International Journal of Industrial Engineering Computations 3 (2012) 843–858

Contents lists available at GrowingScience

International Journal of Industrial Engineering Computations

homepage: www.GrowingScience.com/ijiec

Introducing mass balancing theorem for network flow maximization

Ziauddin Ursani*

Department of Mechanical Engineering and Mathematical Sciences Faculty of Technology, Design and Environment Oxford Brookes University UK

ARTICLEINFO

Article history:
Received 27 March 2012
Received in revised format
31 May 2012
Accepted June 8 2012
Available online
9 June 2012

Keywords:
Network flow maximization
Mass balance theorem
Multi-commodity flow
Flow dissipation

ABSTRACT

Maximization of flow through the network is required in many practical applications such as water supply flow networks, Oil and Gas flow networks, and transportation networks etc. In this paper a new theorem is presented that has direct application on maximization of flow through the network. This theorem suggests that the maximization of network flow can be achieved by visiting only unbalanced nodes rather than the whole network. Therefore based on this theorem a method is developed that maximizes flow thorough the network by visiting only unbalanced nodes. Hence this method can achieve solution in a sub-linear time where network has fewer unbalanced nodes. However this method has worst case complexity of order $O(m^2\text{-}m)$, where m is the number of edges. Furthermore it is shown that this theorem has also potential to make optimization an easier task in a multi-commodity flow environment.

1. Introduction

The network flow maximization problem has many practical applications in road, railway, water supply, oil & Gas, and other physical networks. The problem consists of maximizing the flow from a source to a sink in a network consisting of a number of nodes connected by a number of directed edges. This problem is well studied and a large volume of literature exists on this problem since the last fifty years. The first major breakthrough came in a form of max-flow min-cut theorem (Danzig & Fulkerson, 1956; Ford & Fulkerson, 1956; Elias et al., 1956). According to this theorem maximum flow through the network is equal to its minimum cut. Minimum cut of the network is the cut of the minimum capacity that divides the network into two parts such that no flow could travel from the source to the sink. On the basis of this theorem augmenting path algorithms for the flow maximization have been proposed in (Ford & Fulkerson, 1956, Elias et al., 1956). This algorithm is iterative. In each iteration this algorithm finds a non-zero residual capacity path from source to the sink and augments flow on that path. It continues to augment flow on such paths until it fails to find any non-zero residual capacity path in the network. Since then, a number of Augmenting Path Algorithms have appeared in the literature. These algorithms were invented to improve worst case speed of the augmenting path algorithm by proposing some criterion to adopt some sequence in choosing paths for flow augmentations. Dinic (1970) and Edmonds and Karp (1972) proposed shortest augmenting path

* Corresponding author.
E-mail: ziaursani@yahoo.com (Z. Ursani)

doi: 10.5267/j.ijiec.2012.05.006

algorithm where shortest paths were chosen first for flow augmentations. This algorithm was further improved by Orlin and Ahuja (1987) by using distance labels (Goldberg 1985) to find the shortest paths quickly. The capacity scaling algorithm was introduced by Gabow (1985). This algorithm first selects the paths with higher residual capacity to augment the flow on them. Other augmenting path algorithms can be found in Tarjan (1986), Ahuja and Orlin (1989, 1991), and Ahuja *et al.* (1988). The augmenting path algorithm is feasible flow algorithm where feasibility is maintained all the time. The major drawback of augmenting path algorithm is that in each augmentation we cannot augment the flow more than residual capacity of the path. This slows down its speed considerably. To overcome this drawback concept of pre-flow was developed to increase the algorithm speed. In pre-flows excess flow at nodes is allowed. According to this concept instead of choosing a complete path, flow can be pushed to individual edges that can be pushed further to other edges in the network. In this way flow on all the paths of the network is augmented in parallel. This class of algorithms is called pre-flow push algorithms. The concept of pre-flows was first suggested in (Boldyreff, 1955). However formal conceptual proposal of pre-flows first appeared in (Karzanov, 1974). To make this algorithm faster many improvements have been proposed in this algorithm based on selection of nodes for 'push' operation. Highest label pre-flow push algorithm was proposed by Goldberg and Tarjan (1986), where selection of node for push operation was based on its distance from the sink. More the distance more likely the node is selected. Excess scaling algorithm was proposed by Ahuja and Orlin (1989), where selection of node is based on amount of excess flow. Node having more excess flow is more likely to be selected. Other pre-flow push algorithms can be found in Cherkasky (1977), Malhotra et al., (1978), Galil (1980), Tarjan (1984), Goldberg and Tarjan (1988). In addition to these two major classes of algorithms, some other algorithms also appeared, such as a recent algorithm based on pseudo-flows (Hochbaum, 2008). In pseudo-flows both excess and deficit flows at nodes is allowed. The pseudo-flow algorithm has roots in certificate of optimality (Lerchs and Grossmann 1965), where link was established in Hochbaum (2001). The pseudo-flow algorithm first solves maximum blocking cut problem (Radzik, 1993), then quickly establishes maximum flow. Further literature about pseudo-flows can be seen in Hochbaum (1997, 2003, 2007), and Chandran and Hochbaum (2009).

All the above algorithms only see the global view of the network, i.e., they visit each and every node and edge of the network to achieve maximum flow. However this problem can be solved with only local manipulations while still achieving the global optimum, *i.e.*, it is not necessary to visit all nodes and edges of the network or it is not necessary to find all paths from source to sink. Flow can be maximized with only local interactions between the flows of neighbouring nodes. The objective of maximizing the flow through local interactions between the neighbouring nodes without traversing all possible paths from source to sink can be achieved if maximum flow problem is translated into the mass balancing problem. We define mass balancing problem as a network problem in which nodes have unbalanced flows, where sum of flows coming into the nodes is not equal to sum of flows getting out of the nodes. If mass balancing is performed between the nodes locally, the resultant network will represent solution of the maximum flow problem. A theorem is devised to make this point. It should be noted that other methods like pseudo-flows (Hochbaum, 2008) and draining algorithm (Dong et al., 2009) are also somewhat close to this concept. However proposed method is very simple and doesn't require any complex data structures (Hochbaum, 2008) or network modifications (Dong et al., 2009). The proposed method is also extendable to multi-commodity problem. Furthermore the idea of equivalence between flow maximization and flow balancing problems is innovative and is obtained by discovering a new property of the network that it acts as a balance between its two physical parts on the either side of the minimum cut. This claim is supported by a Mass Balancing Theorem that is first time presented in this paper. The resultant method works very fast on datasets with fewer unbalanced nodes.

This paper is structured as follows. In section 2, Mass Balancing Theorem is presented which shows that solving the mass balancing problem yields the solution of maximum flow problem. Section 2 has 3 subsections. In section 2.1 mass balancing method is explained and it is shown that this method always arrives at optimal solution by discovering a network balance property in section 2.2. Working examples of this method are presented in section 2.3. In section 3, Complexity analysis for this method is

presented. In section 4, this method is extended to multi-commodity flow problem. Finally some conclusions are made and future work is discussed.

2. Mass Balance Theorem

Consider a flow network consisting of a single source '*s*', and a single sink '*t*', a number of nodes (including *s* & *t*) *V* and number of edges *E*. Each edge e_{ij} connecting any two nodes *i* and *j* is directed from node *i* to node *j* and is characterized by the capacity c_{ij}. Each transmission node must have at least one incoming edge and one out going edge. Our objective is to maximize the flow *Q* through the network, considering unlimited input from the source.

There are some obvious physical constraints on maximum quantity of *Q*, i.e., it cannot exceed sum of capacity of edges directly connected to source or sum of capacity of edges directly connected to sink. These constraints are mathematically represented below in relation (1) and relation (2).

$$Q_0 \leq \sum_{k \in \vec{V}_s} C_{sk}, \tag{1}$$

$$Q_0 \leq \sum_{k \in \overleftarrow{V_t}} c_{kt,} \tag{2}$$

where

$\vec{V_j}$ = Set of nodes immediately succeeding source node *s*
$\overleftarrow{V_j}$ = Set of nodes immediately preceding sink node *t*
Q_0= Maximum possible flow through the network

From 1 and 2 we can deduce

$$Q_0 \leq min\left(\sum_{k \in \vec{V_s}} c_{sk}, \sum_{k \in \overleftarrow{V_t}} c_{kt}\right). \tag{3}$$

Eq. (3) means that maximum flow through the network can at most be equal to the lesser value of the two quantities i.e., sum of capacity of edges directly connected to source and sum of capacity of edges directly connected to sink. If the network bears the property having all balanced nodes i.e.,

$$\overset{\forall}{j} \in N \sum_{i \in \overleftarrow{V_j}} c_{ij} = \sum_{k \in \vec{V_j}} c_{jk}, \tag{4}$$

where *N* = Number of transmission nodes i.e., nodes excluding source and sink

then the relationship 3 will become:

$$Q_0 = \sum_{k \in \vec{V_s}} c_{sk} = \sum_{k \in \overleftarrow{V_t}} c_{kt}. \tag{5}$$

The equation 5 implies that if a network without unbalanced nodes (nodes which do not obey Eq. (4)) is fully saturated, the maximum flow can be determined immediately from the sum of the capacities of the outgoing edges at the source or from the sum of the capacities of the edges going into the sink.

