SAPIENZA
UNIVERSITÀ DI ROMA

Achieving Max Flow in Strongly Polynomial Time for Sparse Networks: Beyond the Edmonds-Karp Algorithm

Faculty of Information Engineering, Computer Science and Statistics
Bachelor's Degree in Computer Science

Armando Coppola
ID number 2003964

Advisor
Prof. Paul Joseph Wollan

Academic Year 2023/2024

Achieving Max Flow in Strongly Polynomial Time for Sparse Networks:Beyond the Edmonds-Karp Algorithm
Bachelor's Thesis. Sapienza University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: ArmandoCoppola24@gmail.com

To DO.

Contents

Chapter 1

Goldberg-Rao Algorithm

After understanding how Dinic's algorithm works, we can move to the next step and focus on the algorithm published by Andrew V. Goldberg and Satish Rao in 1998.

By optimizing the **??**, the Goldberg-Rao algorithm achieves a **computational cost** of

$$\tilde{O}(\min\{n^{2/3}, m^{1/2}\} \cdot m)$$

on a network with integer capacities, which, when considering logarithmic factors, becomes $O(\min\{n^{2/3}, m^{1/2}\} m \log n \log nU)$.

Note: From here on, we will abbreviate the expression $\min\{n^{2/3}, m^{1/2}\}$ using Λ.

1.1 Idea

At the core of the optimization is the idea of **contracting** the network according to certain specific parameters. The speed-up lies in computing the flow in a contracted graph, which is more efficient than computing it in the original one. The algorithm is based on **??** and introduces a new **binary length function** : $\bar{l}((v, w)) : E \to \{0, 1\}$.

The new length function assigns a value of zero to all edges that meet certain capacity requirements (which we will describe in more detail later), such edges are called "zero length".

By setting the length of the edges connecting two or more nodes to zero, we can consider them as a single node. Thus, by contracting the components connected by edges of length 0, it is possible to significantly reduce the number of flow increments and therefore the computational cost of the algorithm.

1.2 The Δ parameter

The issue with contracting the graph is that when we send flow from the source to the sink, we must ensure that this flow respects the capacity constraints of all the edges, including those that were contracted. To ensure that the flow calculated

on the contracted graph is valid for the original graph as well, a parameter Δ is used, which serves two purposes. The first function is as a **lower bound** on the capacity of zero-length edges. In fact, edges with residual capacity greater than Δ are first selected, and the length of these edges is set to 0. Subsequently, all components connected by zero-length edges are contracted. Finally, a blocking flow (exactly as in Dinic's algorithm) is calculated in the contracted graph. At this stage, the parameter Δ serves its second function, which is as an **upper bound for the blocking flow**. In fact, the computation of the blocking flow stops either when such a flow is found, or just before the flow value exceeds Δ. This second condition ensures that the flow remains feasible even for the original network.

By increasing the flow by at most Δ, we ensure that the capacity constraints are respected, but we can no longer guarantee that the flow is blocking. Therefore, we must choose a value for Δ that is both small enough to contract the graph as much as possible, but also adequately selected to keep the number of flow increments as low as possible, thus ensuring the desired computational cost.

1.3 Stop condition

To terminate its execution, the algorithm estimates the difference between the maximum achievable flow (which from now on will be called F) and the flow it has computed. When this difference becomes less than 1, the algorithm terminates. Since the capacities of the network are all integers, this ensures that the maximum possible flow has been reached. An initial useful value for F is $F = n \cdot U_{max}$, and later, the residual capacity of the canonical cut will be used (further details will be provided later).

1.4 Stimare il residual flow

We already know that the residual capacity of each cut $r(S, T)$ represents an upper bound on the max flow (**??**). To estimate the residual flow quickly and efficiently, we can analyze the **??**.

Lemma 1.1. *min $r(S_k, T_k)$ in $O(m)$ time The canonical cut with the minimum capacity can be found in $O(m)$ time.*

Proof. Exploiting the fact that each edge has a length of at most 1, and therefore can cross at most one canonical cut, we can define the following subroutine.

