University "Alexandru Ioan Cuza" of Iaşi
Faculty of Computer Science

TECHNICAL REPORT

A simple push-relabel algorithm for sub-modular flows

Cristian Frasinaru, Emanuel F. Olariu

TR 13-02, October 2013

ISSN 1224-9327

Universitatea "Alexandru Ioan Cuza" Iaşi
Facultatea de Informatică
Str. Berthelot 16, 6600-Iaşi, Romania
Tel. +40-32-201090, email: bibl@infoiasi.ro

Abstract

In this paper we present a combinatorial push-relabel algorithm for sub-modular flows. Our procedure, using a lowest level rule combined with a *bfs*-like traversal, needs no lexicographic order of the elements, and gives a time complexity of $\mathcal{O}(n^5)$.

Keywords: sub-modular flows, push-relabel algorithms, combinatorial optimization.

1 Introduction

Push-relabel algorithms are often used in the sub-modular optimization especially for the sub-modular flow problem ([3], [5]). This framework includes some important combinatorial optimization problems as polymatroid intersection, minimum cost problem, or minimizing sub-modular functions ([6], [8], [10]). Push-relabel algorithms are an alternative to the procedures using augmenting path technique (see [7] in the context of minimizing sub-modular functions).

2 Sub-modular flows

We shortly outline here the context of the sub-modular flows. Let $G = (V, A)$ a directed graph with no multiple arcs. If $\mathbf{x} : A \to \mathbb{R}$ is a function on the arcs of G, we define three new functions δ^-, δ^+ and $\Psi_{\mathbf{x}} : \mathcal{P}(V) \to \mathbb{R}$ like this:

$$\delta^-_{\mathbf{x}}(Z) = \sum_{uv \in A, u \notin Z \ni v} \mathbf{x}(uv), \ \delta^+_{\mathbf{x}}(Z) = \sum_{uv \in A, u \in Z \not\ni v} \mathbf{x}(uv), \ \forall Z \subseteq V, \text{ and}$$

$$\Psi_{\mathbf{x}}(Z) = \delta^-_{\mathbf{x}}(Z) - \delta^+_{\mathbf{x}}(Z), \ \forall Z \subseteq V.$$

In order to introduce the notion of *sub-modular flow* we need two bounding functions $f, g : A \to \mathbb{R}$ for which $f \leqslant g$, and a sub-modular function[1] $b : \mathcal{P}(V) \to \mathbb{R}$ for which $b(V) = 0$. Given such a function, the *base-polyhedron* of b is $B(b) = \{\mathbf{y} \in \mathbb{R}^V \ : \ \mathbf{y}(Z) \leqslant b(Z), \ \forall Z \subset V, \ \mathbf{y}(V) = b(V)\}$, where for a function $\mathbf{y} \in \mathbb{R}^V$, $\mathbf{y} : 2^V \to \mathbb{R}$ with $\mathbf{y}(Z) = \sum_{u \in Z} \mathbf{y}(u)$, for any $Z \subseteq V$.

A *sub-modular flow* is a function $\mathbf{x} : A \to \mathbb{R}$ for which

$$\Psi_{\mathbf{x}}(Z) \leqslant b(Z), \ \forall Z \subseteq V.$$

A sub-modular flow is *feasible* if

$$f(e) \leqslant \mathbf{x}(e) \leqslant g(e), \ \forall e \in A.$$

Our study concerns the characterization of the existence of feasible sub-modular flows:

Theorem 2.1. *([1]) Let $G = (V, A)$ a directed graph without multiple arcs, $f, g : A \to \mathbb{R}$ for which $f \leqslant g$, and $b : \mathcal{P}(V) \to \mathbb{R}$ a sub-modular function. In this context there is a feasible sub-modular flow if and only if*

$$\delta^-_f(Z) - \delta^+_g(Z) \leqslant b(Z), \ \forall Z \subseteq V. \tag{2.1}$$

Moreover, if f, g, and b are integral, then the above condition insures the existence of an integral feasible sub-modular flow.

