Let $A \in \mathcal{V}$ source, $B \in \mathcal{V}$ sink. For the following, we suppose that $Turn_{j-1}$ has just finished and $A = Player(j)$ is currently deciding $Turn_j$. We use the following notation:

$$\begin{aligned} c_{Av} &= DTr_{A \to v, j-1} \\ c'_{Av} &= DTr_{A \to v, j} \end{aligned}$$

Moreover, X and X' will be the flows returned by some execution of $MaxFlow_{\mathcal{G}_{j-1}}(A, B)$ and $MaxFlow_{\mathcal{G}_j}(A, B)$ respectively.

Furthermore, we suppose an arbitrary ordering of the members of $N^+(A)$. We set $n = |N^+(A)|$. Thus

$$N^+(A) = \{v_1, ..., v_n\}$$

We use these subscripts to refer to the respective capacities (a.k.a. direct trusts) and flows. Thus

$$\left.\begin{aligned} x_i &= x_{Av_i} \ , \\ c_i &= c_{Av_i} \ , \end{aligned}\right. \text{ where } i \in [n]$$

Definition 1 (Trust Reduction).

Trust Reduction on neighbour i is defined as $\delta_i = c_i - c'_i$.

Flow Reduction on neigbour i is defined as $\Delta_i = x_i - c'_i$.

We will also use the standard notation for 1-norm and ∞*-norm:*

$$\begin{aligned} ||\delta_i||_1 &= \sum_{i=1}^{n} \delta_i \\ ||\delta_i||_\infty &= \max_{1 \leq i \leq n} \delta_i \end{aligned}$$

Definition 2 (Restricted Flow).

Let $i \in [n]$. *Let* $F_{A_i \to B}$ *be* x'_i *when:*

$$\begin{gathered} c'_i = c_i \ \textit{and} \\ \forall k \in [n] \setminus \{i\}, c'_k = 0 \ . \end{gathered}$$

This definition can be rephrased equivalently as follows:
Let $v \in N^+(A)$. *Let* $F_{A_v \to B}$ *be* x'_{Av} *when:*

$$\begin{gathered} c'_{Av} = c_{Av} \ \textit{and} \\ \forall w \in N^+(A) \setminus \{v\}, c'_{Aw} = 0 \ . \end{gathered}$$

Let $L \subset [n]$. *Let* $F_{A_L \to B}$ *be* $\sum_{i \in L} x'_i$ *when:*

$$\forall i \in L, c'_i = c_i \text{ and}$$
$$\forall i \in [n] \setminus L, c'_i = 0 \ .$$

The latter definition can be rephrased equivalently as follows:
Let $S \subset N^+(A)$. *Let* $F_{A_S \to B}$ *be* $\sum_{v \in S} x'_{Av}$ *when:*

$$\forall v \in S, c'_{Av} = c_{Av} \text{ and}$$
$$\forall v \in N^+(A) \setminus S, c'_{Av} = 0 \ .$$

The choice of the definition will depend on whether K *in* $F_{A_K \to B}$ *is a node, an index or a set of nodes or indices.*

Theorem 1 (Saturation theorem).

$$(\forall i \in [n], c'_i \leq x_i) \Rightarrow (\forall i \in [n], x'_i = c'_i)$$

Proof. From the flow definition we know that

$$\forall i \in [n], x'_i \leq c'_i \ . \tag{1}$$

In turn $j - 1$, there exists some valid flow Y such that

$$\forall i \in [n], y_i = c'_i$$

with a flow value $\sum_{i=1}^{n} y_i$, which can be created as follows: We start from X and for each (A, v_i) edge we reduce the flow along paths starting from this edge for a total reduction of $x_i - c'_i$ on all those paths. Y is also obviously valid for turn j and, since all capacities c'_i are saturated, there can be no more outgoing flow from the source, thus Y is a maximum flow in $\mathcal{G}_j$. □

Theorem 2 (Trust transfer theorem (flow terminology)).
Let A *source,* B *sink. We create a new graph where*

$$\forall i \in [n], c'_i \leq x_i \text{ and}$$
$$\sum_{i=1}^{n} c'_i = F - V \ .$$

It then holds that $maxFlow_{\mathcal{G}_j}(A, B) = F' = F - V$.