Algoritmh 1 *canCutCapacity(G_f, d, l)*

1: **for** $k \leftarrow 0$ *to* $d(s)$ **do** $r(S_k, T_k) = 0$
2: **end for**
3: **for** $(u, v) \in E(G_f)$ **do**
4: **if** $d(v) > d(w)$ **then** $r(S_{d(v)}, T_{d(v)}) += r(v, w)$
5: **end if**
6: **end for**
7: **return** argmin $r(S_k, T_k)$

The correctness and computational cost of this routine are fairly straightforward. □

To manage the computational cost, we need to ensure that the value of F decreases quickly enough without overburdening the algorithm. First, we can group all the iterations of the algorithm into **phases** and update the value of F at the minimum canonical cut only between the end of one phase and the start of a new one. If we update the value of F only when $\min r(S_k, T_k) \leq F/2$, the algorithm will terminate after at most $\log nU_{max}$ phases.

1.5 Binary length function

There are two other issues that arise from contracting the graph and modifying the length function:

1. Choosing a Δ that is too small would make the flow increase too slowly, while choosing it too large would not contract the graph enough to justify the management costs.

2. In Dinic's algorithm, the blocking flow always guaranteed an increase in the distance from s to t, but with zero-length edges, this is no longer guaranteed.

In this section, we show the choices that were made to address these two issues. While the effectiveness of the solution to the second problem is promptly demonstrated, the effectiveness of the choice of the Δ parameter will only become clear in the section where the various computational costs are proven.

1.5.1 How to zero lengths

As previously mentioned, we need an upper bound Δ to respect the capacity constraints. At the same time, to meet the declared computational cost, we need the blocking flow increments to be at most Λ. Thus, we can initialize:

$$\Delta = \lceil F/\Lambda \rceil$$

The criterion for assigning zero length to an edge is as follows:

Definition 1.1 (Length function)**.** Let r be the residual function of any residual graph G_f. We define the length function $l((u,v))$ as a function that associate to the edge (u,v) the value 1 or 0 as follow:

$$l(u.w) = \begin{cases} 0 & r_{vw} \geq 3\Delta \\ 1 & altrimenti \end{cases}$$

However, to achieve the desired computational cost, it is necessary to add a specification to this function.

Definition 1.2 (Special Arc)**.** Any edge (v,w) is said **special** If it meets all the following requirements:

- $2\Delta \leq r_{v,w} < 3\Delta$
- $d(v) = d(w)$
- $r_{wv} \geq 3\Delta$

By applying this definition to the length function, we can define a more complex function, which we distinguish from the first by calling it $\bar{l}$. The modified function also takes into account special edges:

$$\bar{l}(u.w) = \begin{cases} 0 & r_{vw} \geq 3\Delta \vee specialArc((v,w)) \\ 1 & altrimenti \end{cases}$$

Observation 1.1. *Introducing special edges does not change the distance labeling:* $d_l = d_{\bar{l}}$

Lemma 1.2 (From contract to original)**.** *Let's suppose we have contracted the original network as described so far, and routed a flow f through the contracted graph.*

The computational cost of adapting this flow through the original graph is $O(m)$.

Proof. Through the following steps, it is intuitive how the flow can be adapted:

1. Choose any vertex in each contracted component.
2. Form an in-tree and an out-tree rooted at the chosen vertices.
3. Route the positive flow from the in-tree to the root.
4. From the out-tree, redirect the incoming flow from the root to all other connected nodes.

Since the maximum flow we route is Δ and all nodes in the contracted components have a cost of at least 2Δ, we are assured that the flow respects the capacities of the network. It is evident that this method has a cost directly proportional to the number of edges in the connected components. □

1.5.2 How to increase distance

In Dinic's Algorithm, the proof that the blocking flow strictly increases the distance between s and t is quite obvious. The same cannot be said for the Goldberg-Rao case due to the presence of zero-length edges. Therefore, it is essential to prove the following theorem to ensure that the algorithm terminates.

Theorem 1.1. *Blocking flow with binary length Let $\bar{f}$ be a flow in $A(f, \bar{l}, d_l)$, let $f' = f + \bar{f}$ be the increased flow, and let l' be the length function corresponding to f'. Then:*

1. *d_l is a distance labeling with respect to l'*
2. *$d_{l'}(s) \geq d_l(s)$*
3. *if $\bar{f}$ is blocking $\implies$ $d_{l'}(s) > d_l(s)$*

Proof. Let's proceed point by point

1. d_l is a distance lebeling with respect to l' By the definition of distance labeling, $d_l(v) \leq d_l(w) + \bar{l}(v, w)$ (remembering that $d_l = d_{\bar{l}}$), we therefore need to prove that $d_l(v) \leq d_l(w) + l'(v, w)$.
This is trivially true if $d_l(v) \leq d_l(w)$.
If $d_l(v) > d_l(w)$ i.e. $d_{\bar{l}}(v) > d_{\bar{l}}(w)$ then (w, v) is not admissible with respect to $\bar{l}$.
Since $l'(v, w) \geq \bar{l}(v, w)$, the statement follows.