However fully saturated network with unbalanced nodes, violates the law of flow conservation at nodes. In such a case it is necessary to get rid of excess and deficit flows at nodes. Let Q_d be the sum of all excess and deficit flows i.e.,

$$Q_d = Q_s + |Q_t|, \tag{6}$$

where

Q_s = Sum of all excess flows at nodes
Q_t = Sum of all deficit flows at nodes

Q_d can be dissipated in 3 different ways.

1. By dumping part of excess flow to the source node (q_s)
2. By dumping part of deficit flow to the sink node (q_t)
3. By cancelling rest of excess $(Q_s - q_s)$ and deficit $(Q_t - q_t)$flows i.e., mass balancing Δq,that can be computed as;

$$\Delta q = (Q_s - q_s) + |(Q_t - q_t)|. \tag{7}$$

Therefore total flow to be dissipated from the network is given by:

$$Q_d = q_s + |q_t| + \Delta q. \tag{8}$$

Therefore after dumping the flow to the source and sink Eq. (5) becomes

$$Q = \sum_{k \in \overrightarrow{V_s}} c_{sk} - q_s = \sum_{k \in \overrightarrow{V_t}} c_{kt} - |q_t|. \tag{9}$$

This shows after dumping excess flow to source and deficit flow to sink the representative flow through the network may not be the maximum. From the above equation it is clear that capacities of edges are constants. However the two variables q_s and q_t can be minimized to maximize Q. Therefore equation 9 can be modified as follows:

$$max(Q) = Q_0 = \sum_{k \in \overrightarrow{V_s}} c_{sk} - min(q_s) = \sum_{k \in \overrightarrow{V_t}} c_{kt} - min(|q_t|) \tag{10}$$

Furthermore Eq. (8) can be rewritten as follows:

$$\Delta q = Q_d - (q_s + |q_t|) \tag{11}$$

From the Eq. (11), it can be seen that if q_s and q_t are minimized, then Δq is maximized because Q_d is a constant and is independent of q_s and q_t. Therefore, Eq. (9) can be rewritten as:

$$max(\Delta q) = Q_d - min(q_s + |q_t|). \tag{12}$$

By comparing Eq. (10) and Eq. (12) we get

$$max(\Delta q) \approx max(Q) \tag{13}$$

Relationship 13 is very fundamental relationship which shows that if cancelation of excess and deficit flows with each other is maximized or in other words mass balancing is maximized the flow in the network is also maximized, provided rest of excess and deficit flows are dissipated to source and sink respectively.

It is very beneficial in terms of complexity to translate the objective function in this way, because for the new objective localized optimization method can be employed that manipulates excess and deficit flows locally at adjacent nodes, without looking at global view of the network. This is further explained in the next section.

2.1 Mass balance method

To explain this method four concepts are introduced here i.e., dissipative flow of node, dissipative flow of edge, dissipative flow of path, and flow dissipation on the path.

The dissipative flow of node *j* is given by;

$$d_j = \sum_{i \in \overleftarrow{V_j}} q_{ij} - \sum_{k \in \overrightarrow{V_j}} q_{jk}, \tag{14}$$

where q_{ij}= flow in the edge e_{ij}

Eq. (14) shows that dissipative flow of the node is the amount of excess or deficit flow at that node. Dissipative flow of source and sink is considered positive and negative infinity respectively.

The dissipative flow of edge e_{ij} is given by;

$$\begin{cases} d_{ij} = q_{ij} \\ d_{ji} = c_{ij} - q_{ij} \end{cases} \tag{15}$$

Eq. (15) means that dissipative flow of edge is different in its forward and backward direction. In forward direction it is equal to the flow present in the edge, while in backward direction it is equal to residual capacity of the edge. The dissipative flow of path P_{if} from node *i* to node *f* is given by;

$$d_{i \to f} = \mp min\left(\Delta, |\mp d_i|, \left|d_f\right|\right) \tag{16}$$

where

Δ= Minimum dissipative flow among all edges included in the path from initial node *i* to final node *f*

Equation 16 means dissipative flow of path is minimum of dissipative flows of initial and final nodes and all edges present in that path. Furthermore it is taken as negative if initial node is negative node else it is taken as positive.

Flow dissipation on the path from node *i* to node *f* means applying following operation on each edge of the path.

$$q_{ij}^{'} = q_{ij} \mp d_{i \to f} \tag{17}$$

Where positive sign stands for forward edge and negative sign stands for backward edge.

*This method can simply be described in following steps.

1. The Algorithm starts with the fully saturated network not observing the flow conservation law at nodes.
2. Each node *n* is assigned value equal to d_n as shown in equation 14.
3. The algorithm scans the node list to locate first node *i* with $d_i < 0$.
4. The algorithm starts search from node *i* through the network and establishes a path P_{if} as soon as it strikes the first node *f* with $d_f > 0$. Please note that nodes *i* and *f* must not be sink and source nodes. The first and last edge of this path must be forward edge and $d_{i \to f}$(eq. 16) must not be zero. Please also note that this path cannot contain another node with positive dissipative flow in its middle because path is established at first encounter with such a node.
5. The algorithm dissipates the flow along this path according to equation 17 and updates the dissipative flow of first and last nodes of the path accordingly.
6. The algorithm repeats the steps 3 to 5 iteratively until it fails to find a feasible P_{if} path. Feasible path means path with non zero dissipative flow.
7. The algorithm repeats step 3.
8. The algorithm finds the path from node *i* to sink node *f* through the network. All the edges of this path must be forward edges and $d_{i \to f}$must not be zero.
9. The algorithm repeats step 5.
10. The algorithm repeats the steps 7 to 9 iteratively until there remains no node *i* with $d_i < 0$.

---

* It is gratefully acknowledged that Prof. Todinov in his review of this paper (Todinov 2011A) helped in description of this method that was later for its correct representation expanded into 14 step description by author of this paper. Later Todinov (2011B) extended this method to Repairable Flow Networks.

11. The algorithm scans the node list again to find the node i with $d_i > 0$.
12. The algorithm finds the path from node i to source node through the network. All the edges of this path must be backward edges and $d_{i \to f}$ must not be zero.
13. The algorithm repeats step 5.
14. The algorithm repeats the steps 11 to 13 iteratively until there remains no node i with $d_i > 0$.

2.2 Proof of Optimality

It can be shown that this method always gives the optimal solution. According to law of flow conservation we have;

$$\sum_{k \in \overrightarrow{V_s}} c_{sk} - \sum_{k \in \overleftarrow{V_t}} c_{kt} = \overset{\forall}{i \in N} \sum d_i. \tag{18}$$

Eq. (18) means difference between sum of capacities of edges directly connected to source and sink must be equal to dissipative flow of all transmission nodes in the network. If the network is partitioned into two parts through the minimum cut, the result will be two disconnected parts of network one without sink (Part A) and another without source (Part B). Take part A of the network and connect all the edges of minimum cut directly to the sink. Hence in this modified network-A, all the edges connected to the sink represent the minimum cut C_0, therefore

$$C_0 = \sum_{k \in \overleftarrow{V_t}} c_{kt}. \tag{19}$$

By comparing Eq. (18) and Eq. (19) we get

$$C_0 = \sum_{k \in \overrightarrow{V_s}} c_{sk} - \overset{\forall}{a \in N_A} \sum d_a. \tag{20}$$

