UNIVERSITA' DEGLI STUDI DI TORINO

M.Sc. Stochastics and Data Science

ALGORITHMS 2020-2021

Zachary's karate club

Maximum flow algorithms to predict fission in groups

Authors:

Jacopo NICOLOSI (858300)

Chiara PRETI (853788)

1 Introduction

Zachary's karate club is a social network of a university karate club, described in the paper *An Information Flow Model for Conflict and Fission in Small Groups*, written by Wayne W. Zachary. His aim was to collect data from a voluntary sport association in order to construct a new formal model for a traditional anthropological problem: fission in small groups.
We are interested in this problem as the process leading to fission can be formalised through a mathematical model. It is indeed possible to generalize the case study by viewing it as a flow of sentiments and information across the ties of the social network. This flow is unequal as it relies on the different relationships between the elements of the network. This inequality led to the formation of subgroups with more internal stability than the group as a whole and, eventually, to fission. We will show that it is possible to predict in a quite accurate way how the group will split by means of methods which measure the potential information flow across each edge of the network.

The karate club was observed from 1970 to 1972. In these three years, the club maintained between 50 and 100 members and its activities included social events as well as regularly scheduled karate lessons. The political organization of the club was informal and, even if there were four officers, most decisions were made at club meetings by majority vote. For the classes, the club employed a part-time karate instructor, who will be referred to as Mr. Hi as a pseudonym. At the beginning of the study, there was an incipient conflict between the club president, mentioned with the pseudonym of John A., and Mr. Hi over the price of karate lessons. The former wished to stabilize prices whereas the latter claimed the authority to set his own lesson fees and raise the prices. As time passed, the entire club became divided over this issue: the supporters of Mr. Hi saw him as a mentor who was just trying to meet his own physical needs, whereas the supporters of John A. saw Mr. Hi as a paid employee who was trying to raise his salary. Eventually, Mr. Hi was fired for attempting to raise lesson prices unilaterally and his supporters resigned to form a new organization headed by Mr. Hi.
The key point here is that the factions were merely ideological groupings: there was indeed no political division and the factions were neither named nor recognized to exist by club members. However, they were groups which emerged from the existing network of friendships among club members at times of political crisis: at direct conflict, the individuals selectively interacted indeed only with who shared the same ideological position. This selective association during confrontations had the effect of strengthening the friendship bonds within these ideological groups and separate more and more the two factions, by the pattern of selective reinforcement. By studying how the elements of the network interacted among them and by analyzing the flow of information, it is then possible to predict the way the elements will split in two groups.

2 Mathematical tools for the flow networks

Before studying the algorithms, it is important to have in mind the mathematical tools and concepts we will need: in this section, we give a definition of flow networks, discuss their properties and define the maximum-flow problem precisely.
A **flow network** $G = (V, E)$ is a directed graph in which each edge $(u, v) \in E$ has a non negative capacity $c(u, v) \geq 0$. For convenience, if $(u, v) \notin E$ we put $c(u, v) = 0$ and we also disallow self-loops.
We distinguish two vertices in a flow network, a **source** s and a **sink** t, and we assume that each vertex lies on some path from the source to the sink. The graph is therefore connected and, since each vertex other than s has at least one entering edge, we have that $|E| \geq |V| - 1$.
Now let $G = (V, E)$ be a flow network with a capacity function c and let s be the source of the network and t the sink. A **flow** in G is a real-valued function $f : V \times V \to \mathbb{R}$, where the quantity $f(u, v)$ is the flow from vertex u to vertex v, which satisfies the following two properties:
Capacity constraint: For all $u, v \in V$, we require $0 \leq f(u, v) \leq c(u, v)$.
Flow conservation: For all $u \in V - \{s, t\}$, we require

$$\sum_{v \in V} f(v, u) = \sum_{v \in V} f(u, v).$$

The capacity constraint simply says that the flow from one vertex to another must be non-negative and must not exceed the given capacity. The flow-conservation property says that the total flow into a vertex other than the source or sink must equal the total flow out of that vertex. Moreover, when $(u, v) \notin E$, there can be no flow from u to v and $f(u, v) = 0$. We add that, in order to give a direction to the flow, one between $f(u, v)$ and $f(v, u)$ must be equal to zero. The value $|f|$ of a flow f is defined as the total flow out of the source minus the flow into the source, i.e.

$$|f| = \sum_{v \in V} f(s, v) - \sum_{v \in V} f(v, s). \tag{1}$$

In the **maximum-flow problem**, we are given a flow network G with source s, sink t and capacity c and we wish to find a flow of maximum value.
Now we want to define the **residual network** G_f, which consists of edges with capacities that represent how we can change the flow on G. G_f is indeed made of edges of G which can admit a residual capacity, that is an amount of additional flow equal to the edge's capacity minus the flow on that edge. Moreover, as an algorithm manipulates the flow to increase the total flow, it might need to decrease the flow on a particular edge. In order to represent a possible decrease of a positive flow $f(u, v)$ on an edge in G, we place an edge (v, u) into G_f with residual capacity $c_f(v, u) = c(v, u) + f(u, v)$: increasing the flow on this edge corresponds to decreasing the flow in (u, v), at most canceling it out, and potentially also increasing the actual flow in (v, u).