2. $d_{l'}(s) \geq d_l(s)$ Let $L := \{l_0, l_1, ..., l_n\}$ be the ordered set of all length functions calculated between the iterations of the algorithm. Then, for any $0 \leq i \leq j \leq n$, we have $d_{l_i}(s) \leq d_{l_j}(s)$.

We distinguish between two iterations as follows:

1. In iteration i: Let $l(u, v) = l_i(u, v)$ be the length function and $d(x) = d_{l_i}(x)$ be the distance. Together with the flow, they define $A(f, l_i, d_{l_i})$. Let Γ be the shortest $s \to t$ path in A, where $\Gamma \subseteq A$.
2. In iteration j: Let $l'(u, v) = l_j(u, v)$ be the length function and $d'(x) = d_{l_j}(x)$ be the distance. Together with the flow, they define $A'(f, l_j, d_{l_j})$. Let Γ' be the shortest $s \to t$ path in A', where $\Gamma' \subseteq A'$.

Suppose by contradiction that there exist two iterations $0 \leq i \leq j$ such that $d(s) > d'(s)$:

$$\implies \exists \Gamma\, s \to t,\ \Gamma'\, s \to t : \sum_{(v,w)\in\Gamma} l(v, w) \geq \sum_{(v,w)\in\Gamma'} l'(v, w)$$

In other words, as the iterations progress, s and t have gotten closer.

We immediately exclude the case where $\Gamma = \Gamma'$ since

$$\forall (v,w) \in A \cap A',\ l(v,w) \leq l'(v,w) \implies l(\Gamma) \leq l'(\Gamma')$$

Note: $l(\Gamma) = \sum_{(v,w)\in\Gamma} l(v,w)$.

Now consider Γ and Γ'. Let w be the last node in Γ for which $d(w) > d'(w)$, and let x be the next node:

$$w \in \Gamma : d(w) > d'(w) \wedge \exists x = succ_\Gamma(w) : d(x) \leq d'(x)$$

w and x are always well defined because we assume $d(s) > d'(s)$ and $d(t) = d'(t) = 0$ by definition.

Thus, there exists an arc (w,y) in Γ' with $y \neq x$ such that $d'(y) < d'(x)$. $x \neq y$, because if they were the same node, then:

$$d'(w) = d'(x) + l'(w,x) \geq d(w)$$

which contradicts the hypothesis.

To summarize, we know that:

1. $d(w) > d'(w) \iff d(x) + l(w,x) > d'(y) + l'(w,y)$.While we don't know the exact distance $d(y)$, we know that:

   $$d'(y) = \sum_{(a,b)\in y-t\subseteq\Gamma'} l'(a,b) \geq \sum_{(a,b)\in y-t\subseteq\Gamma'} l(a,b)$$

   Therefore, the path in iteration j is greater than or equal to the path in iteration i.

2. $d(y) + l(w,y) \leq d'(y) + l'(w,y) < d(x) + l(w,x)$.

However, we know that $d(w) = d(x) + l(w,x)$, which is **absurd** because it is not the minimal distance from $w \to t$, as it is greater than $d(y) + l(w,y)$.

We know for certain that the path $w - y \to t$ exists in A because (unless there is a shorter one) it represents the minimal distance from $w \to t$.

From this contradiction, the only conclusions are that either the path through y was not reachable in iteration i, making it impossible to reach it later, or if a shorter $s \to t$ path exists in iteration j than in iteration i, we made an error in considering the path in iteration i.

3. Se $\bar{f}$ è bloccante allora $d_l(s) < d_{l'}(s)$ To show that the blocking flow increases the distance of node s, we define the following notation:

$$c(v,w) := d_l(w) - d_l(v) + l'(v,w)$$

which represents the change in length of an edge connecting two adjacent nodes.