Similarly take part B of the network and connect all the edges of the minimum cut directly with the source. Hence in this modified network-B, all the edges connected to the sink represent the minimum cut C_0, therefore

$$C_0 = \sum_{k \in \overrightarrow{V_s}} c_{sk}. \tag{21}$$

By comparing Eq. (18) and Eq. (21) we get

$$C_0 = \sum_{k \in \overleftarrow{V_t}} c_{kt} + \overset{\forall}{b \in N_B} \sum d_b \tag{22}$$

By comparing Eq. (20) and Eq. (22) we get

$$\sum_{k \in \overrightarrow{V_s}} c_{sk} - \overset{\forall}{a \in N_A} \sum d_a = C_0 = \sum_{k \in \overleftarrow{V_t}} c_{kt} + \overset{\forall}{b \in N_B} \sum d_b \tag{23}$$

Eq. (23) is fundamental which establishes that value of the minimum cut can be calculated from both parts of the network. This is special property that shows network is a balance between two parts of either side of the minimum cut. This property can be used to establish that proposed method will always arrive at optimal solution. Since the proposed procedure starts with the saturated network hence initially minimum cut is saturated. Therefore at this stage minimum cut represents the maximum flow, therefore equation 23 becomes:

$$\sum_{k \in \overrightarrow{V_s}} c_{sk} - \overset{\forall}{a \in N_A} \sum d_a = Q_0 = \sum_{k \in \overleftarrow{V_t}} c_{kt} + \overset{\forall}{b \in N_B} \sum d_b \tag{24}$$

At this stage, Eq. (24) represents the system of fully saturated network that doesn't obey law of flow conservation at nodes. If minimum cut of above saturated network remains saturated after application of the proposed procedure then optimal solution is achieved. The proposed procedure has three sequential sub-procedures i.e., mass balancing (Steps 3-6), dumping to the sink (Steps 7-10) and then

returning to the source (Steps 11-14). It is mathematically shown below that after application of each of these sub-procedures the minimum cut remains saturated.

If mass balancing is done entirely within part A (Eq. (20)) or part B (Eq. (22)) of the network, this will not remove any flow from the minimum cut and hence it will remain saturated. However if the flow is dissipated across the two parts of the network (A & B) then flow may be removed from the minimum cut. For example, if we dissipate the flow by amount Δ along the path P_{ab} while deficit node *a* present in part A and excess node *b* present in part B, then this will remove amount Δ from the minimum cut and equation 24 will become

$$Q_0 - \Delta = \sum_{k \in \overrightarrow{V_s}} c_{sk} - \left(\overset{\forall}{a \in N_A} \sum d_a + \Delta \right) = \sum_{k \in \overrightarrow{V_t}} c_{kt} + \overset{\forall}{b \in N_B} \sum d_b - \Delta \tag{25}$$

Above equation shows that this operation leaves extra amount of flow +Δ in part A and same amount of flow -Δ in part B of the network. To dissipate this amount a backward path from part B to part A will ultimately be found that will add back same amount Δ in the minimum cut, thus Eq. (25) will return to same status of equation 24 that fundamentally represent the balance of the network. This shows that any dissipation operation across two parts of the network will be matched by the equal operation in the opposite direction to satisfy the Eq. (24). Thus the network acts as a seesaw with support at the minimum cut and two pans A and B on either side of it. If the load is transfered from one pan to another, balance of seesaw is disturbed which is represented through Eq. (24). The load is then transferred back to satisfy this equation. Thus minimum cut will remain saturated after complete mass balancing (Steps 3-6) which is the key condition for the maximum flow. However it is not sufficient condition. The solution may not be optimal after dumping the flow to the sink (Steps 7-10) and returning the flow to the source (Steps 11-14). Because solution may become sub-optimal if any of above two actions dissipate the flow from the minimum cut. However it can be shown here this is not possible.

The above two actions can only dissipate flow from minimum cut if there remains negative dissipative flow in part A of the network or positive dissipative flow in part B of the network. In such a case flow will be again dissipated through the minimum cut to get rid of infeasible flows thus ending up in sub-optimal solution.

From Eq. (20) and Eq. (22) it can be shown that this cannot happen. If after the mass balancing there is a node *i* having $d_i < 0$ then it will always be in the part B of the network and node *j* having $d_j > 0$ will always be in the part A of the network.

Since minimum cut cannot be greater than the sum of capacities of edges directly connected to source i.e.,

$$C_0 \nprec \sum_{k \in \overrightarrow{V_s}} c_{sk} \tag{26}$$

Therefore by comparing Eq. (20) and relation 26, we get

$$\overset{\forall}{a \in N_A} \sum d_a \nprec 0 \tag{27}$$

From the above equation it is clear that total of all dissipative flows of nodes in part A of the network cannot be less than 0 hence any nodes left after the mass balancing will always have $d_j > 0$. Similarly minimum cut cannot be greater than the sum of capacities of edges directly connected to sink i.e.,

$$C_0 \nprec \sum_{k \in \overrightarrow{V_t}} c_{kt} \tag{28}$$

Therefore, by comparing Eq. (22) and relation 28, we get

$$\forall_{b \in N_B} \sum d_b \not\succ 0 \tag{29}$$

From the above equation it is clear that total of all dissipative flows of nodes in part B of the network cannot be greater than 0 hence any nodes left after the mass balancing will always have $d_i < 0$. Therefore even after dumping flow to the sink and returning the flow to the source the minimum cut is not disturbed and remains saturated. Thus final solution represents the optimal solution, as there will not be any *st*-path with non-zero residual capacity.

2.3 Working Examples

Now the working of the proposed method is shown below through some examples. Consider the network in Fig. 1a, which is taken from (Ahuja, et al. 1988).

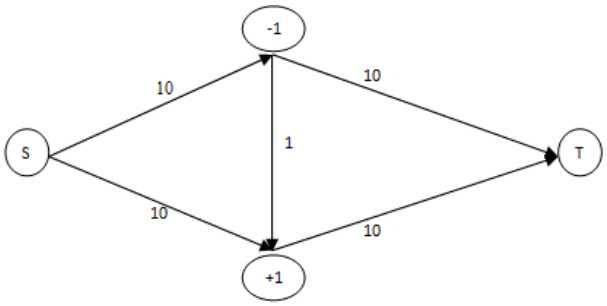

Fig. 1a. A simple network showing nodes with dissipative flows

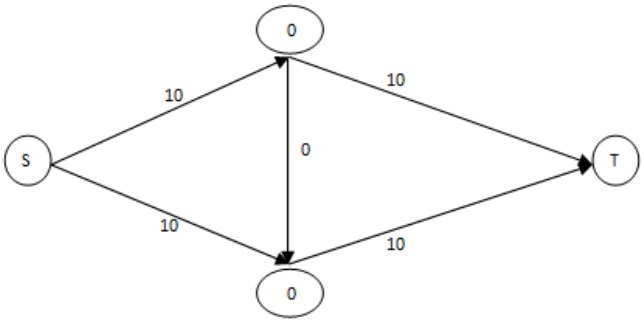

Fig.1b. A maximum flow solution through mass balancing

Fig. 1a shows a simple network with only two nodes in addition to source and sink. The two nodes have dissipative flows +1 and -1. Since the node with negative dissipative flow is directly linked to node with positive dissipative flow hence flow can be dissipated in their connecting edge (steps 3-5). Fig.1b shows the resultant solution of the problem. In Fig. 1b it can be seen that solution of maximum is actually achieved through a simple mass balancing between two nodes. Conventional augmenting path algorithm would have required more iterations and sequential augmentations of paths from the source to the sink. However this is a simplified example. Let us look at a more complex example.

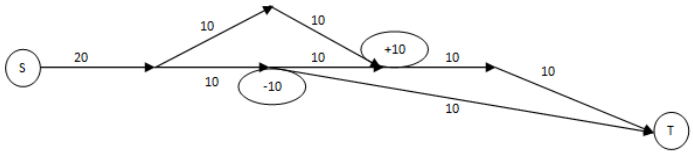

Fig. 2a. A more complex network requiring route selection

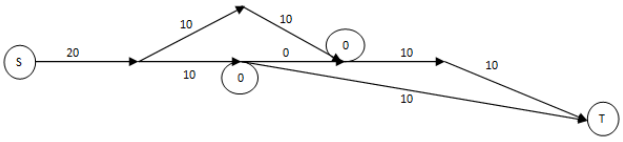

Fig. 2b. A maximum flow solution for the network in Fig.2a

In Fig. 2a, it can be seen that there are two unbalanced nodes. Any other existing method would require a full scanning of the network to arrive at the optimal solution. However the mass balance method just dissipates the flow in the edge connecting the two unbalanced nodes (steps 3-5) and arrives directly at the optimal solution as shown in Fig. 2b. Now let us consider another example which has added complexity specifically for the mass balancing method.

