comp(0) and Comp(8) by a link(0,8)

 Link(8,1) is the representative link between c[1] and c[8]
 Update Tradeoff(c[1]) = 432 - 432 = 0
 Link(8,2) is the representative link between c[2] and c[8]
 Update Tradeoff(c[2]) = 988 - 988 = 0
 Link(0,4) is the representative link between c[4] and c[8]
 Update Tradeoff(c[4]) = 1602 - 2636 = -1034
```

```
 Link(8,5) is the representative link between c[5] and c[8]
 Update Tradeoff(c[5]) = 546 - 2080 = -1534
 Link(0,6) is the representative link between c[6] and c[8]
 Update Tradeoff(c[6]) = 1248 - 1304 = -56
 Link(8,7) is the representative link between c[7] and c[8]
 Update Tradeoff(c[7]) = 1088 - 1088 = 0
 Link(0,9) is the representative link between c[9] and c[8]
 Update Tradeoff(c[9]) = 546 - 1538 = -992
 Link(0,10) is the representative link between c[10] and c[8]
 Update Tradeoff(c[10]) = 1196 - 2350 = -1154
 Update Tradeoff(c[8]) = 1248 - 1456 = -208

 ** Current Component Information **
<Comp#|ConRt |Nearest|  Link :Cost|CostRt |Tradeoff|Weight|Nodes >
 -----+------+-------+-----------+-------+--------+------+------
 c[ 1]|  nc  | c[ 3] |( 1, 3): 432|   432 |      0 |  2400| n[1]
 c[ 2]|  nc  | c[ 3] |( 2, 3): 988|   988 |      0 |  2400| n[2]
 c[ 3]|  *c  | c[ 3] |( 3, 3): 200|   200 |      0 | 69600| n[3]
 c[ 4]|  nc  | c[ 9] |( 4, 9):1602|  2636 |  -1034 |  2400| n[4]
 c[ 5]|  nc  | c[ 9] |( 5, 9): 546|  2080 |  -1534 |  2400| n[5]
 c[ 6]|  nc  | c[ 8] |( 0, 6):1248|  1304 |    -56 |  2400| n[6]
 c[ 7]|  nc  | c[ 3] |( 7, 3):1088|  1088 |      0 |  2400| n[7]
 c[ 8]|  nc  | c[ 6] |( 0, 6):1248|  1456 |   -208 |  4800| n[8] n[0]
 c[ 9]|  nc  | c[ 5] |( 9, 5): 546|  1538 |   -992 |  2400| n[9]
 c[10]|  nc  | c[ 5] |(10, 5):1196|  2350 |  -1154 |  2400| n[10]

Neighbors of Component[1]: <Component#:(link:cost)>
{<c[ 3]:( 1, 3): 432> <c[ 2]:( 1, 2):1210> <c[ 4]:( 1, 4):2850> <c[ 5]:( 1, 5):3510>
 <c[ 6]:( 1, 6):2416> <c[ 7]:( 1, 7):2034> <c[ 8]:( 1, 8):3218> <c[ 9]:( 1, 9):2882>
 <c[10]:( 1,10):3722>}

Neighbors of Component[2]: <Component#:(link:cost)>
{<c[ 3]:( 2, 3): 988> <c[ 1]:( 2, 1):1210> <c[ 4]:( 2, 4):2112> <c[ 5]:( 2, 5):2700>
 <c[ 6]:( 2, 6):2634> <c[ 7]:( 2, 7):2252> <c[ 8]:( 2, 8):3110> <c[ 9]:( 2, 9):2104>
 <c[10]:( 2,10):2876>}

Neighbors of Component[4]: <Component#:(link:cost)>
{<c[ 9]:( 4, 9):1602> <c[ 3]:( 4, 3):2636> <c[ 1]:( 4, 1):2850> <c[ 2]:( 4, 2):2112>
 <c[ 5]:( 4, 5):2166> <c[ 6]:( 4, 6):2718> <c[ 7]:( 4, 7):2372> <c[ 8]:( 0, 4):2730>
 <c[10]:( 4,10):2322>}

Neighbors of Component[5]: <Component#:(link:cost)>
{<c[ 9]:( 5, 9): 546> <c[ 3]:( 5, 3):2080> <c[ 1]:( 5, 1):3510> <c[ 2]:( 5, 2):2700>
 <c[ 4]:( 5, 4):2166> <c[ 6]:( 5, 6):2362> <c[ 7]:( 5, 7):2228> <c[ 8]:( 5, 8):1946>
 <c[10]:( 5,10):1196>}

Neighbors of Component[6]: <Component#:(link:cost)>
{<c[ 8]:( 0, 6):1248> <c[ 3]:( 6, 3):1304> <c[ 1]:( 6, 1):2416> <c[ 2]:( 6, 2):2634>
 <c[ 4]:( 6, 4):2718> <c[ 5]:( 6, 5):2362> <c[ 7]:( 6, 7):2426> <c[ 9]:( 6, 9):2164>
 <c[10]:( 6,10):2904>}

Neighbors of Component[7]: <Component#:(link:cost)>
{<c[ 3]:( 7, 3):1088> <c[ 1]:( 7, 1):2034> <c[ 2]:( 7, 2):2252> <c[ 4]:( 7, 4):2372>
 <c[ 5]:( 7, 5):2228> <c[ 6]:( 7, 6):2426> <c[ 8]:( 7, 8):1302> <c[ 9]:( 7, 9):1294>
 <c[10]:( 7,10):2030>}

Neighbors of Component[8]: <Component#:(link:cost)>
{<c[ 6]:( 0, 6):1248> <c[ 3]:( 8, 3):1456> <c[ 1]:( 1, 8):3218> <c[ 2]:( 2, 8):3110>
```