We can assert that:

$$\forall (v,w) \in E,\ c(v,w) \geq 0$$

since $l'(v,w) \geq l(v,w)$, which implies:

$$d_l(w) - d_l(v) < 0 \iff l(v,w) = 1 \implies l'(v,w) = 1$$

Now, consider any path Γ in $G_{f'}$, the length of the path is equal to:

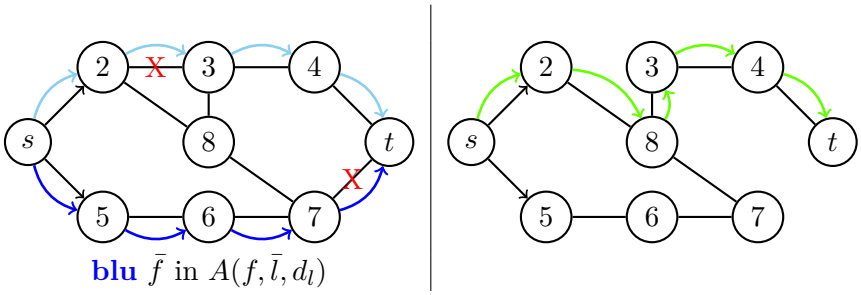
$$l'(\Gamma) = d_l(s) + c(\Gamma)$$

Therefore, to show that the path is longer, we need to show that:

$$\forall \text{ shortest } s-t \text{ path } \Gamma \in G_{f'} \implies \exists (v,w) \in \Gamma \text{ where } c(v,w) > 0$$

We now have a tool to demonstrate that the blocking flow increases the distance of s.

Table 1.1. graphic example to better visualize what was declared



blu $\bar{f}$ in $A(f, \bar{l}, d_l)$

Since $\bar{f}$ is blocking in $A(f, \bar{l}, d_l)$, Γ must contain an edge (v,w) that is not present in $A(f, \bar{l}, d_l)$.

Furthermore, we can state that $d_l(v) \leq d_l(w)$, either because $(v,w) \in G_f$, but then if $d_l(v) > d_l(w)$, we would have $(v,w) \in A(f, \bar{l}, d_l)$, or because $(v,w) \notin G_f$, but it appears in $G_{f'}$, which is only possible if the flow is incremented in the opposite direction, causing the residual edge to appear. Therefore, $(w,v) \in A(f, \bar{l}, d_l)$, which implies that $d_l(v) \leq d_l(w)$.

Now, suppose for contradiction that $c(v,w) = 0$, so $d_l(v) = d_l(w)$ and $l'(v,w) = 0$. The fact that (v,w) is not in $A(f, \bar{l}, d_l)$ implies that either (v,w) is not in G_f, but then we have already shown that the opposite edge $(w,v) \in A(f, \bar{l}, d_l)$, or that $(v,w) \in G_f$ but does not meet the distance labeling requirements to belong to the **??** $A(f, \bar{l}, d_l)$. Since $d_l(v) = d_l(w)$, then $l(v,w) = 1$. We note that $1 = l(v,w) > l'(v,w) = 0$, which implies that we have incremented the flow on the opposite edge (w,v). Thus, in any case, the edge $(w,v) \in A(f, \bar{l}, d_l)$.

As shown earlier, since $d_l(v) = d_l(w)$,

$$(w,v) \in A(f, \bar{l}, d_l) \iff l(w,v) = 0$$

We conclude that:

- During the flow increments, we routed a flow (of at most Δ) through the edge (w,v)

- $u_f(w,v) \geq 3\Delta$ because $l(w,v) = 0$
- After this increment, we have $u_{f'}(v,w) \geq 3\Delta$ because $l'(v,w) = 0$
- Thus $u_f(v,w) \geq 2\Delta$
- But then the edge (v,w) was a *special edge* even before the increment, since $d_l(v) = d_l(w) \wedge u_f(w,v) \geq 3\Delta \wedge u_f(v,w) \geq 2\Delta$

We therefore conclude that:

$$d_l(v) = d_l(w) \implies d_{\bar{l}}(v) = d_{\bar{l}}(w) \wedge \bar{l}(v,w) = 0 \implies (v,w) \in A(f, \bar{l}, d_l)$$

which is a contradiction.

□

1.6 Costo computazionale

We have already shown how to find the maximum flow in the graph. Before diving into the cost of a phase, let's review the structure of the algorithm described so far.