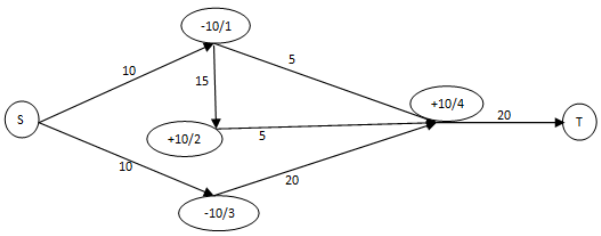

Fig. 3a. A network with added complexity

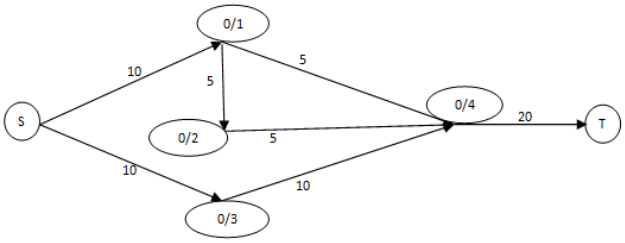

Fig. 3b. Optimal solution of the-a network with added complexity

In Fig.3a, there are 4 nodes numbered 1, 2, 3, and 4 and all the 4 nodes are unbalanced nodes. If fortunately enough the flow is dissipated between right nodes i.e., node-1 with node-2 (steps 3-5) and node-3 with node-4 (steps 3-5) then the optimal solution can be achieved directly as shown in Figure-3b. In case, wrong nodes are chosen to dissipate the flow, then it may need some undirected moves to arrive at the same optimal solution of Fig. 3b. For example, if node-1 and node-4 are chosen to dissipate the flow in first iteration (steps 3-5), then node-1 and node-2 are chosen to dissipate the flow in next iteration (steps 3-5) and finally node-3 and node-4 are chosen to dissipate the flow in last iteration (steps 3-5). The resultant network after these 3 actions in sequence is given in Fig. 3c.

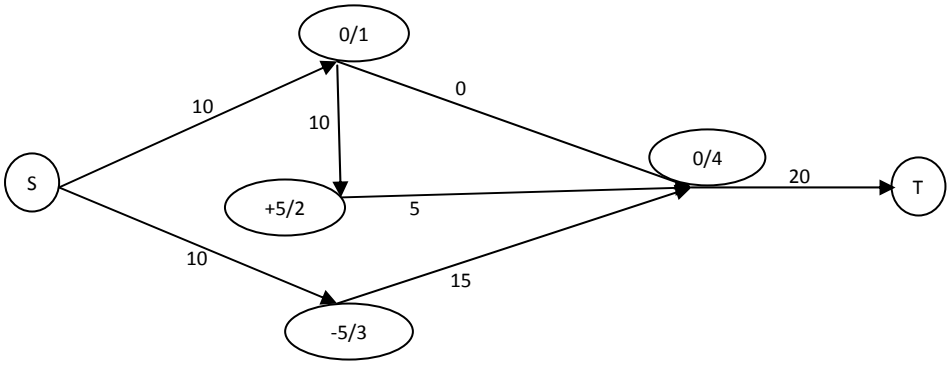

Fig. 3c. Sub-optimal solution of a network with added complexity

It can be seen in Fig. 3c, that sub-optimal solution has been achieved, with still two unbalanced nodes 2 and 3 having no directed link with each other. To solve this problem a path having backward edges i.e., 3→4←1→2, can be established. This path fullfils the conditions that its initial and final edges are forward edges, initial node has $d_i < 0$ and final node has $d_f > 0$ and also dissipative flow of path $d_i \rightarrow_f = -5 \neq 0$. If flow dissipation operation (Eq. (17)) on this path is applied, the optimal solution of Figure-3b is obtained. This experiment was designed specifically to show that it doesn't matter if initially wrong paths are established to dissipate the flow there is always a way to arrive at optimal solution. Now to give a full grasp of the theorem, described in section-2, example of a representative network is presentd, in which all the dissipative flow cannot be dissipated through mass balancing (Step 3-5), as shown in Figure-4a.

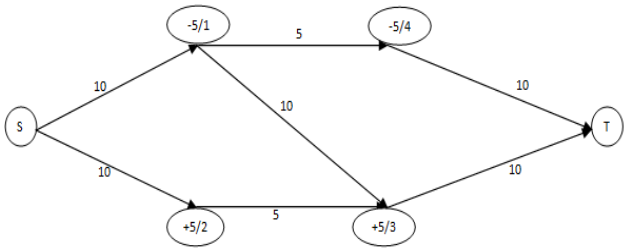

Fig. 4a. A representative network

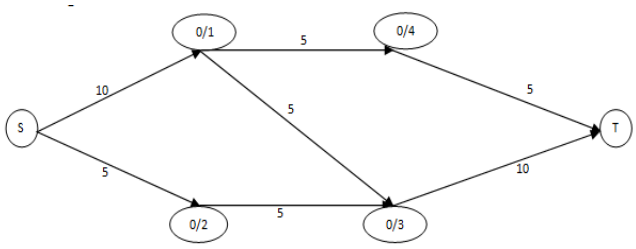

Fig. 4b. Solution of a representative network

In the Fig. 4a, it can be seen that the flow can be dissipated between nodes 1 and 3 (Step 3-5). However no further flow can be dissipated. Nodes 2 and 4 remain unbalanced as any connection between them cannot be seen. In such a case negative dissipative flow of node 4 is dumpt to sink (Step 7-9) and positive dissipative flow of node 2 is sent back to source (Step 11-13). The final solution can be seen in Fig. 4b.To see this example in eyes of theorem, following variable values used in a theorem can be established. $Q_s = 10$, $Q_t = -10$, $Q_d = 20$, $q_s = 5$, $q_t = -5$ and $\Delta q = 10$. It can be seen that these figures satisfy Eqs. (6-12).Therefore as per this theorem the solution in Figure-4b is an optimal solution. In the next section the complexity of this method is discussed.

3. Complexity analysis of mass balancing method

First of all it is needed to be checked that whether or not this method terminates within a finite time. If capacities of edges are considered as integer quantities then each mass dissipation operation will either

add at least +1 to node with negative dissipative flow or subtract the same quantity from the node with positive dissipative flow. The algorithm terminates as soon as all nodes become neutral with zero dissipative flow. Therefore the algorithm will terminate within a finite time. As it can be seen that this method is localized optimization method strives for balancing the node flows with neighbouring nodes. The worst case problem for this method can be the problem with all nodes n having either $d_n > 0$ or all nodes having $d_n < 0$. In such a case any oppositely charged nodes cannot be found to bring them into equilibrium. In such a scenario all unbalanced flows have to be dumped into source or a sink. Fig. 5, shows example of such a problem, where all the deficit flow need to be dumped to the sink.

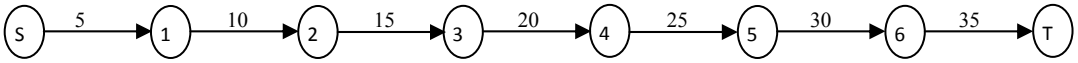

Fig. 5. Worst Case Network

In Fig. 5, there are total of 7 edges in the dataset. To remove deficit flow from node-1 the flow needs to traverse 6 edges to get dumped at the sink. Similarly to remove the flow from node-2, 5 edges are required to be traversed by the flow for the same action and so on. Thus to arrive at the final solution total edges to be traversed by the flow are equal to ½ (m^2-m) edges. Therefore worst case complexity of proposed algorithm is of order O(m^2-m). Fully balanced network is the best case for this algorithm where its complexity is nil, because fully saturated network will represent the optimal solution. This algorithm has capability to provide optimal solution in sublinear time in networks with fewer unbalanced nodes (example Fig 2).

To compare proposed method with two other popular methods in the literature 30 acyclic dense datasets (CATS, 2007) of the size of 10 nodes on different random seeds are generated. Since datasets are random hence all nodes are unbalanced. The datasets are given in Appendix-A. The results on those datasets are presented in Table-1. In Table-1 column-1 shows the dataset number, column-2 presents maximum flow obtained, column 3 gives total number of edges traversed by conventioal augmenting paths, preflow push and the proposed MBT method.