```
 <c[ 4]:( 0, 4):2730> <c[ 5]:( 5, 8):1946> <c[ 7]:( 7, 8):1302> <c[ 9]:( 0, 9):2864>
 <c[10]:( 0,10):3404>}

Neighbors of Component[9]: <Component#:(link:cost)>
{<c[ 5]:( 9, 5): 546> <c[ 3]:( 9, 3):1538> <c[ 1]:( 9, 1):2882> <c[ 2]:( 9, 2):2104>
 <c[ 4]:( 9, 4):1602> <c[ 6]:( 9, 6):2164> <c[ 7]:( 9, 7):1294> <c[ 8]:( 0, 9):2864>
 <c[10]:( 9,10):3784>}

Neighbors of Component[10]: <Component#:(link:cost)>
{<c[ 5]:(10, 5):1196> <c[ 3]:(10, 3):2350> <c[ 1]:(10, 1):3722> <c[ 2]:(10, 2):2876>
 <c[ 4]:(10, 4):2322> <c[ 6]:(10, 6):2904> <c[ 7]:(10, 7):2030> <c[ 8]:( 0,10):3404>
 <c[ 9]:(10, 9):3784>}

A = {(8,0) }

 ** Current Node Information ('*' indicates a root node) **
  Node#|Comp#|Weight|Degree|Depth
 ------+-----+------+------+-----
  n[ 0]|c[ 8]|  2400|     1|    2
  n[ 1]|c[ 1]|  2400|     0|    1
  n[ 2]|c[ 2]|  2400|     0|    1
 *n[ 3]|c[ 3]| 69600|     0|    0
  n[ 4]|c[ 4]|  2400|     0|    1
  n[ 5]|c[ 5]|  2400|     0|    1
  n[ 6]|c[ 6]|  2400|     0|    1
  n[ 7]|c[ 7]|  2400|     0|    1
  n[ 8]|c[ 8]|  2400|     1|    1
  n[ 9]|c[ 9]|  2400|     0|    1
  n[10]|c[10]|  2400|     0|    1

2): The smallest Tradeoff(c[5])=-1534
   Merge Comp(5) and Comp(9) by a link(5,9)

 Link(9,1) is the representative link between c[1] and c[9]
 Update Tradeoff(c[1]) = 432 - 432 = 0
 Link(9,2) is the representative link between c[2] and c[9]
 Update Tradeoff(c[2]) = 988 - 988 = 0
 Link(9,4) is the representative link between c[4] and c[9]
 Update Tradeoff(c[4]) = 1602 - 2636 = -1034
 Link(9,6) is the representative link between c[6] and c[9]
 Update Tradeoff(c[6]) = 1248 - 1304 = -56
 Link(9,7) is the representative link between c[7] and c[9]
 Update Tradeoff(c[7]) = 1088 - 1088 = 0
 Link(5,8) is the representative link between c[8] and c[9]
 Update Tradeoff(c[8]) = 1248 - 1456 = -208
 Link(5,10) is the representative link between c[10] and c[9]
 Update Tradeoff(c[10]) = 1196 - 2350 = -1154
 Update Tradeoff(c[9]) = 1196 - 1538 = -342

 ** Current Component Information **
<Comp#|ConRt |Nearest|  Link :Cost|CostRt |Tradeoff|Weight|Nodes >
 -----+------+-------+------------+-------+--------+------+------
 c[ 1]|  nc  | c[ 3] |( 1, 3): 432|   432 |      0 |  2400| n[1]
 c[ 2]|  nc  | c[ 3] |( 2, 3): 988|   988 |      0 |  2400| n[2]
 c[ 3]|  *c  | c[ 3] |( 3, 3): 200|   200 |      0 | 69600| n[3]
 c[ 4]|  nc  | c[ 9] |( 4, 9):1602|  2636 |  -1034 |  2400| n[4]
 c[ 6]|  nc  | c[ 8] |( 0, 6):1248|  1304 |    -56 |  2400| n[6]
 c[ 7]|  nc  | c[ 3] |( 7, 3):1088|  1088 |      0 |  2400| n[7]
```

```
 c[ 8]|  nc  | c[ 6] |( 0, 6):1248|  1456 |   -208 |  4800| n[8] n[0]
 c[ 9]|  nc  | c[10] |( 5,10):1196|  1538 |   -342 |  4800| n[9] n[5]
 c[10]|  nc  | c[ 9] |( 5,10):1196|  2350 |  -1154 |  2400| n[10]

Neighbors of Component[1]: <Component#:(link:cost)>
{<c[ 3]:( 1, 3): 432> <c[ 2]:( 1, 2):1210> <c[ 4]:( 1, 4):2850> <c[ 6]:( 1, 6):2416>
 <c[ 7]:( 1, 7):2034> <c[ 8]:( 1, 8):3218> <c[ 9]:( 1, 9):2882> <c[10]:( 1,10):3722>}

Neighbors of Component[2]: <Component#:(link:cost)>
{<c[ 3]:( 2, 3): 988> <c[ 1]:( 2, 1):1210> <c[ 4]:( 2, 4):2112> <c[ 6]:( 2, 6):2634>
 <c[ 7]:( 2, 7):2252> <c[ 8]:( 2, 8):3110> <c[ 9]:( 2, 9):2104> <c[10]:( 2,10):2876>}

Neighbors of Component[4]: <Component#:(link:cost)>
{<c[ 9]:( 4, 9):1602> <c[ 3]:( 4, 3):2636> <c[ 1]:( 4, 1):2850> <c[ 2]:( 4, 2):2112>
 <c[ 6]:( 4, 6):2718> <c[ 7]:( 4, 7):2372> <c[ 8]:( 0, 4):2730> <c[10]:( 4,10):2322>}