Proof. From theorem 1 we can see that $x'_i = c'_i$. It holds that

$$F' = \sum_{i=1}^{n} x'_i = \sum_{i=1}^{n} c'_i = F - V \quad .$$

$\square$

Lemma 1 (Flow limit lemma).

$$\forall i \in [n], x_i \leq F_{A_i \to B}$$

Proof. Suppose a flow where $\exists i \in [n] : x_i > F_{A_i \to B}$. If for any $k \neq i$ we choose $c'_k < c_k$, then $x'_i \geq x_i$. We set the new capacities as follows:

$$\begin{aligned} \forall k \neq i, c'_k &= 0 \text{ and} \\ c'_i &= c_i \quad . \end{aligned}$$

Then for X' we will have

$$\begin{aligned} \forall k \neq i, x'_k &= 0 \text{ and} \\ x'_i &= x_i \quad , \end{aligned}$$

which is also a valid flow for $\mathcal{G}_{j-1}$ and thus by definition

$$F_{A_i \to B} = x'_i = x_i > F_{A_i \to B} \quad ,$$

which is a contradiction. Thus the proposition holds. $\square$

Theorem 3 (Trust Saving Theorem).
Suppose some $i \in [n]$ and two alternative capacities configurations, say C'_1 and C'_2 such that

$$\begin{aligned} c'_{1,i} &= F_{A_i \to B} \quad , \\ c'_{2,i} &= c_i \quad , \\ \forall k \in [n] \setminus \{i\}, c'_{1,k} &= c'_{2,k} \quad . \end{aligned}$$

Then $maxFlow_1 = maxFlow_2$.

Proof. From the Flow Limit lemma (1) we know that $x_i \leq F_{A_i \to B}$, thus we can see that any increase in c'_i beyond $F_{A_i \to B}$ will not influence x_i and subsequently will not incur any change on the rest of the flows. $\square$

Theorem 4 (Invariable trust reduction with naive algorithms).
If $\forall i \in [n], c'_i \leq x_i$, then $||\delta_i||_1$ and $||\Delta_i||_1$ are independent of x'_i, c'_i.

Proof. Since $\forall i \in [n], c'_i \leq x_i$, by applying the Saturation theorem (1) we see that $x'_i = c'_i$, thus $\delta_i = c_i - x'_i$ and $\Delta_i = x_i - x'_i$. We know that $\sum_{i=1}^{n} x'_i = F - V$, so we have

$$||\delta_i||_1 = \sum_{i=1}^{n} \delta_i = \sum_{i=1}^{n} (c_i - x'_i) = \sum_{i=1}^{n} c_i - F + V \text{ and}$$
$$||\Delta_i||_1 = \sum_{i=1}^{n} \Delta_i = \sum_{i=1}^{n} (x_i - x'_i) = \sum_{i=1}^{n} x_i - F + V \quad .$$

thus $||\delta_i||_1, ||\Delta_i||_1$ are independent from x'_i and c'_i. □

Until now *MaxFlow* has been viewed purely as an algorithm. This algorithm is not guaranteed to always return the same flow when executed muliple times on the same graph. However, the corresponding flow value, *maxFlow*, is always the same. Thus *maxFlow* can be also viewed as a function from a matrix of capacities to a positive real number. Under this perspective, we prove the following theorem. Let $\mathcal{C}$ be the family of all capacity matrices $C = [c_{vw}]_{V(\mathcal{G}) \times V(\mathcal{G})}$.

Theorem 5 (maxFlow continuity).
Let $p \in \mathbb{N} \cup \{\infty\}$. The function $maxFlow : \mathcal{C} \to \mathbb{R}^+$ is continuous with respect to the $|| \cdot ||_p$ norm.