In Table 1 it can be seen that total edges traversed by MBT method are approximately only 50% of the number of edges traversed by conventional augmenting path method. However MBT method has traversed only 3% lesser edges than preflow push method. This is because these datasets have all unbalanced nodes which is not a good case for MBT method. To present systematic comparative analysis of proposed algorithm with conventional augmenting paths and preflow push methods a dataset is designed. In this dataset variations in the capacity of output edges are introduced to study effect of those changes on the performance of algorithms. The dataset is shown in Fig. 6.

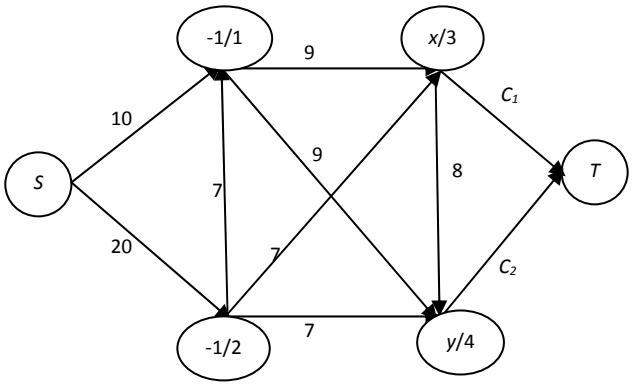

Fig. 6. Analytical Network

In Fig. 6, analytical network is presented that has two edges C_1 and C_2 connected to the sink. Any variations in the capacity of these edges result in the changes in the flow balances x and y of node 3 and node 4 respectively. A total of 5 different datasets are composed out of this structure by assigning different values to C_1 and C_2.

Table 1
Comparison on 10-node Acyclic Dense Datasets

Dataset# (1)	Max Flow (2)	Edges traversed (3)		
		Aug_Paths	Preflow_Push	MBT
1	37	114	72	57
2	51	159	112	58
3	44	176	136	68
4	37	108	95	97
5	33	115	52	95
6	42	179	41	104
7	47	209	131	72
8	43	174	134	79
9	43	129	32	101
10	42	99	34	87
11	49	189	125	63
12	39	129	62	86
13	40	178	119	54
14	35	126	98	104
15	45	180	55	78
16	51	174	62	69
17	44	185	63	83
18	50	168	72	88
19	34	162	81	86
20	26	106	73	79
21	47	223	124	66
22	38	135	106	77
23	43	131	69	84
24	51	199	53	100
25	49	164	59	83
26	53	193	178	55
27	31	129	43	106
28	55	208	74	95
29	45	94	36	102
30	56	214	124	63
Total		4749	2515	2439

The details of datasets are presented in column-1 of Table 2. In the column-1 of Table 2 it can be seen that variations in the capacity of edges C_1 and C_2 introduce corresponding variations in the values of x and y that represent flow balances of node 3 and node 4 respectively. The values of x and y are changed in the range of -1 to +1. In all cases maximum flow of the network is 30.

Table 2
Analytical Datasets

Dataset (1)						# of Edge Traversals (2)		
#	C_1	C_2	x	y	z	Aug_Paths	Preflow_Push	MBT
1	9	25	-1	-1	4	33	43	20
2	8	25	0	-1	3	33	43	15
3	7	25	+1	-1	2	29	51	12
4	7	24	+1	0	1	29	51	8
5	7	23	+1	+1	0	29	51	5

Three methods are tested on these datasets, i.e., conventional augmenting paths and preflow push methods and the proposed method of this paper. The results in terms of number of edge traversals are presented in column 2 of Table-2. It can be seen in Table-2 that there is no significant variation in number of edge traversals in augmenting paths and preflow push methods in all the 5 datasets. However number of edge traversals are consistently reduced from 20 (Dataset 1) to 5 (Dataset 5) in case of MBT method. This is because Augmenting paths and preflow push methods largely depend on

the size of the dataset. However MBT method depends on how flow balances on nodes are distributed along the network. When all the four transmission nodes had negative flow balances (Dataset 1) the speed of MBT was slowest i.e. it traversed 20 edges. However when flow balances at two nodes are changed to positive (Dataset 5) the MBT speed was fastest i.e. it traversed only 5 edges (Sublinear Time). This is because when all the nodes have negative flow balances then mass balancing (Steps 3-6) cannot be performed and number of unbalanced nodes remain the same after the mass balancing. Therefore in Dataset 1 number of unbalanced nodes after the mass balancing remains 4. These nodes can be called unbalanceable nodes and are represented by z in Table-1. Subsequently these unbalanceable nodes reduce to 3, 2, 1, and 0 in Datasets 2, 3, 4, and 5 respectively, and these changes have consistently positive effect on speed on the algorithm. Therefore it can be concluded that number of unbalanceable nodes are directly proportional to edges traversed in MBT method, i.e.,

$$\epsilon \propto z$$

$$\epsilon = kz + k \tag{30}$$

$$\epsilon = k(z + 1)$$

where

ϵ = Number of edges traversed by the MBT method, z = Number of unbalanceable nodes,

k = Coofficient depending on initial number of unbalanced nodes and problem size.

For the above problem value of k roughly equals 4, that gives approximate number of edge traversals for MBT method on all the 5 datasets. From the equation 30 it can be deduced that the proposed algorithm works based on the datasets with fewer unbalanced nodes and it has ability to solve the problem even in sublinear time if those fewer unbalanced nodes could be balanced through mass balancing (Steps 3-6). The examples of this are dataset in Figure-2 and dataset 5 of Table 2.

4. Maximization in multi-commodity flow environment

Suppose the network has more than one source. Each source delivers mixture of number of commodities and each source have mixture of commodities in different proportions. Our objective is to maximize one of those commodities. Let us call it a prime commodity. To explain the proposed algorithm it is needed to be introduced the term prime ratio, which is the ratio of prime commodity in the overall mixture for each source i, and is given by:

$$\forall_{i = 1, n} R_i = \frac{P_i}{Q_i} \tag{31}$$

n = Total number of sources

R_i= Ratio of prime commodity to overall mixture in source i

P_i = Flow of prime commodity in source i

Q_i = Total flow of mixture in source i

The algorithm can be summarized in the following steps.

1. Calculate R_i for each source i present in the network.
2. Sort the sources from the minimum R_i to the maximum R_i, such that source S_1 refers to the source with minimum R_i and S_n refers to the source with maximum R_i.
3. Apply first 10 steps of algorithm for single commodity presented in section 2.
4. The algorithm scans the node list to find the node i with $d_i > 0$.
5. The algorithm tries to find the path from node i to source node S_1 through the network. If it fails to find then it looks for source S_2 and so on until it find such a path. All the edges of this path must be backward edges and $d_{i \to f}$ must not be zero.

6. The algorithm dissipates the flow along this path according to equation 17 and updates the dissipative flow of first node of the path accordingly.
7. The algorithm repeats the steps 4 to 6 iteratively until there remains no node i with $d_i > 0$.

To show working of above algorithm the network in Fig. 2 is modified, as shown in Fig.7a.

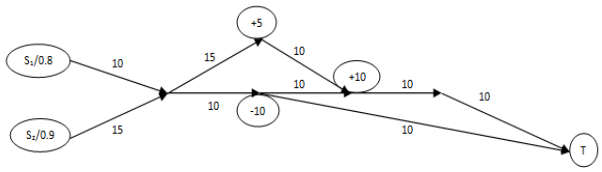

Fig. 7a. Multi-commodity Network

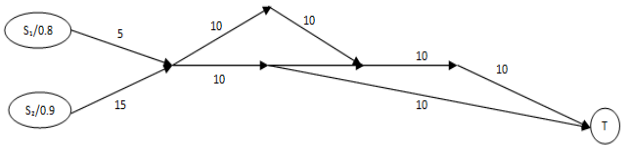

Fig. 7b. Multi-commodity Maximized Flow Network

In Fig.7a, the network has two sources with prime commodity ratios 0.8 and 0.9. The sources are sorted from minimum to maximum prime commodity ratio, hence source with lesser ratio 0.8 is labelled S_1 and source with higher ratio 0.9 is labelled S_2. The network has 3 unbalanced nodes, two with positive dissipative flows +10 and +5 and one with negative dissipative flow -10. The dissipative flows +10 and -10 can easily be cancelled as they are directly connected. However node with +5 dissipative flow still remains, that needs to be sent back to the source. To do this it is first needed to be seen whether path is available to connect this node to source S_1. If it is not available then it is needed to be established the path to source S_2. However path to source S_1 is available in this case and the positive dissipative flow can be removed along this path. The resultant network is shown in Fig. 7b, which represents maximum flow on the multi-commodity network. The interesting thing about this method is that it exactly works with same simplicity on multicommodity as that of single commodity. It saturates the network, performs mass balancing, dissipate the remaining negative flows to the sink without any knowledge of flow constituent ratios. Only when dissipating remaining positive flows to sources it chooses the sources from already sorted list based on prime commodity ratios.