Neighbors of Component[6]: <Component#:(link:cost)>
{<c[ 8]:( 0, 6):1248> <c[ 3]:( 6, 3):1304> <c[ 1]:( 6, 1):2416> <c[ 2]:( 6, 2):2634>
 <c[ 4]:( 6, 4):2718> <c[ 7]:( 6, 7):2426> <c[ 9]:( 6, 9):2164> <c[10]:( 6,10):2904>}

Neighbors of Component[7]: <Component#:(link:cost)>
{<c[ 3]:( 7, 3):1088> <c[ 1]:( 7, 1):2034> <c[ 2]:( 7, 2):2252> <c[ 4]:( 7, 4):2372>
 <c[ 6]:( 7, 6):2426> <c[ 8]:( 7, 8):1302> <c[ 9]:( 7, 9):1294> <c[10]:( 7,10):2030>}

Neighbors of Component[8]: <Component#:(link:cost)>
{<c[ 6]:( 0, 6):1248> <c[ 3]:( 8, 3):1456> <c[ 1]:( 1, 8):3218> <c[ 2]:( 2, 8):3110>
 <c[ 4]:( 0, 4):2730> <c[ 7]:( 7, 8):1302> <c[ 9]:( 5, 8):1946> <c[10]:( 0,10):3404>}

Neighbors of Component[9]: <Component#:(link:cost)>
{<c[10]:( 5,10):1196> <c[ 3]:( 9, 3):1538> <c[ 1]:( 1, 9):2882> <c[ 2]:( 2, 9):2104>
 <c[ 4]:( 4, 9):1602> <c[ 6]:( 6, 9):2164> <c[ 7]:( 7, 9):1294> <c[ 8]:( 5, 8):1946>}

Neighbors of Component[10]: <Component#:(link:cost)>
{<c[ 9]:( 5,10):1196> <c[ 3]:(10, 3):2350> <c[ 1]:(10, 1):3722> <c[ 2]:(10, 2):2876>
 <c[ 4]:(10, 4):2322> <c[ 6]:(10, 6):2904> <c[ 7]:(10, 7):2030> <c[ 8]:( 0,10):3404>}

A = {(8,0) (9,5) }

 ** Current Node Information ('*' indicates a root node) **
  Node#|Comp#|Weight|Degree|Depth
 ------+-----+------+------+-----
  n[ 0]|c[ 8]|  2400|     1|    2
  n[ 1]|c[ 1]|  2400|     0|    1
  n[ 2]|c[ 2]|  2400|     0|    1
 *n[ 3]|c[ 3]| 69600|     0|    0
  n[ 4]|c[ 4]|  2400|     0|    1
  n[ 5]|c[ 9]|  2400|     1|    2
  n[ 6]|c[ 6]|  2400|     0|    1
  n[ 7]|c[ 7]|  2400|     0|    1
  n[ 8]|c[ 8]|  2400|     1|    1
  n[ 9]|c[ 9]|  2400|     1|    1
  n[10]|c[10]|  2400|     0|    1

3): The smallest Tradeoff(c[10])=-1154
   Merge Comp(10) and Comp(9) by a link(5,10)

 Link(5,1) is the representative link between c[1] and c[9]
 Update Tradeoff(c[1]) = 432 - 432 = 0
```

```
Link(5,2) is the representative link between c[2] and c[9]
Update Tradeoff(c[2]) = 988 - 988 = 0
Link(9,4) is the representative link between c[4] and c[9]
Update Tradeoff(c[4]) = 1602 - 2636 = -1034
Link(5,6) is the representative link between c[6] and c[9]
Update Tradeoff(c[6]) = 1248 - 1304 = -56
Link(5,7) is the representative link between c[7] and c[9]
Update Tradeoff(c[7]) = 1088 - 1088 = 0
Merging c[9] and c[8]) would violate the weight-constraint.
Destory a relation between Comp(9) and Comp(8)
Update Tradeoff(c[8]) = 1248 - 1456 = -208
Update Tradeoff(c[9]) = 1538 - 1538 = 0

 ** Current Component Information **
<Comp#|ConRt |Nearest|  Link :Cost|CostRt |Tradeoff|Weight|Nodes >
 -----+------+-------+-----------+-------+--------+------+------
 c[ 1]|  nc  | c[ 3] |( 1, 3): 432|   432 |      0 |  2400| n[1]
 c[ 2]|  nc  | c[ 3] |( 2, 3): 988|   988 |      0 |  2400| n[2]
 c[ 3]|  *c  | c[ 3] |( 3, 3): 200|   200 |      0 | 69600| n[3]
 c[ 4]|  nc  | c[ 9] |( 4, 9):1602|  2636 |  -1034 |  2400| n[4]
 c[ 6]|  nc  | c[ 8] |( 0, 6):1248|  1304 |    -56 |  2400| n[6]
 c[ 7]|  nc  | c[ 3] |( 7, 3):1088|  1088 |      0 |  2400| n[7]
 c[ 8]|  nc  | c[ 6] |( 0, 6):1248|  1456 |   -208 |  4800| n[8] n[0]
 c[ 9]|  nc  | c[ 3] |( 9, 3):1538|  1538 |      0 |  7200| n[9] n[5] n[10]

Neighbors of Component[1]: <Component#:(link:cost)>
{<c[ 3]:( 1, 3): 432> <c[ 2]:( 1, 2):1210> <c[ 4]:( 1, 4):2850> <c[ 6]:( 1, 6):2416>
 <c[ 7]:( 1, 7):2034> <c[ 8]:( 1, 8):3218> <c[ 9]:( 1, 5):3510>}