More formally, let us suppose that we have a flow network $G = (V, E)$ with source s and sink t and capacity c. Let f be a flow on G, and consider a pair of vertices $u, v \in V$. We define the **residual capacity** $c_f(u, v)$ by

$$c_f(u,v) = \begin{cases} c(u,v) - f(u,v) & \text{if } f(u,v) > 0 \\ c(u,v) + f(v,u) & \text{if } f(v,u) \geq 0 \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

Then, the residual network of G induced by f is the network $G_f = (V, E_f)$, where $E_f = \{(u, v) \in V \times V : c_f(u, v) > 0\}$: each edge of the residual network can admit a flow. A flow in a residual network provides a roadmap for adding flow to the original flow network. If f is a flow in G and f' is a flow in the corresponding residual network G_f, we define $f \uparrow f'$, the **augmentation** of flow f by f', to be a function from $V \times V$ to $\mathbb{R}$, defined by

$$(f \uparrow f')(u,v) = \begin{cases} f(u,v) + f'(u,v) - f'(v,u) & \text{if } (u,v) \in E \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

We increase the flow on (u, v) by $f'(u, v)$, but decrease it by $f'(v, u)$ because pushing flow on the reverse edge in the residual network signifies decreasing the flow in the original network. Pushing flow on the reverse edge in the residual network is also known as **cancellation**, which is crucial for any maximum-flow algorithm. It can be proven that the function $f \uparrow f'$ defined in 3 is a flow in G with value $|f \uparrow f'| = |f| + |f'|$.
Given a flow network $G = (V, E)$ and a flow f, we can also define an **augmenting path** p: it is a simple path from s to t in the residual network G_f. By the definition of the residual network, we may increase the flow on an edge (u, v) of an augmenting path by up to $c_f(u, v)$ without violating the capacity constraint in the original flow network G. We call the maximum amount by which we can increase the flow on each edge in an augmenting path p the **residual capacity** of p, that is given by $c_f(p) = \min\{c_f(u, v) : (u, v) \text{ is on } p\}$. Then if we set

$$f'(u,v) = \begin{cases} c_f(p) & \text{if } (u,v) \text{ is on } p \\ 0 & \text{otherwise,} \end{cases}$$

we have that $|f'| = c_f(p) > 0$. This means that the flow is strictly increased by the augmentation: $|f \uparrow f'| = |f| + |f'| > |f|$.
For our application, we also have to introduce the concept of a cut in a flow network. A **cut** (S, T) of flow network $G = (V, E)$ is a partition of V into S and $T = V - S$ such that $s \in S$ and $t \in T$. If f is a flow, then the **net flow** $f(S, T)$ across the cut (S, T) is defined to be

$$f(S,T) = \sum_{u \in S} \sum_{v \in T} f(u,v) - \sum_{u \in S} \sum_{v \in T} f(v,u).$$

The **capacity of the cut** (S, T) is

$$c(S,T) = \sum_{u \in S} \sum_{v \in T} c(u,v).$$

A **minimum cut** of a network is a cut whose capacity is minimum over all cuts of the network.
We also state the important max-flow min-cut theorem, which says that the value of a maximum flow is equal to the capacity of a minimum cut.

Theorem (max-flow min-cut). If f is a flow in a flow network $G = (V, E)$ with source s and sink t, then the following conditions are equivalent:

1. f is a maximum flow in G.
2. The residual network G_f contains no augmenting paths.
3. $|f| = c(S, T)$ for some cut (S, T) of G.