Proof. Let $C_0 \in \mathcal{C}$. We want to prove that

$$\forall \epsilon > 0, \exists \delta > 0 : 0 < ||C - C_0||_p < \delta \Rightarrow |maxFlow\,(C) - maxFlow\,(C_0)\,| < \epsilon \quad .$$

We will prove it by contradiction. Suppose that

$$\exists \epsilon > 0 : \forall \delta > 0, 0 < ||C - C_0||_p < \delta \Rightarrow |maxFlow\,(C) - maxFlow\,(C_0)\,| \geq \epsilon \quad .$$

Let $v_1, u_1 \in V\,(\mathcal{G})$. Let C such that

$$c_{v_1 u_1} = c_{0,v_1 u_1} + \frac{\epsilon}{2}$$
$$\forall\,(v,u) \in E\,(\mathcal{G}) \setminus \{(v_1,u_1)\}, c_{vu} = c_{0,vu} \quad .$$

Due to the construction, for $\delta = \epsilon$ we have

$$0 < ||C - C_0||_p < \delta \quad . \tag{2}$$

Any valid flow for C_0 is also valid for C, thus

$$maxFlow\,(C_0) \leq maxFlow\,(C) \quad . \tag{3}$$

Also, it is obvious by the way that C was constructed that

$$maxFlow\left(C\right) \leq maxFlow\left(C_0\right) + \frac{\epsilon}{2} \tag{4}$$

From (3) we have $maxFlow\left(C_0\right) \leq maxFlow\left(C\right) + \frac{\epsilon}{2}$, which, in combination with (4), gives

$$\left|maxFlow\left(C\right) - maxFlow\left(C_0\right)\right| \leq \frac{\epsilon}{2} < \epsilon \ ,$$

which, together with (2) contradicts our supposition. Thus $maxFlow$ is continuous on C_0. Since C_0 is arbitrary, the result holds for all $C_0 \in \mathcal{C}$, thus $maxFlow$ is continuous with respect to $||\cdot||_p$ for any $p \in \mathbb{N} \cup \{\infty\}$. $\square$

Here we show three naive algorithms for calculating new direct trusts so as to maintain invariable risk when paying a trusted party. Let $F = \sum\limits_{i=1}^{n} x_i$. To prove the correctness of the algorithms, it suffices to prove that

$$\forall i \in [n], c'_i \leq x_i \text{ and} \tag{5}$$

$$\sum_{i=1}^{n} c'_i = F - V \ . \tag{6}$$

```
First Come First Served Trust Transfer
Input : old flows $x_i$, value V
Output : new capacities $c'_i$
fcfs(($x_i$), V) :
  n = length($x_i$)
  F = $\sum_{i=1}^{n} x_i$
  if (F < V)
    return(⊥)
  F$_{cur}$ = F
  for (i = 1 to n)
    $c'_i$ = $x_i$
  i = 1
  while (F$_{cur}$ > F - V)
    reduce = min($x_i$, F$_{cur}$ - (F - V))
    F$_{cur}$ = F$_{cur}$ - reduce
    $c'_i$ = $x_i$ - reduce
    i += 1
```

```
15    U′ = ⋃_{i=1}^{n} {c′_i}
16    return(U′)
```

Proof of correctness for fcfs.

- We will show that $\forall i \in [n]\ u'_i \leq x_i$.
  Let $i \in [n]$. In line 8 we can see that $u'_i = x_i$ and the only other occurence of u'_i is in line 13 where it is never increased ($\mathit{reduce} \geq 0$), thus we see that, when returned, $u'_i \leq x_i$.
- We will show that $\sum\limits_{i=1}^{n} u'_i = F - V$.
  $F_{cur,0} = F$
  If $F_{cur,i} \leq F - V$, then $F_{cur,i+1}$ does not exist because the *while* loop breaks after calculating $F_{cur,i}$.
  Else $F_{cur,i+1} = F_{cur,i} - \min\left(x_{i+1}, F_{cur,i} - F + V\right)$.
  If for some $i, \min\left(x_{i+1}, F_{cur,i} - F + V\right) = F_{cur,i} - F + V$, then $F_{cur,i+1} = F - V$, so if $F_{cur,i+1}$ exists, then $\forall k < i, F_{cur,k} = F_{cur,k-1} - x_k \Rightarrow F_{cur,i} = F - \sum\limits_{k=1}^{i} x_k$
  Furthermore, if $F_{cur,i+1} = F - V$ then $u'_{i+1} = x_{i+1} - F_{cur,i} + F - V = x_i - F + \sum\limits_{k=1}^{i-1} x_k + F - V = \sum\limits_{k=1}^{i} x_k - V$, $\forall k \leq i, u'_k = 0$ and $\forall k > i+1, u'_k = x_k$.
  In total, we have $\sum\limits_{k=1}^{n} u'_k = \sum\limits_{k=1}^{i} x_k - V + \sum\limits_{k=i+1}^{n} x_k = \sum\limits_{k=1}^{n} x_k - V \Rightarrow \sum\limits_{k=1}^{n} u'_k = F - V$.