Neighbors of Component[2]: <Component#:(link:cost)>
{<c[ 3]:( 2, 3): 988> <c[ 1]:( 2, 1):1210> <c[ 4]:( 2, 4):2112> <c[ 6]:( 2, 6):2634>
 <c[ 7]:( 2, 7):2252> <c[ 8]:( 2, 8):3110> <c[ 9]:( 2, 5):2700>}

Neighbors of Component[4]: <Component#:(link:cost)>
{<c[ 9]:( 4, 9):1602> <c[ 3]:( 4, 3):2636> <c[ 1]:( 4, 1):2850> <c[ 2]:( 4, 2):2112>
 <c[ 6]:( 4, 6):2718> <c[ 7]:( 4, 7):2372> <c[ 8]:( 0, 4):2730>}

Neighbors of Component[6]: <Component#:(link:cost)>
{<c[ 8]:( 0, 6):1248> <c[ 3]:( 6, 3):1304> <c[ 1]:( 6, 1):2416> <c[ 2]:( 6, 2):2634>
 <c[ 4]:( 6, 4):2718> <c[ 7]:( 6, 7):2426> <c[ 9]:( 5, 6):2362>}

Neighbors of Component[7]: <Component#:(link:cost)>
{<c[ 3]:( 7, 3):1088> <c[ 1]:( 7, 1):2034> <c[ 2]:( 7, 2):2252> <c[ 4]:( 7, 4):2372>
 <c[ 6]:( 7, 6):2426> <c[ 8]:( 7, 8):1302> <c[ 9]:( 5, 7):2228>}

Neighbors of Component[8]: <Component#:(link:cost)>
{<c[ 6]:( 0, 6):1248> <c[ 3]:( 8, 3):1456> <c[ 1]:( 1, 8):3218> <c[ 2]:( 2, 8):3110>
 <c[ 4]:( 0, 4):2730> <c[ 7]:( 7, 8):1302>}

Neighbors of Component[9]: <Component#:(link:cost)>
{<c[ 3]:( 9, 3):1538> <c[ 1]:( 1, 5):3510> <c[ 2]:( 2, 5):2700> <c[ 4]:( 4, 9):1602>
 <c[ 6]:( 5, 6):2362> <c[ 7]:( 5, 7):2228>}

A = {(8,0) (9,5) (5,10) }

 ** Current Node Information ('*' indicates a root node) **
  Node#|Comp#|Weight|Degree|Depth
 ------+-----+------+------+-----
```

```
  n[ 0]|c[ 8]|  2400|     1|    2
  n[ 1]|c[ 1]|  2400|     0|    1
  n[ 2]|c[ 2]|  2400|     0|    1
 *n[ 3]|c[ 3]| 69600|     0|    0
  n[ 4]|c[ 4]|  2400|     0|    1
  n[ 5]|c[ 9]|  2400|     2|    2
  n[ 6]|c[ 6]|  2400|     0|    1
  n[ 7]|c[ 7]|  2400|     0|    1
  n[ 8]|c[ 8]|  2400|     1|    1
  n[ 9]|c[ 9]|  2400|     1|    1
  n[10]|c[ 9]|  2400|     1|    3

4): The smallest Tradeoff(c[4])=-1034
   Merge Comp(4) and Comp(9) by a link(4,9)

 Update Tradeoff(c[8]) = 1248 - 1456 = -208
 Merging c[9] and c[1]) would violate the weight-constraint.
 Destory a relation between Comp(9) and Comp(1)
 Update Tradeoff(c[1]) = 432 - 432 = 0
 Merging c[9] and c[2]) would violate the weight-constraint.
 Destory a relation between Comp(9) and Comp(2)
 Update Tradeoff(c[2]) = 988 - 988 = 0
 Merging c[9] and c[6]) would violate the weight-constraint.
 Destory a relation between Comp(9) and Comp(6)
 Update Tradeoff(c[6]) = 1248 - 1304 = -56
 Merging c[9] and c[7]) would violate the weight-constraint.
 Destory a relation between Comp(9) and Comp(7)
 Update Tradeoff(c[7]) = 1088 - 1088 = 0
 Update Tradeoff(c[9]) = 1538 - 1538 = 0

 ** Current Component Information **
<Comp#|ConRt |Nearest|  Link :Cost|CostRt |Tradeoff|Weight|Nodes >
 -----+------+-------+-----------+-------+--------+------+------
 c[ 1]|  nc  | c[ 3] |( 1, 3): 432|   432 |      0 |  2400| n[1]
 c[ 2]|  nc  | c[ 3] |( 2, 3): 988|   988 |      0 |  2400| n[2]
 c[ 3]|  *c  | c[ 3] |( 3, 3): 200|   200 |      0 | 69600| n[3]
 c[ 6]|  nc  | c[ 8] |( 0, 6):1248|  1304 |    -56 |  2400| n[6]
 c[ 7]|  nc  | c[ 3] |( 7, 3):1088|  1088 |      0 |  2400| n[7]
 c[ 8]|  nc  | c[ 6] |( 0, 6):1248|  1456 |   -208 |  4800| n[8] n[0]
 c[ 9]|  nc  | c[ 3] |( 9, 3):1538|  1538 |      0 |  9600| n[9] n[5] n[10] n[4]

Neighbors of Component[1]: <Component#:(link:cost)>
{<c[ 3]:( 1, 3): 432> <c[ 2]:( 1, 2):1210> <c[ 6]:( 1, 6):2416> <c[ 7]:( 1, 7):2034>
 <c[ 8]:( 1, 8):3218>}