Remark 1. For every flow f and cut (S, T), we have $|f| \leq c(S, T)$. Hence the cut of the condition 3 is a minimum cut. Moreover, it can be proved that, given a maximum flow f, a minimum cut is provided by

$$S = \{v \in V : \text{ there exists a path } p \text{ on } G_f \text{ from } s \text{ to } v\}$$
$$T = V - S.$$

3 Algorithms for maximum flow problems

3.1 Ford Fulkerson

In this section we analyze a first way to solve the maximum-flow problem: the Ford-Fulkerson algorithm. It relies on three important mathematical concepts we have presented before: residual networks, augmenting paths and cuts. The aim of this method is to increase the flow in the network. The flow is initialised to be 0 already in the class. Then, at each iteration, the flow value in G is increased by finding an augmenting path p in the associated residual network G_f, which easily identifies specific edges in G for which we can change the flow. Although each iteration of the Ford-Fulkerson method increases the value of the flow, it can happen that the flow on any particular edge of G may increase or decrease: as we said before, decreasing the flow on some edges may be necessary in order to enable an algorithm to send more flow from the source to the sink. Therefore, the act of updating the flow can be done either by increasing the flow of the capacity of the augmenting path or by decreasing the flow in the opposite direction of the same quantity. The flow is repeatedly augmented until the residual network has no more augmenting paths. The max-flow min-cut theorem then asserts that, upon termination, this process yields a maximum flow. The pseudocode is the following:

Algorithm 1 FORD-FULKERSON(G)

```
1: while there exists a path $p$ from $s$ to $t$ in the residual network $G_f$
2:     $c_f(p) = \min\{c_f(u,v) : (u,v) \text{ is in } p\}$
3:     for each edge $(u,v)$ in $p$
4:         increase the flow of $(u,v)$ by $c_f(p)$
```

The running time of the Ford-Fulkerson depends on how we find the augmentig path p in the line 1: the algorithm might indeed not even terminate if the flow do not converge to the maximum flow value by successive augmentations.
We have indeed that the other operations are $O(E)$: finding the residual graph G_f costs $O(E)$, the cost of line 2 is also $O(E)$, the for cycle is performed in the worst case for each edge of the graph, so E times, and line 4 has a costant cost.
A way to find the augmentig path p is by using a **breath-first search**, which allows us to choose the augmeting path as a *shortest* (in terms of number of edges traversed) path from s to t. The Ford-Fulkerson implemented in this way is called **Edmonds-Karp algorithm**. As we have seen in class, finding a path from s to t with BFS is $O(E)$. We have thus that the cost of each iteration of the while loop is $O(E)$.
We now have to determine how many iterations are performed, that boils down to finding how many paths from s to t can be found in the residual graph G_f. We will show that if the Edmonds-Karp algorithm is run on a flow network $G = (V, E)$, with source s and sink t, the total number of flow augmentations performed by the algorithm is $O(VE)$, therefore the total running time for the Edmonds-Karp algorithm is $O(VE^2)$.

From now on the word *shortest* will refer to the distance in terms of number of edges, neglecting momentaneusly the capacities of the edges and assuming them to be unitary. The analysis of the cost depends on the distances between vertices in the residual network G_f, this distance can be defined in this way:

$$\delta_f(u,v) = |\{\text{edges in } G_f \text{ belonging to a shortest path from } u \text{ to } v\}|.$$

It can be shown that for all vertices $v \in V - \{s,t\}$, $\delta_f(s,v)$ does not descrease with each flow augmentation. This result can be used to give an upper bound for the number α of augmentations.

Theorem. The number α of augmentations is bounded by $\frac{EV}{2}$.

Proof. Firstly, we define an edge (u,v) in G_f to be **critical** if, after the augmentation, it will disappear from G_f, that is if $c_f(p) = c_f(u,v)$ and (u,v) belongs to p. It is then clear that the number of augmentations will be at most equal to the total number of critical edges appeared in the process. We will show that each of the E edges can become critical at most $\frac{V}{2}$ times.
Let us consider an edge $(u,v) \in E$. When (u,v) is critical for the first time, since augmenting paths are by definition shortest paths, we have that $\delta_f(s,v) = \delta_f(s,u)+1$. Then, once the flow is augmented, the critical edge (u,v) disappears from G_f. This edge can again be included in the residual graph if (v,u) is on a successive augmenting path. In this situation, we have that again as before $\delta_{f'}(s,u) = \delta_{f'}(s,v) + 1$, where f' is the new augmented flow. Since $|f'| > |f|$ and, as stated above, δ_f is not decreasing, we obtain that

$$\delta_{f'}(s,u) = \delta_{f'}(s,v) + 1 \geq \delta_f(s,v) + 1 = \delta_f(s,u) + 2.$$

Consequently, (u,v) becomes critical again when the the distance of u from the source s increases by at least 2. At the beginning, this distance is at least 0 and it can only increase. The last time it becomes critical, the distance can be at most $V-2$, as in the case of maximun distance u is the last node before the sink which is then v. Hence, after the first time, (u,v) can become critical again at most $\frac{V-2}{2}$ times, so in total it becomes critical at most $\frac{V}{2}$ times, situation which is followed by an augmentation of the flow. Thus, since there are E edges, the total number of augmentations performed in the graph is at most $\frac{EV}{2}$. □

To conclude, this theorem implies that the number of augmenting paths that can be found is of the order of $O(VE)$ and, since every iteration of the loop is $O(E)$, the total running time of the *Edmonds-Karp algorithm* is $O(VE^2)$.