□

Complexity of algorithm . First we will prove that on line **??** $i \leq n+1$. Suppose that $i > n+1$ on line **??**. This means that $F_{cur,n}$ exists and $F_{cur,n} = F - \sum\limits_{i=1}^{n} x_i = 0 \leq F - V$ since, according to the condition on line **??**, $F - V \geq 0$. This means however that the *while* loop on line **??** will break, thus $F_{cur,n+1}$ cannot exist and $i = n+1$ on line **??**, which is a contradiction, thus $i \leq n+1$ on line **??**. Since i is incremented by 1 on every iteration of the *while* loop (line **??**), the complexity of the *while* loop is $O(n)$ in the worst case. The complexity of lines **??** - **??** and **??** is $O(1)$ and the complexity of lines **??**, **??** - **??** and **??** is $O(n)$, thus the total complexity of algorithm is $O(n)$. □

Algorithm 1: Absolute equality trust transfer ($||\Delta_i||_\infty$ minimizer)

Input : x_i flows, $n = |N^+(s)|$, V value
Output: u_i' capacities

1 $F \leftarrow \sum_{i=1}^{n} x_i$
2 **if** $F < V$ **then**
3   **return** $\perp$
4 **for** $i \leftarrow 1$ *to* n **do**
5     $u_i' \leftarrow x_i$
6 $reduce \leftarrow \frac{V}{n}$
7 $reduction \leftarrow 0$
8 $empty \leftarrow 0$
9 $i \leftarrow 0$
10 **while** $reduction < V$ **do**
11   **if** $u_i' > 0$ **then**
12   **if** $x_i < reduce$ **then**
13             $empty \leftarrow empty + 1$
14   **if** $empty < n$ **then**
15                 $reduce \leftarrow reduce + \frac{reduce - x_i}{n - empty}$
16             $reduction \leftarrow reduction + u_i'$
17             $u_i' \leftarrow 0$
18   **else if** $x_i \geq reduce$ **then**
19             $reduction \leftarrow reduction + u_i' - (x_i - reduce)$
20             $u_i' \leftarrow x_i - reduce$
21     $i \leftarrow (i+1) mod\, n$
22 **return** $U' = \bigcup_{k=1}^{n} \{u_k'\}$

We will start by showing some results useful for the following proofs. Let j be the number of iterations of the **while** loop for the rest of the proofs for algorithm 1 (think of i from line 21 without the $mod\, n$).
First we will show that $empty \leq n$. $empty$ is only modified on line 13 where it is incremented by 1. This happens only when $u_i' > 0$ (line 11), which is assigned the value 0 on line 17. We can see that the incrementation of $empty$ can happen at most n times because $|U'| = n$. Since $empty_0 = 0$, $empty \leq n$ at all times of the execution.
Next we will derive the recursive formulas for the various variables.
$empty_0 = 0$

$$empty_{j+1} = \begin{cases} empty_j, & u'_{(j+1)\ mod\ n} = 0 \\ empty_j + 1, & u'_{(j+1)\ mod\ n} > 0 \ \wedge \ x_{(j+1)\ mod\ n} < reduce_j \\ empty_j, & u'_{(j+1)\ mod\ n} > 0 \ \wedge \ x_{(j+1)\ mod\ n} \geq reduce_j \end{cases}$$