Neighbors of Component[2]: <Component#:(link:cost)>
{<c[ 3]:( 2, 3): 988> <c[ 1]:( 2, 1):1210> <c[ 6]:( 2, 6):2634> <c[ 7]:( 2, 7):2252>
 <c[ 8]:( 2, 8):3110>}

Neighbors of Component[6]: <Component#:(link:cost)>
{<c[ 8]:( 0, 6):1248> <c[ 3]:( 6, 3):1304> <c[ 1]:( 6, 1):2416> <c[ 2]:( 6, 2):2634>
 <c[ 7]:( 6, 7):2426>}

Neighbors of Component[7]: <Component#:(link:cost)>
{<c[ 3]:( 7, 3):1088> <c[ 1]:( 7, 1):2034> <c[ 2]:( 7, 2):2252> <c[ 6]:( 7, 6):2426>
 <c[ 8]:( 7, 8):1302>}

Neighbors of Component[8]: <Component#:(link:cost)>
```

```
{<c[ 6]:( 0, 6):1248> <c[ 3]:( 8, 3):1456> <c[ 1]:( 1, 8):3218> <c[ 2]:( 2, 8):3110>
 <c[ 7]:( 7, 8):1302>}

Neighbors of Component[9]: <Component#:(link:cost)>
{<c[ 3]:( 9, 3):1538>}

A = {(8,0) (9,5) (5,10) (9,4) }

 ** Current Node Information ('*' indicates a root node) **
  Node#|Comp#|Weight|Degree|Depth
 ------+-----+------+------+-----
  n[ 0]|c[ 8]|  2400|     1|    2
  n[ 1]|c[ 1]|  2400|     0|    1
  n[ 2]|c[ 2]|  2400|     0|    1
 *n[ 3]|c[ 3]| 69600|     0|    0
  n[ 4]|c[ 9]|  2400|     1|    2
  n[ 5]|c[ 9]|  2400|     2|    2
  n[ 6]|c[ 6]|  2400|     0|    1
  n[ 7]|c[ 7]|  2400|     0|    1
  n[ 8]|c[ 8]|  2400|     1|    1
  n[ 9]|c[ 9]|  2400|     2|    1
  n[10]|c[ 9]|  2400|     1|    3

5): The smallest Tradeoff(c[8])=-208
   Merge Comp(8) and Comp(6) by a link(0,6)

 Link(0,1) is the representative link between c[1] and c[6]
 Update Tradeoff(c[1]) = 432 - 432 = 0
 Link(0,2) is the representative link between c[2] and c[6]
 Update Tradeoff(c[2]) = 988 - 988 = 0
 Link(0,7) is the representative link between c[7] and c[6]
 Update Tradeoff(c[7]) = 1088 - 1088 = 0
 Update Tradeoff(c[6]) = 1304 - 1304 = 0

 ** Current Component Information **
<Comp#|ConRt |Nearest|  Link :Cost|CostRt |Tradeoff|Weight|Nodes >
 -----+------+-------+------------+-------+--------+------+------
 c[ 1]|  nc  | c[ 3] |( 1, 3): 432|   432 |      0 |  2400| n[1]
 c[ 2]|  nc  | c[ 3] |( 2, 3): 988|   988 |      0 |  2400| n[2]
 c[ 3]|  *c  | c[ 3] |( 3, 3): 200|   200 |      0 | 69600| n[3]
 c[ 6]|  nc  | c[ 3] |( 6, 3):1304|  1304 |      0 |  7200| n[6] n[0] n[8]
 c[ 7]|  nc  | c[ 3] |( 7, 3):1088|  1088 |      0 |  2400| n[7]
 c[ 9]|  nc  | c[ 3] |( 9, 3):1538|  1538 |      0 |  9600| n[9] n[5] n[10] n[4]

Neighbors of Component[1]: <Component#:(link:cost)>
{<c[ 3]:( 1, 3): 432> <c[ 2]:( 1, 2):1210> <c[ 6]:( 0, 1):3434> <c[ 7]:( 1, 7):2034>}

Neighbors of Component[2]: <Component#:(link:cost)>
{<c[ 3]:( 2, 3): 988> <c[ 1]:( 2, 1):1210> <c[ 6]:( 0, 2):3496> <c[ 7]:( 2, 7):2252>}

Neighbors of Component[6]: <Component#:(link:cost)>
{<c[ 3]:( 6, 3):1304> <c[ 1]:( 0, 1):3434> <c[ 2]:( 0, 2):3496> <c[ 7]:( 0, 7):1604>}

Neighbors of Component[7]: <Component#:(link:cost)>
{<c[ 3]:( 7, 3):1088> <c[ 1]:( 7, 1):2034> <c[ 2]:( 7, 2):2252> <c[ 6]:( 0, 7):1604>}

Neighbors of Component[9]: <Component#:(link:cost)>
{<c[ 3]:( 9, 3):1538>}
```

```
A = {(8,0) (9,5) (5,10) (9,4) (6,0) }

 ** Current Node Information ('*' indicates a root node) **
  Node#|Comp#|Weight|Degree|Depth
 ------+-----+------+------+-----
  n[ 0]|c[ 6]|  2400|     2|    2
  n[ 1]|c[ 1]|  2400|     0|    1
  n[ 2]|c[ 2]|  2400|     0|    1
 *n[ 3]|c[ 3]| 69600|     0|    0
  n[ 4]|c[ 9]|  2400|     1|    2
  n[ 5]|c[ 9]|  2400|     2|    2
  n[ 6]|c[ 6]|  2400|     1|    1
  n[ 7]|c[ 7]|  2400|     0|    1
  n[ 8]|c[ 6]|  2400|     1|    3
  n[ 9]|c[ 9]|  2400|     2|    1
  n[10]|c[ 9]|  2400|     1|    3