5. Conclusion and future work

In this paper a mass balancing theorem is presented, in which it is shown that optimal solution of the maximum flow problem can be achieved just by balancing the flow between the unbalanced nodes. Therefore flow balancing objective has the same effect as that of maximum flow objective. Based on these results Mass Balancing Method is developed that balances the flow between unbalanced nodes with their localized interactions. Second part of mass balancing theorem is also presented in which it is shown that this method always arives at optimal solution. It is shown with examples that this method based on localized balancing between nodes end up in a maximum flow solution. It is also shown that this method has worst case complexity of order $O(m^2\text{-}m)$, where m is the number of edges. 30 random acyclic dense datasets were also generated and the mass balance method maximized the flow much faster than the conventional augmenting paths method and little faster than the conventional preflow push algorithm. The method is also anaysed and compared with other methods on a analytical dataset by introducing variations in it, where it is shown that proposed method performs much faster than other methods when the dataset has very good mass ballancing opportunity to cancel excess and deficit flows. It is shown experimentally and emperically (Equation 30) that the proposed method can solve some of the problems even in sublinear time. This approach is also extended to multicommodity problem and have shown that how this approach has made this task easier. To this end it can be concluded that Mass Balancing Theorem has potential to become a major player in devising the new optimization methods of sublinear time complexity for various network related problems for the coming decades.

Acknowledgement

It is gratefully acknowledged that Professor Michael Todinov arranged financial help from the Leverhulme trust with the research grant F/00 382/J ‘High-speed algorithms for the output flow in repairable flow networks’, for this piece of research.

References

Ahuja, R. K., Magnanti, T. L., & Orlin J. B. (1988). Network flows. Working paper: OR 185-88. *Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA*.

Ahuja, R. K., & Orlin, J. B. (1989). A fast and simple algorithm for the maximum flow problem. *Operations Research*, 37(5), 748-759.

Ahuja, R. K., & Orlin, J. B. (1991). Distance-directed augmenting path algorithms for maximum flow and parametric maximum flow problems. *Naval Research Logistics*, 38, 413-430.

Albert, R., & Barabasi, A. L. (2002). Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74(1), 47-97.

Boldyreff, A. W. (1955). Determination of the maximal Steady State Flow of Traffic Through a Railroad Network. *JORSA*, 3(4), 443-465.

CATS. (2007). Combinatorial Algorithms Test Sets.

http://www.avglab.com/andrew/CATS/gens/, accessed January 2012.

Cherkasky, R. V. (1977). Algorithm for construction of maximum flow in networks with complexity of $O(V^2\sqrt{E})$ operation. *Mathematical Methods of Solution of Economical Problems*, 7, 112-125 (in Russian).

Chandran, B. G., & Hochbaum, D. S. (2009). A Computational Study of the Pseudoflow and Push-Relabel Algorithms for the Maximum Flow Problem. Operations Research Vol. 57, No. 2, March–April 2009, pp. 358–376 issn 0030-364X _ eissn 1526-5463 _ 09 _ 5702 _ 0358.

Danzig, G. B., & Fulkerson, D. R. (1956). *On Max-Flow Min-Cut Theorem of Networks*. In H.W. Kuhn and A. W. Tucker (ed.), Linear Inequalities and Related Systems, Annals of Mathematics Study 38, Princeton University Press, 215-221.

Dinic, E. A. (1970). Algorithm for solution of a problem of maximum flow in networks with power estimation. *Soviet Math*. Dokl. 11, 1277-1280.

Dong, J., Wei, L., Cai, C., & Chen, Z. (2009). Draining algorithm for the maximum flow problem. *International Conference on Communications and Mobile Computing.*

Edmonds, J., & Karp, R. M. (1972). Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19, 248-264.

Elias, P., Feinstein, A., & Shanon C. E. (1956). Note on maximum flow through a network. *IRE Transactions on Information Theory*, 117-119.

Ford, L. R. Jr., & Fulkerson D. R. (1956). Maximal flow through a network. *Canadian Journal of Mathematics*, 8, 399-404.

Gabow, H. N. (1985). Scaling algorithms for network problems. *Journal of Computer and System Sciences*, 31, 148-168.

Galil, Z. (1980). $O(V^5/3E^{2/3})$ algorithm for the maximum flow problem. *Acta Informatica*, 14, 221-242.

Goldberg, A. V. (1985). *A new max-flow algorithm*. Technical Report MIT/LCS/TM-291, Laboratory for Computer Science, MIT, Cambridge, Mass.

Goldberg, A. V., & Tarjan R. E. (1986). *A new approach to the maximum flow problem, in Proc. 18th Annual ACM Symposium on the Theory of Computing*. Association for Computing Machinery, New York, pp. 136-146.

Goldberg, A. V., & Tarjan, R.E. (1988). A New Approach to the Maximum-Flow Problem. *Journal of the Association for Computing Machinery*, 35(4), 921-940.

Hochbaum, D. S. (1997). The pseudoflow algorithm and the pseudoflow-based simplex for the maximum flow problem. *Integer Programming and Combinatorial Optimization*, 1412, 325-337.

Hochbaum D. S. (2001). A new-old algorithm for minimum-cut and maximum-flow in closure graphs. *Networks*, 37(4) 171-193.

Hochbaum D. S. (2003). *A pseudoflow algorithm for the directed minimum cut problem*. Manuscript, UC Berkeley.

Hochbaum, D. S., & Orlin, J.B. (2007). The pseudoflow algorithm in $O(mn\log n^2/m)$ and $O(n^3)$. UC Berkeley manuscript. Submitted.

Hochbaum, D. S. (2008). The Pseudo-flow Algorithm. A new algorithm for the maximum flow problem. *Operations Research* (Informs) 56(4), 992-1009.

Karzanov, A. V. (1974). Determining the maximal flow in a network by the method of pre-flows. *Soviet Mathematics Doklady*, 15, 434-437.

Lerchs, H., & Grossman, I. (1965). Optimum design of open pit mines. *Transactions, C.I.M*, 68, 17-24.

Malhotra, V. M., Kumar, M. P., & Maheshwari S. N. (1978). An $O(V^3)$ Algorithm for Finding Maximum Flows in Networks. *Information Processing Letters*, 7, 277-278.

Orlin, J. B., & Ahuja R. K. (1987). *New distance-directed algorithms for maximum flow and parametric maximum flow problems*. Working Paper 1908-87, Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA.

Radzik T. (1993). Parametric Flows, Weighted means of cuts, and fractional combinatorial optimization. *In Complexity in Numerical Optimization*, World Scientific, P. M. Pardalos Ed. 351-386.

Sawitzki D. (2004) Implicit flow maximization by iterative squaring. P. Van Emde Boas et al. (Eds.): SOFSEM 2004, *Lecture Notes in Computer Science*, 2932, 301–313.

Tarjan, R. E. (1984). A simple version of Karzanov's blocking flow algorithm. *Operations Research Letters*, 2, 265-268.

Tarjan, R. E. (1986). Algorithms for Maximum Network Flow. *Mathematical Programming*, 26, 1-11.

Todinov, M. (2011A). Extended comments. A review of this paper, private communication, email dated: 04-02-2011.

Todinov, M. (2011B). Fast augmentation algorithms for maximising the flow in repairable flow networks after a component failure. *IEEE 11th International Conference on Computer and Information Technology.*

Appendix-A

Acyclic Dense Datasets

In this appendix details of 30 randomly generated acyclic dense datasets is given. Code is downloaded from (CATS, 2007). The left most column shows the tail node. The main column contains the list of head nodes along with the capacity of the edge connecting them to the tail node.