$$reduce_0 = \frac{V}{n}$$

$$reduce_{j+1} = \begin{cases} reduce_j, & u'_{(j+1)\ mod\ n} = 0 \\ reduce_j + \frac{reduce_j - x_{(j+1)\ mod\ n}}{n - empty_{j+1}}, & u'_{(j+1)\ mod\ n} > 0 \ \wedge \ x_{(j+1)\ mod\ n} < reduce_j \\ reduce_j, & u'_{(j+1)\ mod\ n} > 0 \ \wedge \ x_{(j+1)\ mod\ n} \geq reduce_j \end{cases}$$

$$reduction_0 = 0$$

$$reduction_{j+1} = \begin{cases} reduction_j, & u'_{(j+1)\ mod\ n} = 0 \\ reduction_j + u'_{(j+1)\ mod\ n}, & u'_{(j+1)\ mod\ n} > 0 \ \wedge \ x_{(j+1)\ mod} \\ reduction_j + u'_{(j+1)\ mod\ n} - x_{(j+1)\ mod\ n} + reduce_{j+1}, & u'_{(j+1)\ mod\ n} > 0 \ \wedge \ x_{(j+1)\ mod} \end{cases}$$

In the end, $r = reduce$ is such that $r = \frac{V - \sum\limits_{x \in S} x}{n - |S|}$ where $S = \{$flows y from s to $N^+(s)$ according to max
$y < r\}$. Also, $\sum\limits_{i=1}^{n} u'_i = \sum\limits_{i=1}^{n} \max(0, x_i - r)$. TOPROVE

Proof of correctness for algorithm 1. – We will show that $\forall i \in [n]\ u'_i \leq x_i$.

On line 5, $\forall i \in [n]\ u'_i = x_i$. Subsequently u'_i is modified on line 17, where it becomes equal to 0 and on line 20, where it is assigned $x_i - reduce$. It holds that $x_i - reduce \leq x_i$ because initially $reduce = \frac{V}{n} \geq 0$ and subsequently $reduce$ is modified only on line 15 where it is increased ($n > empty$ because of line 14 and $reduce > x_i$ because of line 12, thus $\frac{reduce - x_i}{n - empty} > 0$). We see that $\forall i \in [n], u'_i \leq x_i$.

– We will show that $\sum\limits_{i=1}^{n} u'_i = F - V$.

The variable *reduction* keeps track of the total reduction that has happened and breaks the **while** loop when $reduction \geq V$. We will first show that $reduction = \sum\limits_{i=1}^{n}(x_i - u'_i)$ at all times and then we will prove that $reduction = V$ at the end of the execution. Thus we will have proven that $\sum\limits_{i=1}^{n} u'_i = \sum\limits_{i=1}^{n} x_i - V = F - V$.

- On line 5, $u'_i = x_i \Rightarrow \sum\limits_{i=1}^{n}(x_i - u'_i) = 0$ and $reduction = 0$.

  On line 17, u'_i is reduced to 0 thus $\sum\limits_{i=1}^{n}(x_i - u'_i)$ is increased by u'_i. Similarly, on line 16 *reduction* is increased by u'_i, the same as the increase in $\sum_{i=1}^{n}(x_i - u'_i)$.

On line 20, u'_i is reduced by $u'_i - x_i + reduce$ thus $\sum_{i=1}^{n}(x_i - u'_i)$ is increased by $u'_i - x_i + reduce$. On line 19, *reduction* is increased by $u'_i - x_i + reduce$, which is equal to the increase in $\sum_{i=1}^{n}(x_i - u'_i)$. We also have to note that neither u'_i nor *reduction* is modified in any other way from line 10 and on, thus we conclude that $reduction = \sum_{i=1}^{n}(x_i - u'_i)$ at all times.

- Suppose that $reduction_j > V$ on the line 22. Since $reduction_j$ exists, $reduction_{j-1} < V$. If $x_{j\ mod\ n} < reduce_{j-1}$ then $reduction_j = reduction_{j-1} + u'_{j\ mod\ n}$. Since $reduction_j > V$, $u'_{j\ mod\ n} > V - reduction_{j-1}$. TOCOMPLETE

□

Complexity of algorithm 1. In the worst case scenario, each time we iterate over all capacities only the last non-zero capacity will become zero and every non-zero capacity must be recalculated. This means that every n steps exactly 1 capacity becomes zero and eventually all capacities (maybe except for one) become zero. Thus we need $O(n^2)$ steps in the worst case. □

A variation of this algorithm using a Fibonacci heap with complexity $O(n)$ can be created, but that is part of further research.