6): The smallest Tradeoff(c[1])=0
   Merge Comp(1) and Comp(3) by a link(1,3)


 ** Current Component Information **
<Comp#|ConRt |Nearest|  Link :Cost|CostRt |Tradeoff|Weight|Nodes >
 -----+------+-------+-----------+-------+--------+------+------
 c[ 1]|   c  | c[ 3] |( 1, 3): 432|   432 |      0 |  2400| n[1]
 c[ 2]|  nc  | c[ 3] |( 2, 3): 988|   988 |      0 |  2400| n[2]
 c[ 3]|  *c  | c[ 3] |( 3, 3): 200|   200 |      0 | 69600| n[3]
 c[ 6]|  nc  | c[ 3] |( 6, 3):1304|  1304 |      0 |  7200| n[6] n[0] n[8]
 c[ 7]|  nc  | c[ 3] |( 7, 3):1088|  1088 |      0 |  2400| n[7]
 c[ 9]|  nc  | c[ 3] |( 9, 3):1538|  1538 |      0 |  9600| n[9] n[5] n[10] n[4]

Neighbors of Component[1]: <Component#:(link:cost)>
{<c[ 3]:( 1, 3): 432> <c[ 2]:( 1, 2):1210> <c[ 6]:( 0, 1):3434> <c[ 7]:( 1, 7):2034>}

Neighbors of Component[2]: <Component#:(link:cost)>
{<c[ 3]:( 2, 3): 988> <c[ 1]:( 2, 1):1210> <c[ 6]:( 0, 2):3496> <c[ 7]:( 2, 7):2252>}

Neighbors of Component[6]: <Component#:(link:cost)>
{<c[ 3]:( 6, 3):1304> <c[ 1]:( 0, 1):3434> <c[ 2]:( 0, 2):3496> <c[ 7]:( 0, 7):1604>}

Neighbors of Component[7]: <Component#:(link:cost)>
{<c[ 3]:( 7, 3):1088> <c[ 1]:( 7, 1):2034> <c[ 2]:( 7, 2):2252> <c[ 6]:( 0, 7):1604>}

Neighbors of Component[9]: <Component#:(link:cost)>
{<c[ 3]:( 9, 3):1538>}

A = {(8,0) (9,5) (5,10) (9,4) (6,0) (1,3) }

 ** Current Node Information ('*' indicates a root node) **
  Node#|Comp#|Weight|Degree|Depth
 ------+-----+------+------+-----
  n[ 0]|c[ 6]|  2400|     2|    2
  n[ 1]|c[ 1]|  2400|     1|    1
  n[ 2]|c[ 2]|  2400|     0|    1
 *n[ 3]|c[ 3]| 69600|     1|    0
  n[ 4]|c[ 9]|  2400|     1|    2
  n[ 5]|c[ 9]|  2400|     2|    2
```

```
  n[ 6]|c[ 6]|  2400|     1|    1
  n[ 7]|c[ 7]|  2400|     0|    1
  n[ 8]|c[ 6]|  2400|     1|    3
  n[ 9]|c[ 9]|  2400|     2|    1
  n[10]|c[ 9]|  2400|     1|    3

7): The smallest Tradeoff(c[2])=0
   Merge Comp(2) and Comp(3) by a link(2,3)


 ** Current Component Information **
<Comp#|ConRt |Nearest|  Link :Cost|CostRt |Tradeoff|Weight|Nodes >
 -----+------+-------+-----------+-------+--------+------+------
 c[ 1]|   c  | c[ 3] |( 1, 3): 432|   432 |      0 |  2400| n[1]
 c[ 2]|   c  | c[ 3] |( 2, 3): 988|   988 |      0 |  2400| n[2]
 c[ 3]|  *c  | c[ 3] |( 3, 3): 200|   200 |      0 | 69600| n[3]
 c[ 6]|  nc  | c[ 3] |( 6, 3):1304|  1304 |      0 |  7200| n[6] n[0] n[8]
 c[ 7]|  nc  | c[ 3] |( 7, 3):1088|  1088 |      0 |  2400| n[7]
 c[ 9]|  nc  | c[ 3] |( 9, 3):1538|  1538 |      0 |  9600| n[9] n[5] n[10] n[4]

Neighbors of Component[1]: <Component#:(link:cost)>
{<c[ 3]:( 1, 3): 432> <c[ 2]:( 1, 2):1210> <c[ 6]:( 0, 1):3434> <c[ 7]:( 1, 7):2034>}

Neighbors of Component[2]: <Component#:(link:cost)>
{<c[ 3]:( 2, 3): 988> <c[ 1]:( 2, 1):1210> <c[ 6]:( 0, 2):3496> <c[ 7]:( 2, 7):2252>}

Neighbors of Component[6]: <Component#:(link:cost)>
{<c[ 3]:( 6, 3):1304> <c[ 1]:( 0, 1):3434> <c[ 2]:( 0, 2):3496> <c[ 7]:( 0, 7):1604>}

Neighbors of Component[7]: <Component#:(link:cost)>
{<c[ 3]:( 7, 3):1088> <c[ 1]:( 7, 1):2034> <c[ 2]:( 7, 2):2252> <c[ 6]:( 0, 7):1604>}