For the only if part: let $\mathbf{x}$ be a feasible sub-modular flow; we have $\delta^-_f(Z) - \delta^+_g(Z) \leqslant \delta^-_{\mathbf{x}}(Z) - \delta^+_{\mathbf{x}}(Z) \leqslant b(Z), \ \forall Z \subseteq V$. For the if part there are two different approaches, one which use an augmenting path type argument (Frank in [1]), and a later one which uses a push-relabel algorithm (see Fujishige and Zhang in [4] or Frank and Miklós in [2]). The push-relabel algorithm either finds a feasible sub-modular flow or finds a subset $Z_0 \subseteq V$ which violates condition (2.1):

$$\delta^-_f(Z_0) - \delta^+_g(Z_0) > b(Z_0)$$

[1] A function f is (fully) sub-modular if, for every $S, T \subseteq V$, we have that $f(S) + f(T) \geqslant f(S \cap T) + f(S \cup T)$.

We develop here a simple push-relabel algorithm based on a modified procedure used in [2].

For a vector (function) $m : V \to \mathbb{R}$, an *m-flow* is a function $\mathbf{x} : A \to \mathbb{R}$ for which $\Psi_{\mathbf{x}}(v) = m(v)$, $\forall v \in V$.

Lemma 2.1. *There is a feasible sub-modular flow if and only if we have a vector $m \in B(b)$ for which it exists a feasible m-flow.*

Proof. First, suppose that, for a certain $m \in B(b)$, it exists a feasible m-flow $\mathbf{x}$; we need only to check that $\mathbf{x}$ is sub-modular. Using the modularity of $\Psi_{\mathbf{x}}$ we have that

$$\Psi_{\mathbf{x}}(Z) = \sum_{v \in Z} \Psi_{\mathbf{x}}(v) = \sum_{v \in Z} m(v) = m(Z) \leqslant b(Z).$$

If $\mathbf{x}$ is a feasible sub-modular flow, we define $m : V \to \mathbb{R}$ by taking $m(v) = \Psi_{\mathbf{x}}(v)$; using again the modularity of $\Psi_{\mathbf{x}}$ we get $m(Z) \leqslant b(Z)$, $\forall Z \subseteq V$. On the other hand $m(V) = \Psi_{\mathbf{x}}(V) = 0$, hence m is in $B(b)$ and $\mathbf{x}$ is a (feasible) m-flow.

□

3 Preliminaries of the algorithm

Our procedure maintains a feasible vector $\mathbf{x}$ and a vector m from the base polyhedron. We modify $\mathbf{x}$ until $\Psi_{\mathbf{x}}(v) \leqslant m(v)$, for all $v \in V$. It is noteworthy that our strategy differs from that of Frank and Miklós in [2] as m remains unchanged during the algorithm.

A vertex $u \in V$ will be called Ψ-*larger* (Ψ-*smaller* or Ψ-*neutral*) if $\Psi_{\mathbf{x}}(u) > m(u)$ ($\Psi_{\mathbf{x}}(u) < m(u)$ or $\Psi_{\mathbf{x}}(u) = m(u)$). An arc $e \in A$ can be *increasable* if $\mathbf{x}(e) < g(e)$, *decreasable* if $\mathbf{x}(e) > f(e)$.

For a given vector $m \in B(b)$, a subset Z will be called m-*tight* if $m(Z) = b(Z)$.

Remark 3.1. *b being a fully sub-modular function, it is easy to see that the collection of m-tight sets is closed under union and intersection. Therefore, for a vertex $v \in V$, we can define $T(v)$, the smallest (inclusion wise) m-tight set containing v.*

The algorithm maintains a level function $\varphi : V \to \{0, 1, \ldots, n\}$ (where $|V| = n$), the level sets are $\Lambda_k = \varphi^{-1}(k)$, $0 \leqslant k \leqslant n$. For a given set $Z \subseteq V$, we denote

$$\varphi_{min}(Z) = \min_{u \in Z} \varphi(u).$$

A generic push-relabel type procedure performs some local optimization steps - at a vertex in our case - and the selection of the vertex for the next step is made using the level function. Usually the strategies for choosing the next vertex is to take a vertex from the maximum or minimum level.

Throughout the execution of the algorithm the following three level properties are invariant

$$(L1) \qquad \varphi(u) = 0, \text{ for every } \Psi\text{-smaller } u \in V;$$

$$(L2) \quad (a) \quad \varphi(v) \geqslant \varphi(u) - 1, \text{ for every increasable arc } uv;$$

$$(b) \quad \varphi(v) \leqslant \varphi(u) + 1, \text{ for every decreasable arc } uv;$$

$$(L3) \qquad \varphi_{min}(T(u)) \geqslant \varphi(u) - 1, \text{ for every vertex } u \in V.$$

Property (L_2) says that every increasable (decreasable) arc steps down (up) at most one level. The algorithm ends when either

(I) there is no Ψ-larger elements

or

(II) there is an empty level Λ_k and a Ψ-larger vertex over k*th* level.