Proof that algorithm 1 minimizes the $||\Delta_i||_\infty$ norm. Suppose that U' is the result of an execution of algorithm 1 that does not minimize the $||\Delta_i||_\infty$ norm. Suppose that W is a valid solution that minimizes the $||\Delta_i||_\infty$ norm. Let δ be the minimum value of this norm. There exists $i \in [n]$ such that $x_i - w_i = \delta$ and $u'_i < w_i$. Because both U' and W are valid solutions ($\sum_{i=1}^{n} u'_i = \sum_{i=1}^{n} w_i = F - V$), there must exist a set $S \subset U'$ such that $\forall u'_j \in S, u'_j > w_j$ TOCOMPLETE. □

Algorithm 2: Proportional equality trust transfer

Input : x_i flows, $n = |N^+(s)|$, V value

Output: u'_i capacities

1 $F \leftarrow \sum_{i=1}^{n} x_i$

2 **if** $F < V$ **then**

3   **return** $\perp$

4 **for** $i \leftarrow 1$ *to* n **do**

5     $u'_i \leftarrow x_i - \frac{V}{F} x_i$

6 **return** $U' = \bigcup_{k=1}^{n} \{u'_k\}$

Proof of correctness for algorithm 2. − We will show that $\forall i \in [n]\ u'_i \leq x_i$.

According to line 5, which is the only line where u'_i is changed, $u'_i = x_i - \frac{V}{F} x_i \leq x_i$ since $x_i, V, F > 0$ and $V \leq F$.

− We will show that $\sum_{i=1}^{n} u'_i = F - V$.

With $F = \sum_{i=1}^{n} x_i$, on line 6 it holds that $\sum_{i=1}^{n} u'_i = \sum_{i=1}^{n} (x_i - \frac{V}{F} x_i) = \sum_{i=1}^{n} x_i - \frac{V}{F} \sum_{i=1}^{n} x_i = F - V$.

□

Complexity of algorithm 2. The complexity of lines 1, 4 - 5 and 6 is $O(n)$ and the complexity of lines 2 - 3 is $O(1)$, thus the total complexity of algorithm 2 is $O(n)$. □

Naive algorithms result in $u'_i \leq x_i$, thus according to 4, $||\delta_i||_1$ is invariable for any of the possible solutions U', which is not necessarily the minimum (usually it will be the maximum). The following algorithms concentrate on minimizing two δ_i norms, $||\delta_i||_\infty$ and $||\delta_i||_1$.

Algorithm 3: $||\delta_i||_\infty$ minimizer

Input : $X = \{x_i\}$ flows, $n = |N^+(s)|$, V value, ϵ_1, ϵ_2
Output: u'_i capacities

if $\epsilon_1 < 0 \vee \epsilon_2 < 0$ **then**
  return $\perp$
$F \leftarrow \sum_{i=1}^{n} x_i$
if $F < V$ **then**
  return $\perp$
$\delta_{max} \leftarrow \max_{i \in [n]}\{u_i\}$
$\delta^* \leftarrow$ `BinSearch(`$0, \delta_{max}, F\text{-}V, n, X, \epsilon_1, \epsilon_2$`)`
for $i \leftarrow 1$ *to* n **do**
  $u'_i \leftarrow \max(u_i - \delta^*, 0)$
return $U' = \bigcup_{k=1}^{n} \{u'_k\}$

Since trust should be considered as a continuous unit and binary search dissects the possible interval for the solution on each recursive call, inclusion of the ϵ-parameters in `BinSearch` is necessary for the algorithm to complete in a finite number of steps.