Neighbors of Component[9]: <Component#:(link:cost)>
{<c[ 3]:( 9, 3):1538>}

A = {(8,0) (9,5) (5,10) (9,4) (6,0) (1,3) (2,3) }

 ** Current Node Information ('*' indicates a root node) **
  Node#|Comp#|Weight|Degree|Depth
 ------+-----+------+------+-----
  n[ 0]|c[ 6]|  2400|     2|    2
  n[ 1]|c[ 1]|  2400|     1|    1
  n[ 2]|c[ 2]|  2400|     1|    1
 *n[ 3]|c[ 3]| 69600|     2|    0
  n[ 4]|c[ 9]|  2400|     1|    2
  n[ 5]|c[ 9]|  2400|     2|    2
  n[ 6]|c[ 6]|  2400|     1|    1
  n[ 7]|c[ 7]|  2400|     0|    1
  n[ 8]|c[ 6]|  2400|     1|    3
  n[ 9]|c[ 9]|  2400|     2|    1
  n[10]|c[ 9]|  2400|     1|    3

8): The smallest Tradeoff(c[6])=0
   Merge Comp(6) and Comp(3) by a link(6,3)


 ** Current Component Information **
<Comp#|ConRt |Nearest|  Link :Cost|CostRt |Tradeoff|Weight|Nodes >
```

```
-----+------+-------+-----------+-------+--------+------+------
 c[ 1]|    c  | c[ 3] |( 1, 3): 432|   432 |      0 |  2400| n[1]
 c[ 2]|    c  | c[ 3] |( 2, 3): 988|   988 |      0 |  2400| n[2]
 c[ 3]|   *c  | c[ 3] |( 3, 3): 200|   200 |      0 | 69600| n[3]
 c[ 6]|    c  | c[ 3] |( 6, 3):1304|  1304 |      0 |  7200| n[6] n[0] n[8]
 c[ 7]|   nc  | c[ 3] |( 7, 3):1088|  1088 |      0 |  2400| n[7]
 c[ 9]|   nc  | c[ 3] |( 9, 3):1538|  1538 |      0 |  9600| n[9] n[5] n[10] n[4]

Neighbors of Component[1]: <Component#:(link:cost)>
{<c[ 3]:( 1, 3): 432> <c[ 2]:( 1, 2):1210> <c[ 6]:( 0, 1):3434> <c[ 7]:( 1, 7):2034>}

Neighbors of Component[2]: <Component#:(link:cost)>
{<c[ 3]:( 2, 3): 988> <c[ 1]:( 2, 1):1210> <c[ 6]:( 0, 2):3496> <c[ 7]:( 2, 7):2252>}

Neighbors of Component[6]: <Component#:(link:cost)>
{<c[ 3]:( 6, 3):1304> <c[ 1]:( 0, 1):3434> <c[ 2]:( 0, 2):3496> <c[ 7]:( 0, 7):1604>}

Neighbors of Component[7]: <Component#:(link:cost)>
{<c[ 3]:( 7, 3):1088> <c[ 1]:( 7, 1):2034> <c[ 2]:( 7, 2):2252> <c[ 6]:( 0, 7):1604>}

Neighbors of Component[9]: <Component#:(link:cost)>
{<c[ 3]:( 9, 3):1538>}

A = {(8,0) (9,5) (5,10) (9,4) (6,0) (1,3) (2,3) (6,3) }

 ** Current Node Information ('*' indicates a root node) **
  Node#|Comp#|Weight|Degree|Depth
 ------+-----+------+------+-----
  n[ 0]|c[ 6]|  2400|     2|    2
  n[ 1]|c[ 1]|  2400|     1|    1
  n[ 2]|c[ 2]|  2400|     1|    1
 *n[ 3]|c[ 3]| 69600|     3|    0
  n[ 4]|c[ 9]|  2400|     1|    2
  n[ 5]|c[ 9]|  2400|     2|    2
  n[ 6]|c[ 6]|  2400|     2|    1
  n[ 7]|c[ 7]|  2400|     0|    1
  n[ 8]|c[ 6]|  2400|     1|    3
  n[ 9]|c[ 9]|  2400|     2|    1
  n[10]|c[ 9]|  2400|     1|    3

9): The smallest Tradeoff(c[7])=0
   Merge Comp(7) and Comp(3) by a link(7,3)


 ** Current Component Information **
<Comp#|ConRt |Nearest|  Link :Cost|CostRt |Tradeoff|Weight|Nodes >
 -----+------+-------+-----------+-------+--------+------+------
 c[ 1]|    c  | c[ 3] |( 1, 3): 432|   432 |      0 |  2400| n[1]
 c[ 2]|    c  | c[ 3] |( 2, 3): 988|   988 |      0 |  2400| n[2]
 c[ 3]|   *c  | c[ 3] |( 3, 3): 200|   200 |      0 | 69600| n[3]
 c[ 6]|    c  | c[ 3] |( 6, 3):1304|  1304 |      0 |  7200| n[6] n[0] n[8]
 c[ 7]|    c  | c[ 3] |( 7, 3):1088|  1088 |      0 |  2400| n[7]
 c[ 9]|   nc  | c[ 3] |( 9, 3):1538|  1538 |      0 |  9600| n[9] n[5] n[10] n[4]

Neighbors of Component[1]: <Component#:(link:cost)>
{<c[ 3]:( 1, 3): 432> <c[ 2]:( 1, 2):1210> <c[ 6]:( 0, 1):3434> <c[ 7]:( 1, 7):2034>}

Neighbors of Component[2]: <Component#:(link:cost)>
```

```
{<c[ 3]:( 2, 3): 988> <c[ 1]:( 2, 1):1210> <c[ 6]:( 0, 2):3496> <c[ 7]:( 2, 7):2252>}

Neighbors of Component[6]: <Component#:(link:cost)>
{<c[ 3]:( 6, 3):1304> <c[ 1]:( 0, 1):3434> <c[ 2]:( 0, 2):3496> <c[ 7]:( 0, 7):1604>}