Algoritmh 2 *Goldberg-RaoAlgorithm(G, c)*

1: $n = |N(G)|$
2: $F = U \cdot n$
3: $\Delta = F/\Lambda$
4: **for** $(i,j) \in E(G)$ **do** $f_{ij} = 0$
5: **end for**
6: **while** $F \geq 1$ **do**
7: $l = update\_length(n, \Delta)$ ▷ return a length function w.r.t. Δ
8: $d_l = BFS(G_f, l)$ ▷ return a distance labeling w.r.t. l
9: $G^c = contract(G_f, l)$
10: $A = A(G^c, d_l, l)$ ▷ return the admissible graph
11: $f' = find\_blocking\_or\_Delta\_flow(A)$ ▷ Dinic's style
12: $f_{ad} = fit(f')$ ▷ the procedure to adapt the flow to the original graph
13: $f = f + f_{ad}$
14: $c = r(canCutCapacity(G_f, d, l))$ ▷ residual capacity of min canon. cut
15: **if** $c \leq F/2$: **then**
16: $F = c$
17: $\Delta = F/\Lambda$
18: **end if**
19: **end while**
20: **return** f

Remark:

The cost stated at the beginning is in:

$$O(\min\{n^{2/3}, m^{1/2}\} \cdot m \log n \log mU_{max})$$

Using more advanced data structures, you can achieve the cost of:

$$O(\min\{n^{2/3}, m^{1/2}\} \cdot m \log \frac{n^2}{m} \log U_{max})$$

We divided the number of **phases** so that the estimated maximum flow (F) is halved in each phase. This gives us a number of phases on the order of $\log(F)$, which is $\log(mU_{max})$. We have shown how both the calculation of the minimum canonical cut and the adjustment of the flow to the original network can be computed in $O(m)$ time. However, we still need to analyze the cost of each phase, that is, how quickly the minimum canonical cut is halved.

From the corollary of the following lemma, we demonstrate what was previously stated when we fixed the value of the parameter Δ. With this lemma, we estimate the maximum capacity for the canonical cut, which is then used to estimate F, while in the subsequent corollary, we show how the parameters for which we contract the graph lead this capacity to halve within $O(\Lambda)$ blocking flows.

Lemma 1.3. *The minimum capacity of a canonical cut $(\bar{S}, \bar{T})$ satisfies*

$$u_f(\bar{S}, \bar{T}) \leq \frac{mM}{d_l(s)}$$

where M represents the length-one edge with the highest capacity.

Proof. It is clear that the best way to maximize the capacity of the minimum canonical cut is by assuming that all edges have the capacity of the edge with the highest capacity, and then evenly dividing the edges among the various cuts.

□

From this initial estimate follows the corollary.

Corollary 1.1. *During each phase, there are at most $O(\Lambda)$ blocking flow increments.*

Proof. Let us assume that $\Lambda = m^{1/2}$. Since we have shown that each blocking flow increases $d(s)$ by at least one, we can be sure that after $6\lceil\Lambda\rceil$ increments, $d_l(s) \geq 6m^{1/2}$. Thus, we can take the estimate from the lemma and state that:

$$u_f(\bar{S}, \bar{T}) \leq \frac{mM}{d_l(s)} \leq \frac{3m}{d_l(s)}\Delta \leq \frac{3m}{6m^{1/2}} \frac{F}{m^{1/2}} = \frac{F}{2}$$

Thus, after $\lceil\Lambda\rceil$, the phase ends.
For $\Lambda = n^{2/3}$, the proof is analogous and leads to the same conclusion. In conclusion, the cost of each phase is on the order of $O(\Lambda)$. □

At this point, the last bottleneck is represented by the cost of finding a blocking flow or a flow of value Δ (which are computationally equivalent): this would require a cost of:

- $O(mn)$ in a naive approach;
- $O(m \log n)$ using dynamic trees;
- $O(m \log(n^2/m))$ using size-bounded dynamic trees;

Combining the cost of:

$\times$ finding a blocking flow

$\times$ iterations in each phase

$\times$ the number of phases

$\times$ additional costs in $O(m)$

Conclusion

Goldberg and Rao published their algorithm in a 1998 paper. About four years earlier, V. King, S. Rao, and R. Tarjan had published a paper in which they presented an algorithm capable of finding the maximum flow in $O(nm)$ time, provided the algorithm had enough edges relative to the number of nodes. The stated cost is $O(nm(\log_{m/n \log n} n))$, but if $m/n = \Omega(n^{\varepsilon})$ for some $\varepsilon > 0$, the cost becomes $O(nm)$. However, the problem of finding the maximum flow in polynomial time remains unresolved, as it is still unclear how to calculate it for sparser graphs than those covered by the King-Rao-Tarjan algorithm. The next chapter analyzes the solution proposed in 2013 by James B. Orlin, which leverages a specific condition of the Goldberg-Rao algorithm and develops a strategy for compacting and contracting the graph, along with approximations in a series of optimal flows, to make the algorithm strictly polynomial where King-Rao-Tarjan fails.