Algorithm 4: *

Input : bot, top, F', n, X, ϵ_1, ϵ_2
Output: δ^*

function `BinSearch` **if** $bot = top$ **then**
  return bot
else
  for $i \leftarrow 1$ *to* n **do**
    $u'_i \leftarrow \max(0, u_i - \frac{top+bot}{2})$
  if $maxFlow < F' - \epsilon_1$ **then**
    return `BinSearch(`$bot, \frac{top+bot}{2}, F', n, X, \epsilon_1, \epsilon_2$`)`
  else if $maxFlow > F' + \epsilon_2$ **then**
    return `BinSearch(`$\frac{top+bot}{2}, top, F', n, X.\epsilon_1, \epsilon_2$`)`
  else
    return $\frac{top+bot}{2}$

Proof that $maxFlow(\delta)$ is strictly decreasing for $\delta : maxflow(\delta) < F$. Let $maxFlow(\delta)$ be the $maxFlow$ with $\forall i \in [n], u'_i = max(0, u_i - \delta)$. We will prove that the function $maxFlow(\delta)$ is strictly decreasing for all

$\delta \leq \max_{i\in[n]}\{u_i\}$ such that $maxFlow(\delta) < F$.
Supppose that $\exists \delta_1, \delta_2 : \delta_1 < \delta_2 \wedge maxFlow(\delta_1) \leq maxFlow(\delta_2) < F$. We will work with configurations of $x'_{i,j}$ such that $x'_{i,j} \leq x_i, j \in \{1, 2\}$.
Let $S_j = \{i \in N^+(s) : i \in MinCut_j\}$. It holds that $S_1 \neq \emptyset$ because otherwise $MinCut_1 = MinCut_{\delta=0}$ which is a contradiction because then $maxFlow(\delta_1) = F$. Moreover, it holds that $S_1 \subseteq S_2$, since $\forall u'_{i,2} > 0, u'_{i,2} < u'_{i,1}$. Every node in the $MinCut_j$ is saturated, thus $\forall i \in S_1, x'_{i,j} = u'_{i,j}$. Thus $\sum_{i\in S_1} x_{i,2} < \sum_{i\in S_1} x_{i,1}$ and, since $maxFlow(\delta_1) \leq maxFlow(\delta_2)$, we conclude that for the same configurations, $\sum_{i\in N^+(s)\setminus S_1} x_{i,2} > \sum_{i\in N^+(s)\setminus S_1} x_{i,1}$.
However, since $x'_{i,j} \leq x_i, j \in \{1, 2\}$, the configuration $[x''_{i,1} = x'_{i,2}, i \in N^+(s) \setminus S_1], [x''_{i,1} = x'_{i,1}, i \in S_1]$ is valid for $\delta = \delta_1$ and then $\sum_{i\in S_1} x''_{i,1} + \sum_{i\in N^+(s)\setminus S_1} x''_{i,1} = \sum_{i\in S_1} x'_{i,1} + \sum_{i\in N^+(s)\setminus S_1} x'_{i,2} > maxFlow(\delta_1)$, contradiction.
Thus $maxFlow(\delta)$ is strictly decreasing. □

We can see that if $V > 0, F' = F - V < F$ thus if $\delta \in (0, \max_{i\in[n]}\{u_i\}]$: $maxFlow(\delta) = F' \Rightarrow \delta = \min ||\delta_i||_\infty : maxFlow(||\delta_i||_\infty) = F'$.

Proof of correctness for function 4. Supposing that $[F' - \epsilon_1, F' + \epsilon_2] \subset [maxFlow(top), maxFlow(bot)]$, or equivalently $maxFlow(top) \leq F' - \epsilon_1 \wedge maxFlow(bot) \geq F' + \epsilon_2$, we will prove that $maxFlow(\delta^*) \in [F' - \epsilon_1, F' + \epsilon_2]$.
First of all, we should note that if an invocation of `BinSearch` returns without calling `BinSearch` again (line 2 or 11), its return value will be equal to the return value of the initial invocation of `BinSearch`, as we can see on lines 7 and 9, where the return value of the invoked `BinSearch` is returned without any modification. The case where `BinSearch` is called again is analyzed next:

- If $maxFlow(\frac{top+bot}{2}) < F' - \epsilon_1 < F'$ (line 6) then, since $maxFlow(\delta)$ is strictly decreasing, $\delta^* \in [bot, \frac{top+bot}{2})$. As we see on line 7, the interval $(\frac{top+bot}{2}, top]$ is discarded when the next `BinSearch` is called. Since $F'+\epsilon_2 \leq maxFlow(bot)$, we have $[F'-\epsilon_1, F'+\epsilon_2] \subset [maxFlow(\frac{top+bot}{2}), maxFlow(bot)]$ and the length of the available interval is divided by 2.
- Similarly, if $maxFlow(\frac{top+bot}{2}) > F' + \epsilon_2 > F'$ (line 8) then $\delta^* \in (\frac{top+bot}{2}, top]$. According to line 9, the interval $[bot, \frac{top+bot}{2})$ is discarded when the next `BinSearch` is called. Since $F' - \epsilon_1 \geq maxFlow(top)$, we have $[F' - \epsilon_1, F' + \epsilon_2] \subset (maxFlow(top), maxFlow(\frac{top+bot}{2})]$ and the length of the available interval is divided by 2.

As we saw, $[F' - \epsilon_1, F' + \epsilon_2] \subset [maxFlow(top), maxFlow(bot)]$ in every recursive call and $top - bot$ is divided by 2 in every call. From topology we know that $A \subset B \Rightarrow |A| < |B|$, so the recursive calls cannot continue infinitely. $|[F' - \epsilon_1, F' + \epsilon_2]| = \epsilon_1 + \epsilon_2$. Let bot_0, top_0 the input values given to the initial invocation of `BinSearch`, bot_j, top_j the input values given to the j-th recursive call of `BinSearch` and $len_j = |[bot_j, top_j]| = top_j - bot_j$. We have $\forall j > 0, len_j = top_j - bot_j = \frac{top_{j-1} - bot_{j-1}}{2} \Rightarrow \forall j > 0, len_j = \frac{top_0 - bot_0}{2^j}$. We understand that in the worst case $len_j = \epsilon_1 + \epsilon_2 \Rightarrow 2^j = \frac{top_0 - bot_0}{\epsilon_1 + \epsilon_2} \Rightarrow j = \log_2(\frac{top_0 - bot_0}{\epsilon_1 + \epsilon_2})$. Also, as we saw earlier, δ^* is always in the available interval, thus $maxFlow(\delta^*) \in [F' - \epsilon_1, F' + \epsilon_2]$. □

Complexity of function 4. Lines 1 - 2 have complexity $O(1)$, lines 4 - 5 have complexity $O(n)$, lines 6 - 11 have complexity $O(maxFlow) + O(BinSearch)$. As we saw in the proof of correctness for function 4, we need at most $\log_2(\frac{top - bot}{\epsilon_1 + \epsilon_2})$ recursive calls of `BinSearch`. Thus the function 4 has worst-case complexity $O((maxFlow + n)\log_2(\frac{top - bot}{\epsilon_1 + \epsilon_2}))$. □

Proof of correctness for algorithm 3. We will show that $maxFlow \in [F - V - \epsilon_1, F - V + \epsilon_2]$, with u_i' decided by algorithm 3.
Obviously $maxFlow(0) = F, maxFlow(\max_{i \in [n]}\{u_i\}) = 0$, thus $\delta^* \in \max_{i \in [n]}\{u_i\}$.
According to the proof of correctness for function 4, we can directly see that $maxFlow(\delta^*) \in [F - V - \epsilon_1, F - V + \epsilon_2]$, given that ϵ_1, ϵ_2 are chosen so that $F - V - \epsilon_1 \geq 0, F - V + \epsilon_2 \leq F$, so as to satisfy the condition $[F' - \epsilon_1, F' + \epsilon_2] \subset [maxFlow(top), maxFlow(bot)]$. □

Complexity of algorithm 3. The complexity of lines 1 - 2 and 4 - 5 is $O(1)$, the complexity of lines 3, 6, 8 - 9 and 10 is $O(n)$ and the complexity of line 7 is $O(BinSearch) = O((maxFlow + n)\log_2(\frac{\delta_{max}}{\epsilon_1 + \epsilon_2}))$, thus the total complexity of algorithm 3 is $O((maxFlow + n)\log_2(\frac{\delta_{max}}{\epsilon_1 + \epsilon_2}))$. □

However, we need to minimize $\sum_{i=1}^{n}(u_i - u_i') = ||\delta_i||_1$.