	Dataset #1	**Dataset #2**	**Dataset #3**
1	2-3, 3-1, 4-6, 5-3, 6-9, 7-4, 8-5, 9-2, 10-6	2-7, 3-5, 4-9, 5-4, 6-8, 7-4, 8-7, 9-3, 10-9	2-3, 3-10, 4-7, 5-9, 6-8, 7-3, 8-3, 9-7, 10-6
2	3-8, 4-8, 5-7, 6-1, 7-1, 8-3, 9-7, 10-9	3-8, 4-3, 5-1, 6-3, 7-3, 8-6, 9-7, 10-3	3-6, 4-8, 5-1, 6-10, 7-7, 8-5, 9-1, 10-2
3	4-6, 5-1, 6-10, 7-1, 8-10, 9-2, 10-4	4-1, 5-3, 6-1, 7-7, 8-2, 9-10, 10-9	4-7, 5-10, 6-6, 7-10, 8-8, 9-6, 10-8
4	5-6, 6-10, 7-6, 8-4, 9-1, 10-9	5-1, 6-5, 7-4, 8-4, 9-10, 10-8	5-2, 6-6, 7-2, 8-4, 9-3, 10-5
5	6-2, 7-9, 8-8, 9-5, 10-5	6-4, 7-9, 8-3, 9-5, 10-3	6-10, 7-8, 8-9, 9-4, 10-6
6	7-3, 8-4, 9-2, 10-9	7-10, 8-3, 9-4, 10-5	7-7, 8-8, 9-10, 10-1
7	8-8, 9-9, 10-2	8-8, 9-9, 10-7	8-5, 9-10, 10-7
8	9-4, 10-2	9-3, 10-4	9-9, 10-6
9	10-5	10-3	10-3
	Dataset #4	**Dataset #5**	**Dataset #6**
1	2-1, 3-3, 4-7, 5-2, 6-10, 7-9, 8-2, 9-8, 10-10	2-9, 3-6, 4-4, 5-1, 6-1, 7-7, 8-1, 9-10, 10-3	2-9, 3-6, 4-5, 5-7, 6-2, 7-6, 8-2, 9-8, 10-3
2	3-4, 4-1, 5-10, 6-7, 7-5, 8-7, 9-3, 10-9	3-1, 4-7, 5-4, 6-3, 7-4, 8-4, 9-1, 10-2	3-6, 4-5, 5-3, 6-1, 7-1, 8-8, 9-2, 10-7
3	4-8, 5-9, 6-10, 7-6, 8-6, 9-7, 10-9	4-8, 5-8, 6-7, 7-10, 8-4, 9-3, 10-9	4-8, 5-4, 6-1, 7-5, 8-7, 9-6, 10-5
4	5-1, 6-4, 7-8, 8-4, 9-7, 10-2	5-5, 6-3, 7-9, 8-10, 9-8, 10-8	5-7, 6-7, 7-10, 8-4, 9-10, 10-10
5	6-2, 7-7, 8-2, 9-5, 10-7	6-5, 7-7, 8-4, 9-8, 10-9	6-1, 7-10, 8-5, 9-8, 10-6
6	7-7, 8-7, 9-9, 10-3	7-6, 8-7, 9-8, 10-2	7-1, 8-2, 9-9, 10-10
7	8-10, 9-8, 10-4	8-10, 9-4, 10-3	8-7, 9-9, 10-9
8	9-3, 10-10	9-6, 10-9	9-3, 10-6
9	10-1	10-1	10-2
	Dataset #7	**Dataset #8**	**Dataset #9**
1	2-3, 3-9, 4-5, 5-9, 6-7, 7-3, 8-9, 9-5, 10-2	2-4, 3-5, 4-5, 5-10, 6-4, 7-9, 8-9, 9-6, 10-5	2-10, 3-1, 4-8, 5-1, 6-5, 7-1, 8-2, 9-5, 10-10
2	3-6, 4-3, 5-1, 6-6, 7-2, 8-6, 9-4, 10-9	3-4, 4-9, 5-3, 6-6, 7-10, 8-9, 9-1, 10-2	3-5, 4-4, 5-1, 6-2, 7-3, 8-2, 9-9, 10-8
3	4-2, 5-6, 6-5, 7-2, 8-3, 9-10, 10-8	4-1, 5-5, 6-2, 7-1, 8-6, 9-1, 10-1	4-1, 5-4, 6-10, 7-1, 8-5, 9-1, 10-10
4	5-5, 6-7, 7-3, 8-8, 9-1, 10-1	5-7, 6-7, 7-4, 8-4, 9-8, 10-5	5-8, 6-2, 7-7, 8-5, 9-8, 10-8
5	6-2, 7-6, 8-8, 9-8, 10-4	6-6, 7-10, 8-5, 9-9, 10-7	6-5, 7-9, 8-1, 9-5, 10-4
6	7-2, 8-1, 9-2, 10-4	7-1, 8-3, 9-8, 10-7	7-2, 8-1, 9-3, 10-8
7	8-3, 9-8, 10-10	8-10, 9-8, 10-2	8-2, 9-4, 10-6
8	9-7, 10-8	9-5, 10-8	9-5, 10-2
9	10-7	10-6	10-8