3.2 Push Relabel

In this section we present an alternative approach to computing the maximum flow: the Push Relabel algorithm, in particular Golberg's **generic Push Relabel algorithm**. As Ford Fulkerson, Push Relabel algorithm also works on the residual graph, but more localized: rather than examining the entire residual network to find an augmenting path, Push Relabel works on one vertex at a time. Moreover, while in Ford-Fulkerson the flow is conserved (except for source and sink), Push Relabel allows preflows, for which inflow can exceed exceed outflow, before reaching the final flow, when the net difference is still 0 (except for source and sink). Moreover, we will see that it is more efficient in terms of time complexity. The intuition behind the Push Relabel algorithm comes from considering a fluid flow problem: we can indeed consider the edges as water pipes and the nodes as joints. The source is considered to be at the highest level and it sends water to all adjacent nodes. Once a node has excess water, it pushes water to a smaller height node. If water gets locally trapped at a vertex, the vertex is relabeled, which means its height is increased. According to this new interpretation, each vertex has then associated to it a height variable and an excess flow. Height is used to determine whether a vertex can push flow to an adjacent or not, as a vertex can push flow only to a smaller height vertex. Excess flow is the difference between the total flow coming into the vertex and the total flow going out of the vertex. As in the Ford Fulkerson algorithm, each edge is also associated to the current flow and to a capacity. There are three main operations on the residual graph in the generic Push Relabel algorithm: Push, Relabel and Initialize preflow.

Push() is used to make the flow move from a node which has excess flow through an edge with positive residual capacity. If there is an adjacent vertex with smaller height (in the residual graph), we push the flow from the vertex to the adjacent with lower height. The amount of pushed flow through the edge is equal to the minimum between the excess flow and the capacity of the edge. Then the excess flows in u and v are updated. In order to make the algorithm more efficient, we have also added the overflowing list: a double linked list that contains all the overflowing vertices of G except for the sink.

Algorithm 2 PUSH(G,u,v)

```
1: // Applies when: $u$ is overflowing, $c_f(u,v) > 0$ and $u.h > v.h$
2: // Action: Push $\Delta_f(u,v) = \min(u.e, c_f(u,v))$ units of flow from $u$ to $v$
3: $\Delta_f(u,v) = \min(u.e, c_f(u,v))$
4: increase the flow of $(u,v)$ by $\Delta_f(u,v)$
5: $u.e = u.e - \Delta_f(u,v)$
6: if $u.e = 0$
7:     remove $u$ from the overflowing list
8: $v.e = v.e + \Delta_f(u,v)$
9: study $v$ and add it to the overflowing list if needed
```

We can see from the pseudocode that all the lines are executed in constant time as they are mainly just assignments. We have also chosen to use double linked lists in order to have the running times of lines 7 (removing from the linked list) and 9 (adding to the linked list) constant. Therefore Push() belongs to $O(1)$.

Relabel() operation is used when a vertex u has excess flow and none of its adjacent (verteces in E_f, i.e. verteces in the residual graph for which there is a positive residual capacity from u to that vertex, so we can add flow) is at lower height: we then increase height of the vertex so that we can perform Push(). The height will be relabeled to be the minimum height of the adjacent vertices in E_f plus 1. We also recall that, by definition, neither the source s nor the sink t can be overflowing and so they are ineligible for relabeling. As we did with the overflowing linked list, we also want to keep track, for each vertex u, of the set $u.Adj_h$ of all the adjacent vertices in the residual graph that have lower height.

Algorithm 3 RELABEL(G,u)

```
1: // **Applies when**: $u$ is overflowing and, for all $v \in V$ such that $(u, v) \in E_f$,
   we have $u.h \leq v.h$
2: // **Action**: increase the height of $u$
3: $u.h = 1 + \min\{v.h : (u, v) \in E_f\}$
4: update $Adj_h$ for all the needed vertices
```

We can see from the pseudocode that the cost of line 3 is $O(V)$ as we are computing the minimum of a set of cardinality at most V. We also have that the cost of line 4 is $O(V)$ as we have used sets to implement Adj_h. The operation Relabel() is then $O(V)$.

Once height and overflow of every vertex are initialized at 0, **Initialize-Preflow()** operation is used to set, for all vertices adjacent to the source s, the flow and excess flow equal to the initial capacity. We set also the height of the source to be equal to the total number of vertices.