Neighbors of Component[7]: <Component#:(link:cost)>
{<c[ 3]:( 7, 3):1088> <c[ 1]:( 7, 1):2034> <c[ 2]:( 7, 2):2252> <c[ 6]:( 0, 7):1604>}

Neighbors of Component[9]: <Component#:(link:cost)>
{<c[ 3]:( 9, 3):1538>}

A = {(8,0) (9,5) (5,10) (9,4) (6,0) (1,3) (2,3) (6,3) (7,3) }

 ** Current Node Information ('*' indicates a root node) **
  Node#|Comp#|Weight|Degree|Depth
 ------+-----+------+------+-----
  n[ 0]|c[ 6]|  2400|     2|    2
  n[ 1]|c[ 1]|  2400|     1|    1
  n[ 2]|c[ 2]|  2400|     1|    1
 *n[ 3]|c[ 3]| 69600|     4|    0
  n[ 4]|c[ 9]|  2400|     1|    2
  n[ 5]|c[ 9]|  2400|     2|    2
  n[ 6]|c[ 6]|  2400|     2|    1
  n[ 7]|c[ 7]|  2400|     1|    1
  n[ 8]|c[ 6]|  2400|     1|    3
  n[ 9]|c[ 9]|  2400|     2|    1
  n[10]|c[ 9]|  2400|     1|    3

10): The smallest Tradeoff(c[9])=0
   Merge Comp(9) and Comp(3) by a link(9,3)


 ** Current Component Information **
<Comp#|ConRt |Nearest|  Link :Cost|CostRt |Tradeoff|Weight|Nodes >
 -----+------+-------+-----------+-------+--------+------+------
 c[ 1]|   c  | c[ 3] |( 1, 3): 432|   432 |      0 |  2400| n[1]
 c[ 2]|   c  | c[ 3] |( 2, 3): 988|   988 |      0 |  2400| n[2]
 c[ 3]|  *c  | c[ 3] |( 3, 3): 200|   200 |      0 | 69600| n[3]
 c[ 6]|   c  | c[ 3] |( 6, 3):1304|  1304 |      0 |  7200| n[6] n[0] n[8]
 c[ 7]|   c  | c[ 3] |( 7, 3):1088|  1088 |      0 |  2400| n[7]
 c[ 9]|   c  | c[ 3] |( 9, 3):1538|  1538 |      0 |  9600| n[9] n[5] n[10] n[4]

Neighbors of Component[1]: <Component#:(link:cost)>
{<c[ 3]:( 1, 3): 432> <c[ 2]:( 1, 2):1210> <c[ 6]:( 0, 1):3434> <c[ 7]:( 1, 7):2034>}

Neighbors of Component[2]: <Component#:(link:cost)>
{<c[ 3]:( 2, 3): 988> <c[ 1]:( 2, 1):1210> <c[ 6]:( 0, 2):3496> <c[ 7]:( 2, 7):2252>}

Neighbors of Component[6]: <Component#:(link:cost)>
{<c[ 3]:( 6, 3):1304> <c[ 1]:( 0, 1):3434> <c[ 2]:( 0, 2):3496> <c[ 7]:( 0, 7):1604>}

Neighbors of Component[7]: <Component#:(link:cost)>
{<c[ 3]:( 7, 3):1088> <c[ 1]:( 7, 1):2034> <c[ 2]:( 7, 2):2252> <c[ 6]:( 0, 7):1604>}

Neighbors of Component[9]: <Component#:(link:cost)>
{<c[ 3]:( 9, 3):1538>}

A = {(8,0) (9,5) (5,10) (9,4) (6,0) (1,3) (2,3) (6,3) (7,3) (9,3) }
```

```
** Current Node Information ('*' indicates a root node) **
 Node#|Comp#|Weight|Degree|Depth
------+-----+------+------+-----
 n[ 0]|c[ 6]|  2400|     2|    2
 n[ 1]|c[ 1]|  2400|     1|    1
 n[ 2]|c[ 2]|  2400|     1|    1
*n[ 3]|c[ 3]| 69600|     5|    0
 n[ 4]|c[ 9]|  2400|     1|    2
 n[ 5]|c[ 9]|  2400|     2|    2
 n[ 6]|c[ 6]|  2400|     2|    1
 n[ 7]|c[ 7]|  2400|     1|    1
 n[ 8]|c[ 6]|  2400|     1|    3
 n[ 9]|c[ 9]|  2400|     3|    1
 n[10]|c[ 9]|  2400|     1|    3


*** Complete EW Algorithm ***

The accepted links and the cost are following

   Link   | Cost
  --------+------
  (8,0)   | 806
  (9,5)   | 546
  (5,10)  | 1196
  (9,4)   | 1602
  (6,0)   | 1248
  (1,3)   | 432
  (2,3)   | 988
  (6,3)   | 1304
  (7,3)   | 1088
  (9,3)   | 1538
----------------------------
Total Network Cost = 10748
```

VITA

Shinji Fujino

Candidate for the Degree of

Master of Science

Thesis: IMPROVING THE ESAU-WILLIAMS ALGORITHM FOR DESIGNING LOCAL ACCESS NETWORKS

Major Field: Computer Science

Biographical:

Personal Data: Born in Aomori, Japan, On December 20, 1974, son of Ryuuichi and Sachie

Education: Graduated from Hachinohe Nishi High School, Aomori, Japan in March 1993; received Bachelor of Science degree in Computer Science and Engineering from University of Aizu, Fukushima, Japan in March 1997. Completed the requirements for the Master of Science degree at the Computer Science Department at Oklahoma State University in May 2000.

Experience: Employed as a graduate assistant developing online applications as part of the College of Arts and Sciences online learning research program, Oklahoma State University from January to December in 1999.