The following result from [2] concerns the link between the level properties and the stopping rules.

Lemma 3.1. *Let* $\mathbf{x}$ *be a feasible function,* $m \in B(b)$*, and* φ *a level function which satisfies properties* (L_1) *-* (L_3)*. Then*

(i) If $\Lambda_k = \varnothing$*, and there are vertices over the* k*th level, then* $Z = \{v \in V : \varphi(v) > k\}$ *is* m*-tight and* $\Psi_{\mathbf{x}}(Z) = \delta_f^-(Z) - \delta_g^+(Z)$*.*

(ii) If (I) *holds, then* $\mathbf{x}$ *is a sub-modular flow;*

(iii) If (II) *holds, then* $\{v \in V : \varphi(v) > k\}$ *violates condition* (2.1)*.*

Proof. (i) for every $v \in Z$, and every $u \in T(v)$ we have that $\varphi(u) \geqslant \varphi_{min}(T(v)) \geqslant \varphi(v) - 1 \geqslant k$, hence $u \in Z$, $\forall u \in T(v)$. It follows that $T(v) \subseteq Z$, $\forall v \in Z$, and, from here, $\bigcup_{v \in Z} T(v) = Z$ which means that Z is m-tight. As $\Lambda_k = \varnothing$, every arc leaving Z steps down at least two levels, hence there are no leaving increasable arcs from Z ($\mathbf{x}(e) = g(e)$, $\forall$ arc e leaving Z), hence $\delta_{\mathbf{x}}^+(Z) = \delta_g^+(Z)$. In the same way we get $\delta_{\mathbf{x}}^-(Z) = \delta_f^-(Z)$ (every arc entering Z steps up at least two levels means there are no entering decreasable arcs to Z). It follows that $\Psi_{\mathbf{x}}(Z) = \delta_f^-(Z) - \delta_g^+(Z)$.

(ii) Suppose there are no Ψ-larger vertices, then, for every $X \subseteq V$ we have

$$\Psi_{\mathbf{x}}(X) = \sum_{v \in X} \Psi_{\mathbf{x}}(v) \leqslant \sum_{v \in X} m(v) = m(X) \leqslant b(X).$$

(iii) If (II) holds, from (i), Z is m-tight, and we have $\Psi_{\mathbf{x}}(Z) = \delta_f^-(Z) - \delta_g^+(Z)$, where $Z = \{v \in V : \varphi(v) > k\}$. $\Lambda_0 \cap Z = \varnothing$, therefore Z contains only Ψ-larger (at least one) or -neutral vertices and therefore

$$\Psi_{\mathbf{x}}(Z) = \sum_{v \in Z} \Psi_{\mathbf{x}}(v) > \sum_{v \in Z} m(v) = m(Z) = b(Z).$$

It follows that $\delta_f^-(Z) - \delta_g^+(Z) = \Psi_{\mathbf{x}}(Z) > b(Z)$ which violates (2.1).

□

4 The algorithm

At each step of the algorithm, while there are Ψ-larger vertices, we select such a vertex s, and, as long as s stays larger, we perform a number of pushes along its incident increasable or decreasable arcs; when such arcs are no more available we increment the label of s. This is a version of the algorithm proposed by Frank and Miklós in [2].

Therefore there are three basic operations:

(a) *arc-push* at s on an arc e incident with s (we say that the arc-push is carried out along e):

1. (*increase*) if $e = su$ is an increasable arc with $\varphi(u) = \varphi(s) - 1$, then increase $\mathbf{x}(su)$ by $\alpha = \min\{g(su) - \mathbf{x}(su), \Psi_{\mathbf{x}}(s) - m(s)\}$;
2. (*decrease*) if $e = us$ is a decreasable arc with $\varphi(u) = \varphi(s) - 1$, then decrease $\mathbf{x}(us)$ by $\alpha = \min\{\mathbf{x}(us) - f(us), \Psi_{\mathbf{x}}(s) - m(s)\}$;

(b) *node-push* at s, when $min\varphi(T(s)) = \varphi(s) - 1$: finds a vertex $t \in T(s)$ with $\varphi(t) = \varphi(s)???1$, increases $m(s)$ by $\alpha = \min\{\Psi_{\mathbf{x}}(s) - m(s), \Delta(t, s)\}$, and decreases $m(t)$ by α (we say that the node-push is carried out along (s, t)).