Algorithm 4 INITIALIZE-PREFLOW(G)

```
1: $s.h = |G.V|$
2: for each vertex $v \in s.Adj$
3:     $f(s, v) = c(s, v)$
4:     $v.e = c(s, v)$
5:     add $v$ to the overflowing list if it is not the sink
6:     $s.e = s.e - c(s, v)$
```

From the pseudocode, we can see that lines $1, 3, 4$ and 6 are assignments so their cost is constant and, since the overflowing list is a double linked list, also line 5 is constant time. The for cycle in line 2 is performed at most on each vertex, so at most V times. The operation Initialize-Preflow() is then of the order $O(V)$.

Initialization, followed by a sequence of push and relabel operations, executed in no particular order, yields to the generic Push-Relabel. It can be proved that as long as an overflowing vertex exists (i.e. the overflowing list is not empty), at least one between the operations push and relabel can be applied: if we cannot push on some vertex, we can then do the relabel. When no vertex is overflowing anymore, then the preflow on G is a flow and it has maximum value.

Algorithm 5 GENERIC-PUSH-RELABEL(G)

```
1: INITIALIZE-PREFLOW($G, s$)
2: while the linked list with the overflowing vertices is not empty
3:     select an applicable push or relable operation and perform it
```

We have shown before that the cost of line 1 is $O(V)$. Moreover, since we keep track of the overflowing linked list and the Adj_h sets, we can choose which operation to perform in constant time. We know the cost of each Push() or Relabel() operation, thus we just need to know how many times these operations are performed in the while. We will bound separately three types of operations: relabels, saturating pushes and nonsaturating pushes.

Remark 2. The number of relabel operations performed is at most $2V^2$.

An intuition for this can be found by thinking at the structure of the vertices. It can be shown that, for every vertex $u \in V$, during the entire process, $u.h \leq 2V-1$. We note that only the $V-2$ vertices in $V-\{s,t\}$ can be relabeled. Since the relabel operation increases the height at least by 1 and the starting height is 0, each of these vertices is relabeled at most $2V-1$ times. Then the total number of relabel operations is $(2V-1)(V-2) < 2V^2$.

We will now focus on the saturating pushes, that are pushes from u to v where $\Delta_f(u,v) = c_f(u,v)$: after this operation is performed, (u,v) is no longer in E_f.

Remark 3. The number of saturating pushes performed is at most $2VE$.

This can be explained by considering any pair of vertices u and v in V such that at least one edge between (u,v) and (v,u) is in E. Now suppose that a saturating push from u to v has occurred: this means that $v.h = u.h - 1$. Since it is a saturating push, (u,v) is no longer in E_f: next time a saturating push from u to v occurs, first we need a push from v to u and thus $v.h = u.h + 1$. The fact that $u.h$ cannot decrease implies that $v.h$ must increase by at least 2 before another saturating push can happen from u to v. Likewise, $u.h$ must increase of at least 2 between two saturating pushes form v to u. Since the height cannot exceed $2V-1$, the number of times any vertex can have the height increased by 2 is less than V. Since at least one of u and v must be increased by 2 between any two saturating pushes involving u and v, there are less than $2V$ of such saturating pushes. Taking into consideration all the edges, we have that the total number of saturating pushes is less than $2VE$.

Now we consider nonsaturating pushes.

Remark 4. The number of nonsaturating pushes is at most $4V^2(V+E)$.
To show this, we define a potential function $\Phi = \sum_{v:v.e>0} v.h$: its initial value is 0 and its value may change after each basic operation. When we relabel the vertex u, we increase Φ at most by $2V-1 < 2V$. When we perform a saturating push from a vertex u to a vertex v, the increase of Φ is less than $2V$, since the heights cannot change (we are not doing a relabel) and an increase can be brought only by the vertex v, which could have been not overflowing before the push and so its height $v.h$ can contribute to increase Φ of at most $2V-1$.
Now we consider nonsaturating pushes, which decrease Φ at least by 1. Indeed, when we apply the nonsaturating push, u passes from being overflowing to not being it anymore, therefore Φ is decreased by $u.h$. On the other hand, v may or may not have been overflowing. If v was already overflowing, the contribute of $v.h$ to Φ remains the same (we are not performing a relabel), otherwise it is increased by $v.h$. Recalling that $v.h < u.h$ since we are performing a push from u to v, the final decrease of Φ will be at least of $u.h - v.h \geq 1$.
We will use these results to derive an upper bound on the number of nonsaturating pushes at the end of the algorithm. We have indeed that an upper bound for the total increase of Φ is $(2V)\left(2V^2\right) + (2V)(2VE) = 4V^2(V+E)$. Since Φ is initialised at 0 and it cannot be negative, the decrease caused by the nonsaturating pushes is at most $4V^2(V+E)$, which is also the upper bound for the number of the nonsaturating pushes performed.