	Dataset #10	Dataset #11	Dataset #12
1	2-2, 3-2, 4-4, 5-5, 6-4, 7-9, 8-3, 9-3, 10-10	2-9, 3-10, 4-7, 5-10, 6-5, 7-10, 8-3, 9-5, 10-4	2-3, 3-2, 4-3, 5-9, 6-1, 7-6, 8-6, 9-7, 10-4
2	3-4, 4-5, 5-5, 6-4, 7-9, 8-6, 9-4, 10-3	3-7, 4-7, 5-4, 6-7, 7-2, 8-3, 9-8, 10-4	3-7, 4-1, 5-10, 6-10, 7-1, 8-2, 9-2, 10-10
3	4-2, 5-7, 6-8, 7-9, 8-2, 9-3, 10-7	4-4, 5-7, 6-6, 7-9, 8-4, 9-2, 10-8	4-1, 5-7, 6-5, 7-7, 8-3, 9-6, 10-5
4	5-10, 6-2, 7-6, 8-3, 9-3, 10-7	5-10, 6-9, 7-8, 8-7, 9-8, 10-7	5-10, 6-8, 7-9, 8-9, 9-4, 10-3
5	6-7, 7-10, 8-6, 9-2, 10-1	6-10, 7-4, 8-5, 9-10, 10-6	6-8, 7-3, 8-10, 9-2, 10-8
6	7-2, 8-8, 9-8, 10-7	7-4, 8-2, 9-6, 10-4	7-10, 8-9, 9-10, 10-3
7	8-1, 9-6, 10-9	8-6, 9-10, 10-3	8-3, 9-7, 10-3
8	9-6, 10-7	9-5, 10-8	9-10, 10-5
9	10-8	10-5	10-9
	Dataset #13	**Dataset #14**	**Dataset #15**
1	2-9, 3-1, 4-9, 5-7, 6-9, 7-6, 8-10, 9-2, 10-6	2-1, 3-2, 4-10, 5-2, 6-4, 7-3, 8-5, 9-6, 10-9	2-8, 3-10, 4-3, 5-4, 6-2, 7-2, 8-3, 9-3, 10-10
2	3-2, 4-10, 5-5, 6-6, 7-9, 8-6, 9-1, 10-9	3-3, 4-5, 5-10, 6-1, 7-3, 8-7, 9-2, 10-8	3-4, 4-1, 5-8, 6-2, 7-4, 8-10, 9-7, 10-9
3	4-2, 5-4, 6-5, 7-3, 8-5, 9-9, 10-2	4-10, 5-1, 6-6, 7-5, 8-4, 9-2, 10-8	4-2, 5-5, 6-2, 7-3, 8-6, 9-6, 10-6
4	5-8, 6-6, 7-6, 8-8, 9-8, 10-3	5-1, 6-6, 7-4, 8-2, 9-10, 10-1	5-1, 6-2, 7-4, 8-9, 9-2, 10-9
5	6-3, 7-5, 8-8, 9-6, 10-6	6-5, 7-3, 8-2, 9-9, 10-8	6-1, 7-9, 8-1, 9-5, 10-1
6	7-6, 8-9, 9-1, 10-2	7-6, 8-10, 9-2, 10-2	7-7, 8-1, 9-3, 10-4
7	8-10, 9-2, 10-3	8-1, 9-4, 10-8	8-9, 9-7, 10-9
8	9-2, 10-2	9-7, 10-8	9-2, 10-2
9	10-8	10-1	10-6
	Dataset #16	**Dataset #17**	**Dataset #18**
1	2-3, 3-10, 4-3, 5-5, 6-10, 7-10, 8-2, 9-8, 10-6	2-10, 3-1, 4-8, 5-10, 6-2, 7-2, 8-5, 9-4, 10-2	2-9, 3-3, 4-6, 5-8, 6-1, 7-8, 8-4, 9-8, 10-6
2	3-8, 4-6, 5-10, 6-3, 7-2, 8-6, 9-4, 10-4	3-6, 4-7, 5-9, 6-2, 7-2, 8-1, 9-4, 10-7	3-8, 4-1, 5-9, 6-3, 7-8, 8-8, 9-1, 10-5
3	4-3, 5-5, 6-8, 7-2, 8-6, 9-6, 10-4	4-9, 5-6, 6-8, 7-3, 8-1, 9-9, 10-8	4-8, 5-2, 6-8, 7-9, 8-6, 9-4, 10-1
4	5-7, 6-5, 7-9, 8-4, 9-1, 10-8	5-6, 6-3, 7-9, 8-6, 9-5, 10-8	5-9, 6-5, 7-3, 8-2, 9-9, 10-9
5	6-1, 7-4, 8-5, 9-8, 10-9	6-4, 7-1, 8-2, 9-5, 10-8	6-4, 7-5, 8-3, 9-10, 10-3
6	7-2, 8-7, 9-2, 10-10	7-5, 8-4, 9-8, 10-4	7-7, 8-6, 9-1, 10-8
7	8-1, 9-9, 10-7	8-8, 9-1, 10-6	8-9, 9-5, 10-10
8	9-7, 10-2	9-6, 10-3	9-9, 10-5
9	10-5	10-8	10-5
	Dataset #19	**Dataset #20**	**Dataset #21**
1	2-2, 3-4, 4-9, 5-1, 6-5, 7-9, 8-3, 9-8, 10-1	2-1, 3-3, 4-3, 5-5, 6-4, 7-3, 8-2, 9-10, 10-4	2-10, 3-9, 4-5, 5-5, 6-8, 7-7, 8-9, 9-9, 10-4
2	3-10, 4-7, 5-9, 6-8, 7-2, 8-4, 9-4, 10-8	3-9, 4-6, 5-5, 6-6, 7-8, 8-7, 9-6, 10-4	3-7, 4-1, 5-4, 6-4, 7-9, 8-5, 9-10, 10-9
3	4-9, 5-8, 6-4, 7-4, 8-1, 9-10, 10-6	4-6, 5-4, 6-4, 7-10, 8-8, 9-8, 10-5	4-7, 5-5, 6-1, 7-10, 8-7, 9-1, 10-3
4	5-6, 6-7, 7-1, 8-4, 9-1, 10-5	5-2, 6-5, 7-8, 8-7, 9-7, 10-8	5-2, 6-8, 7-6, 8-5, 9-4, 10-2
5	6-1, 7-1, 8-2, 9-9, 10-6	6-6, 7-4, 8-10, 9-9, 10-3	6-7, 7-7, 8-6, 9-10, 10-10
6	7-6, 8-5, 9-1, 10-3	7-10, 8-2, 9-7, 10-5	7-7, 8-8, 9-7, 10-9
7	8-8, 9-6, 10-2	8-7, 9-4, 10-1	8-10, 9-10, 10-3
8	9-2, 10-7	9-5, 10-2	9-6, 10-3
9	10-5	10-4	10-4
	Dataset #22	**Dataset #23**	**Dataset #24**
1	2-2, 3-7, 4-3, 5-9, 6-5, 7-1, 8-5, 9-6, 10-4	2-1, 3-2, 4-10, 5-8, 6-1, 7-2, 8-10, 9-4, 10-8	2-9, 3-6, 4-5, 5-9, 6-2, 7-4, 8-9, 9-7, 10-8
2	3-2, 4-10, 5-9, 6-6, 7-6, 8-10, 9-4, 10-1	3-9, 4-8, 5-4, 6-8, 7-6, 8-2, 9-2, 10-1	3-3, 4-2, 5-3, 6-7, 7-4, 8-7, 9-2, 10-9
3	4-8, 5-4, 6-2, 7-9, 8-9, 9-3, 10-8	4-7, 5-10, 6-6, 7-6, 8-9, 9-7, 10-7	4-7, 5-4, 6-5, 7-5, 8-1, 9-5, 10-7
4	5-8, 6-6, 7-10, 8-9, 9-8, 10-5	5-1, 6-9, 7-2, 8-4, 9-6, 10-10	5-10, 6-9, 7-10, 8-3, 9-1, 10-6
5	6-7, 7-2, 8-6, 9-4, 10-5	6-7, 7-8, 8-9, 9-6, 10-1	6-9, 7-8, 8-9, 9-1, 10-6
6	7-1, 8-2, 9-10, 10-1	7-10, 8-2, 9-2, 10-5	7-1, 8-2, 9-1, 10-8
7	8-6, 9-6, 10-6	8-4, 9-3, 10-6	8-8, 9-10, 10-9
8	9-9, 10-3	9-8, 10-5	9-5, 10-7
9	10-9	10-6	10-1
	Dataset #25	**Dataset #26**	**Dataset #27**
1	2-3, 3-7, 4-5, 5-3, 6-6, 7-3, 8-6, 9-10, 10-9	2-6, 3-10, 4-10, 5-2, 6-10, 7-5, 8-6, 9-7, 10-10	2-1, 3-6, 4-6, 5-2, 6-3, 7-3, 8-3, 9-8, 10-1
2	3-9, 4-2, 5-5, 6-9, 7-3, 8-6, 9-2, 10-8	3-2, 4-8, 5-1, 6-2, 7-5, 8-5, 9-6, 10-2	3-9, 4-3, 5-6, 6-7, 7-10, 8-8, 9-8, 10-8
3	4-9, 5-10, 6-4, 7-8, 8-2, 9-4, 10-6	4-3, 5-6, 6-5, 7-9, 8-5, 9-2, 10-1	4-1, 5-5, 6-4, 7-9, 8-3, 9-3, 10-5
4	5-7, 6-4, 7-8, 8-3, 9-9, 10-2	5-9, 6-5, 7-6, 8-5, 9-1, 10-4	5-8, 6-4, 7-3, 8-6, 9-3, 10-8
5	6-5, 7-6, 8-1, 9-4, 10-7	6-2, 7-9, 8-2, 9-7, 10-4	6-4, 7-5, 8-2, 9-1, 10-8
6	7-6, 8-9, 9-5, 10-1	7-2, 8-1, 9-1, 10-6	7-8, 8-9, 9-6, 10-3
7	8-1, 9-5, 10-7	8-7, 9-8, 10-8	8-6, 9-10, 10-1
8	9-5, 10-10	9-1, 10-8	9-9, 10-7
9	10-7	10-10	10-6
	Dataset #28	**Dataset #29**	**Dataset #30**
1	2-10, 3-2, 4-10, 5-6, 6-3, 7-10, 8-3, 9-7, 10-7	2-2, 3-3, 4-2, 5-8, 6-8, 7-3, 8-1, 9-8, 10-10	2-10, 3-7, 4-2, 5-4, 6-8, 7-7, 8-7, 9-8, 10-10
2	3-5, 4-2, 5-1, 6-1, 7-10, 8-2, 9-6, 10-9	3-5, 4-2, 5-7, 6-8, 7-1, 8-5, 9-3, 10-8	3-1, 4-1, 5-4, 6-7, 7-1, 8-8, 9-3, 10-3
3	4-2, 5-10, 6-10, 7-10, 8-1, 9-10, 10-2	4-4, 5-3, 6-9, 7-4, 8-7, 9-5, 10-6	4-7, 5-10, 6-10, 7-2, 8-1, 9-1, 10-6
4	5-3, 6-9, 7-7, 8-9, 9-8, 10-10	5-5, 6-4, 7-6, 8-3, 9-9, 10-1	5-10, 6-8, 7-9, 8-7, 9-1, 10-8
5	6-5, 7-8, 8-10, 9-2, 10-3	6-4, 7-8, 8-7, 9-6, 10-7	6-5, 7-2, 8-2, 9-5, 10-3
6	7-2, 8-5, 9-3, 10-10	7-10, 8-10, 9-9, 10-4	7-5, 8-10, 9-1, 10-9
7	8-4, 9-4, 10-5	8-4, 9-4, 10-9	8-5, 9-5, 10-7
8	9-4, 10-7	9-8, 10-7	9-8, 10-9
9	10-5	10-8	10-4