(c) *lift* s: increment $\varphi(s)$.

A push at s is *neutralizing* if s becomes neutral and *non-neutralizing* otherwise. All these basic operations preserve the level properties [2].

The algorithm works as follows:

- a queue is initialized with a Ψ-larger node from the lowest level; we redo this initialization as long as there are Ψ-larger nodes, and the queue becomes empty;
- in a typical step of the procedure, we extract a vertex s from the queue;
- as long as s remains Ψ-larger: we apply edge-pushes at s, when no more such pushes are possible we apply node-pushes at s; in the end, if s is still Ψ-larger, we lift s;
- if, during a push, a new Ψ-larger vertex is created (see remark 4.1 below), we introduce it in the queue (if it is not already there).

Algorithm 1 The algorithm

$\mathcal{Q} \leftarrow \varnothing$; ▷ $\mathcal{Q}$ is a queue
for $(u \in V)$ **do**
  $visited[u] \leftarrow 0$; $parent[u] \leftarrow 0$;
end for
while (not(I) and not(II)) **do**
  let $s \in V$ be a Ψ-larger, φ-minimum vertex;
  $\mathcal{Q}.push(s)$;
  while $(\mathcal{Q} \neq \varnothing)$ **do**
    $s \leftarrow \mathcal{Q}.pop()$; $visited[s] \leftarrow 1$; $parent[s] \leftarrow s$;
    while ($\exists$ an increasable arc su and $\varphi((u) = \varphi(s) - 1)$ **do**
      $\alpha \leftarrow \min\{g(su) - \mathbf{x}(su), \Psi_{\mathbf{x}}(s) - m(s)\}$;
      $\mathbf{x}(su) \leftarrow \mathbf{x}(su) + \alpha$;
      if ($\Psi_{\mathbf{x}}(u) > m(u)$ and $visited[u] = 0$) **then** ▷ if u becomes Ψ-larger
        $\mathcal{Q}.push(u)$; $visited[u] \leftarrow 1$; $parent[u] = s$;
      end if
    end while
    while (($\exists$ a decreasable arc us and $\varphi((u) = \varphi(s) - 1)$ **do**
      $\alpha \leftarrow \min\{\mathbf{x}(us) - f(us), \Psi_{\mathbf{x}}(s) - m(s)\}$;
      $\mathbf{x}(us) \leftarrow \mathbf{x}(us) - \alpha$;
      if ($\Psi_{\mathbf{x}}(u) > m(u)$ and $visited[u] = 0$) **then** ▷ if u becomes Ψ-larger
        $\mathcal{Q}.push(u)$; $visited[u] \leftarrow 1$; $parent[u] = s$;
      end if
    end while
    while ($\varphi_{min}(T(s)) = \varphi(s) - 1$) **do**
      let $t \in T(s)$, $\varphi(u) = \varphi_{min}(T(s))$;
      $\alpha \leftarrow \min\{\Psi_{\mathbf{x}}(s) - m(s), \Delta(u, s)\}$;
      $m(s) \leftarrow m(s) + \alpha$; $m(u) \leftarrow m(u) - \alpha$;
      if ($\Psi_{\mathbf{x}}(u) > m(u)$ and $visited[u] = 0$) **then** ▷ if u becomes Ψ-larger
        $\mathcal{Q}.push(u)$; $visited[u] \leftarrow 1$; $parent[u] = s$;
      end if
    end while
    if ($\Psi_{\mathbf{x}}(s) > m(s)$) **then** ▷ if s is still Ψ-larger
      $\varphi(s) + +$; $\mathcal{Q} \leftarrow \varnothing$;
      for $u \in V$ **do**
        $visited[u] \leftarrow 0$; $parent[u] \leftarrow 0$;
      end for
    end if
  end while
end while

Remark 4.1. *After an arc-push at s, as $\Psi_{\mathbf{x}}(u)$ increases, u can become Ψ-larger, and, in a similar manner, after a nod-push along (s, u), u can become Ψ-larger, as $m(u)$ decreases.*

5 Complexity

As in [9] we use a minimum level strategy in order to choose the next vertex to be treated by the algorithm. Obviously the time complexity depends on the strategy of this choosing. We use a *bfs* like procedure starting with a Ψ-larger element s from a minimum possible level. We restart with such an element whenever the queue becomes empty and the stopping rules are not yet fulfilled.