Let us now compute the total running time of the Push-Relabel algorithm.

$$\begin{aligned} T(V,E) = &\ (\text{cost initialize-preflow}) \\ &+ (\text{cost relabel})(\#\text{ of relabels}) \\ &+ (\text{cost sat-pushes})(\#\text{ of sat-pushes}) \\ &+ (\text{cost non-sat-pushes})(\#\text{ of non-sat-pushes}) \\ &\leq O(V) + O(V) * 2V^2 + O(1) * 2VE + O(1) * 4V^2(V+E) \\ &= O(V^2E). \end{aligned}$$

3.3 Memory cost and comparison

The implementation of both algorithms requires the same asymptotic space, that is $O(V+E)$. This is due to the fact that both algorithms take as input the data structure of graphs implemented with their adjacency sets. However, since for the Push Relabel algorithm the data structures keep track of more information (e.g. the class Vertex stores height, excess flow and the set Adj_h), we could say that for small size inputs it is more convenient space-wise to use Ford Fulkerson. We notice, though, that this additional information does not change the asymptotic spatial complexity, so for large outputs we cannot prefer one algorithm to another in terms of space. Then the substantial difference between them is between their running times: since Generic Push Relabel algorithm is $O(V^2E)$, it is asymptotically better than Edmonds-Karp algorithm, which is $O(VE^2)$.

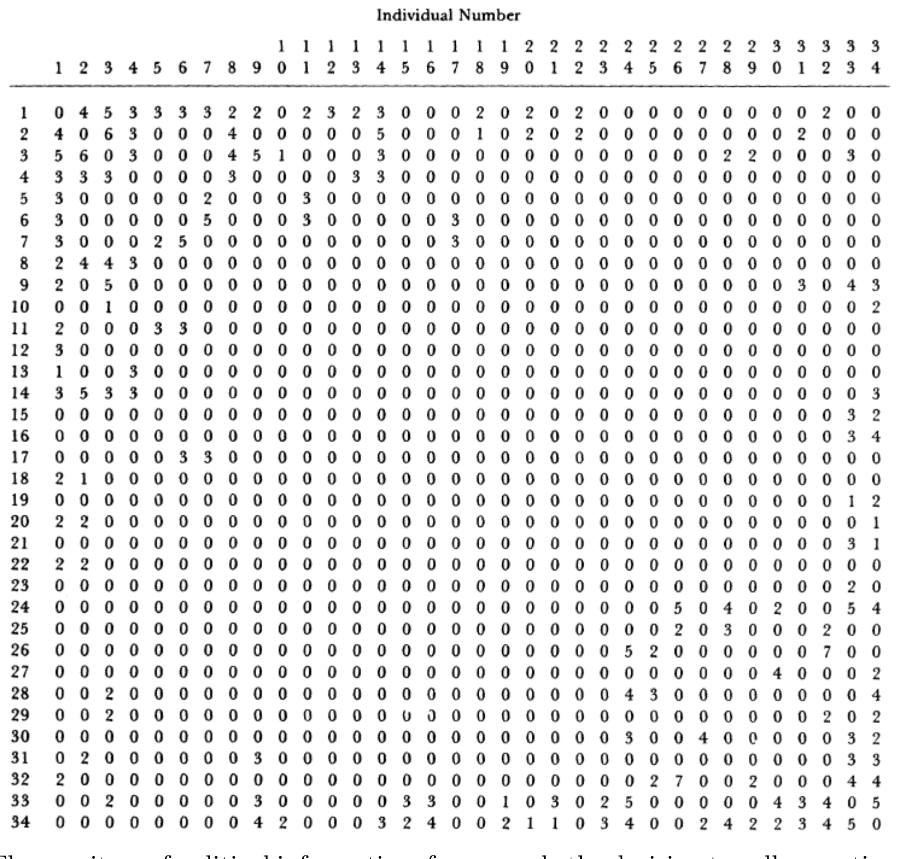
4 Application to Zachary's karate club

Now that we have the tools, we can define more formally the model. The first thing we have to do is to formalize the relationships among the components of the karate club in order to define the network of friendships among the club members. Our aim is to construct a formal model of the friendship network which can represent the political activities and the fission process.
In order to do so, it is necessary to observe how the members interact outside the normal activities of the club, that are karate classes and club meetings. Even if the club membership was around 60 at that time, we consider only 34 individuals: they were the only ones who interacted with other club members outside the context of meetings and classes and carried on the study of karate after the political conflict.
When we build the model of friendship structure within the club, we must keep in mind that the political process operated through the transmission of political information (both tactical, for example when a meeting would occur, and ideological). This process brought the factions into existence during crisis periods: crises caused indeed the selective reinforcement of only those relationships within people who shared ideologies, thus making the factions more defined.
From the point of view of information flow, the ties in the network can be thought of as channels across which information may flow. The absence of an edge precludes any direct passage of political information between the two individuals and it is obvious that the flow is not identical across all the edges in the network. In order to represent the network accurately as a net of channels for information flow, the different relationships, and therefore edges, must be quantified and included in the model.
We can create a *capacity matrix* which quantifies the relationships of the existing edges in the network: each value of the capacity matrix represents the maximum possible flow for the corresponding edge. Unfortunately, there is no way known to quantify what has been referred to as political information and, therefore, the capacity matrix. However we can make certain assumptions about the social settings in which the information was communicated and get a possible capacity matrix. Political information was indeed communicated in contexts outside the regular activities of the club and, since political activity was not recognized by the club members, the transmission of political information was incidental to normal social interaction. Therefore, it is reasonable to think that the amount of information transmitted was proportional to the number of contexts in which the information might have been communicated: the greater the number of contexts where a pair interacted, the more the information that could be passed. The relationships between each pair of individual is then quantified by a value which indicates in how many of the following 8 contexts the two individuals interacted:

- Association in and between academic classes at the university.
- Membership in Mr. Hi's private karate studio on the east side of the city where Mr. Hi taught as a part-time instructor at night.

- Membership in Mr. Hi's private karate studio on the east side of the city, where many of his supporters trained on weekends.
- Students who taught at the private karate studio.
- Interaction at the university rathskeller, located in the same basement as the karate club's workout area.
- Interaction at a student-oriented bar located across the street from the university campus.
- Attendance at open karate tournaments held through the area at private karate studios.
- Attendance at intercollegiate karate tournaments held at local universities. Since both open and intercollegiate tournaments were held on Saturdays, attendance at both was impossible.

By taking into account this, we have the following capacity matrix.



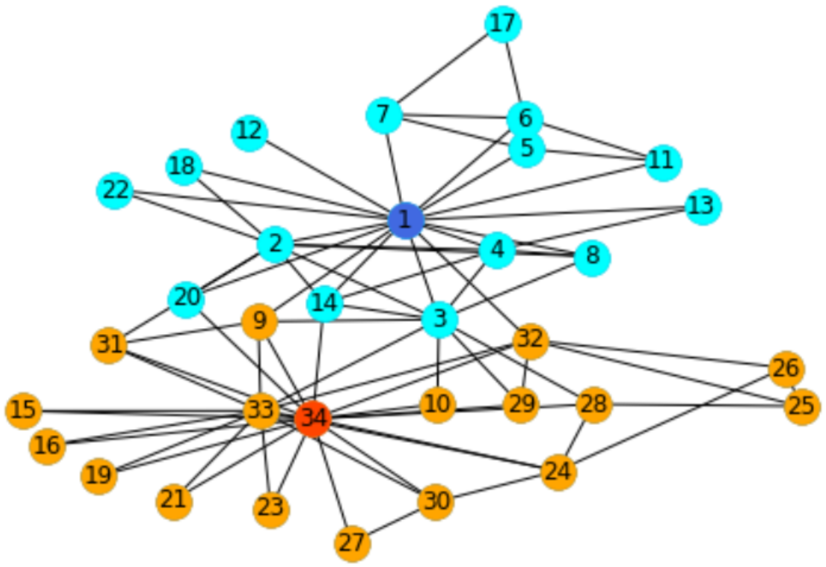
Individual Number

										1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2	2	3	3	3	3	3
	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4
1	0	4	5	3	3	3	3	2	2	0	2	3	2	3	0	0	0	2	0	2	0	2	0	0	0	0	0	0	0	0	0	2	0	0
2	4	0	6	3	0	0	0	4	0	0	0	0	0	5	0	0	0	1	0	2	0	2	0	0	0	0	0	0	0	0	2	0	0	0
3	5	6	0	3	0	0	0	4	5	1	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	2	2	0	0	0	3	0
4	3	3	3	0	0	0	0	3	0	0	0	0	3	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	3	0	0	0	0	0	2	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	3	0	0	0	0	0	5	0	0	0	3	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	3	0	0	0	2	5	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	2	4	4	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	2	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	4	3
10	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
11	2	0	0	0	3	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	1	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	3	5	3	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	2
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	4
17	0	0	0	0	0	3	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2
20	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	1
22	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	4	0	2	0	0	5	4
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	3	0	0	0	2	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	2	0	0	0	0	0	0	7	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	2
28	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	3	0	0	0	0	0	0	0	0	4
29	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	2
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	4	0	0	0	0	0	3	2
31	0	2	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	3
32	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	7	0	0	2	0	0	0	4	4
33	0	0	2	0	0	0	0	0	3	0	0	0	0	0	3	3	0	0	1	0	3	0	2	5	0	0	0	0	0	4	3	4	0	5
34	0	0	0	0	0	0	0	0	4	2	0	0	0	3	2	4	0	0	2	1	1	0	3	4	0	0	2	4	2	2	3	4	5	0

Then an item of political information, for example the decision to call a meeting, originated with one or several individuals was spread through the club accordingly to this matrix.

The information (the calling of a meeting) usually began with one of the factional leaders, Mr. Hi or John A. This individual will be called the source of the information, or simply the *source*. Because of the political conflict in the club, it would be advantageous to the source if individuals in the opposite faction did not receive the information: the more closely a person was aligned with the other side, the less the source would benefit from their knowing about the meeting. By this reasoning, the leader of the other faction should be the last to receive the information. He will therefore be called the sink of the information, or simply the *sink*. However, not all individuals in the network were solidly members of a faction. Some vacillated indeed between the two ideological positions, while others were simply satisfied not to take sides. These individuals are key nodes in the network as it was through them that information was likely to pass from one faction to the other. It is then reasonable to think that there is a bottleneck in the network, that is a structural limitation on information flow between factions from the source to the sink. We can then use this to predict the break that occurred in the club at the time of the fission. This is indeed equivalent to finding the minimum cut, which will separate the club members who joined Mr. Hi's club after the split from those who joined the officers' club.

We first apply the Ford Fulkerson algorithm. We have intended Mr. Hi as the first vertex and the source, whereas John A. is the last node (34) and the sink. By running the algorithm, we get two groups within the network: in light blue we have coloured the set of the cut which includes the source, whereas in orange the others with the sink.



The algorithm predicted in a pretty accurate way how the club split, except individual number 9. The members on the source side of the cut belonged to Mr. Hi's faction and joined the club formed by his supporters after the split.

On the other hand, individuals on the sink side of the cut belonged to John A.'s faction and joined the club founded by the officers. Person number 9 was a weak supporter of John A., but still joined Mr. Hi's club after the split. This can be explained by noting that he was only three weeks away from a test for black belt when the split occurred. If he had joined the officers' club, he would have had to give up his rank and begin again with a white belt as the officers had decided to change the style of karate practiced. Having four years of study invested in the style of Mr. Hi, member 9 could not bring himself to repudiate his rank and start again. However, his/her ideological affiliation was well predicted by the algorithm. If we apply the Push-Relabel algorithm, we notice that we obtain the same result. The difference is just in the computational time. Once again, the member 9 was mispredicted, but we can say that this is due to a bias embedded in the model: the prediction is indeed correct if we consider the member 9 was ideologically standing with the faction of John A.

It can be concluded that, in the friendship networks, club members communicated not only political information about the meetings, but also their perceptions and understandings of the nature of the club in conscious and unconscious ways. Then the bottle-necking of the network tended to allow communication to a greater extent within the factions than across the factional boundary. The strategy of the factions, whose membership was based on ideology, acted to strengthen the cut in the network, which itself was the organizational basis for the factions' existence. It indeed acted to intensify the ideological divisions on which club members based their factional affiliation. This created a vicious cycle which ensured the fission of the club. While the content of this mechanism was specific to the karate club, its form is a general feature of communication patterns in small groups, that is selective reinforcement. This conclusion has broad implications for the anthropological study of small group. The minimum cut in a capacitaded network can indeed be seen as the result of a positive feedback process in a small group: whenever a unique minimum cut exists in a capacitated network representation of a small group, it will act as a barrier to group and under certain conditions, will act as a selective factor for fission.

References

[1] Cormen, Thomas H., Leiserson, Charles E., Rivest, Ronald L. (2009). *Introduction to Algorithms*, Chapter 26. MIT Press Ltd.

[2] Mestre, Julian (2009). *Optimization II*, Lecture 3. Max Planck Institut Informatik.

[3] Roughgarden, Tim (2016). *A Second Course in Algorithms*, Lecture 3. Columbia University.

[4] Zachary, Wayne W. (1977). *An Information Flow Model for Conflict and Fission in Small Groups.* The University of Chicago Press.