The execution of the algorithm is decomposed in phases: a *phase* is the segment of execution between two liftings.

Lemma 5.1. *There are at most n^2 phases.*

Proof. Each elements starts on the zero level and can reach the nth level at most. □

Each phase consists of a variable number of waves: a *wave* starts when the queue is empty and a new Ψ-larger element is added to it and will end when the queue becomes again empty or a lifting arises - whichever comes first.

To a wave W we can associate a (bfs) tree $T = (W, E)$ - for which the root (say s) and the interior vertices are (become) neutral (after the wave is finished). Apart from the edges of T, there are unapparent (invisible) edges of the form su: when we made an arc-push along su(or us), or a node-push along (s, u), and the vertex u was already visited.

To each phase we associate the forest corresponding to all waves which belong to it.

Lemma 5.2. *Each phase consists of at most n waves.*

Proof. Let W_i be an intermediate wave. A level of the tree $T_i = (W_i, E_i)$ contains vertices from the same φ-level and in decreasing order, as $\varphi(parent[u]) = \varphi(u) + 1$, except for the root, s_i. Clearly $\varphi(s_{i+1}) \geqslant \varphi(s_i)$, and because after the wave W_i vertex s_i becomes neutral, it follows that $s_{i+1} \neq s_i$; s_i will remain neutral after each subsequent wave, hence $s_i \neq s_j$, for every $i < j$. As an exception, if W_i is the last wave, we can have $\varphi(s_i) = 0$.

Therefore the number of waves in a phase is at most the number of vertices in the initial graph.

□

Lemma 5.3. *In a wave we have at most n^2 node-pushes.*

Proof. This is obvious, as each tree wave has at most n vertices, and in-between these vertices we can have $n(n-1)/2$, apparent or not, edges - each such edge corresponds to a node- or arc-push. □

Lemma 5.4. *In a phase the number of non-neutralizing arc-pushes is at most $m = |A|$, and the number of neutralizing arc-pushes is at most n^2. Therefore the total number of arc-pushes is $\mathcal{O}(n^4)$.*

Proof. First, for non-neutralizing arc-pushes, we discuss only the case of increasable arcs (the same proof works for decreasable arcs). If su is increasable $\varphi(s) = \varphi(u) - 1$, and $x(su)$ becomes $g(su)$; $x(su)$ cannot be modified until su becomes decreasable, which implies that $\varphi(u) = \varphi(s) - 1$. Hence, in the same phase $x(su)$ can be modified at most once. As we have m arcs in G, the number of non-neutralizing arc-pushes will be $\mathcal{O}(m)$.

On the other hand, in each wave we can have at most one neutralizing arc-push per vertex: after a neutralizing push a node v remains neutral until the end of the current wave. Therefore a wave has at most n neutralizing arc-pushes, and a phase has at most n^2 such pushes.

□

The above lemmata give

Theorem 5.1. *The time complexity of the algorithm is $\mathcal{O}(n^5)$.*

6 Conclusions

In this study we emphasize the strategy that uses the lowest level rule. This rule does not work alone as new Ψ-larger nodes on lower levels can appear during treatment of a current node. Hence, it is reinforced with a *bfs* traversal: the new appeared larger nodes are added to a queue, and when this queue becomes empty a new lowest level, larger vertex is chosen to start a new queue.

This strategy brings a forest structure of the treated nodes, where the basic operations (pushes and liftings) can be easily numbered. The other known strategy ([2]) based only on the largest level rule has a claimed complexity of $\mathcal{O}(n^4)$ and exploits the structure of sets $T(s)$. It seems that a future work should avoid these sets, those determination imply a non-negligible cost - $\mathcal{O}(n)$ - which is not apparent in the cited article. The authors believe that this technical rapport begins a way for a simpler $\mathcal{O}(n^2m)$ algorithm for finding a feasible sub-modular flow - at least for particular types of networks.

Acknowledgments. The authors thank Prof. C. Croitoru for his comments on an earlier version of this rapport.

References

[1] Frank, A., *Finding feasible vectors of Edmonds-Giles polyhedra*, Journal of Combinatorial Theory, Series B, 36 (4), pp. 221–239, 1984.

[2] Frank, A., Miklós, Z., *Simple push-relabel algorithms for matroids and submodular flows*, Japan Journal of Industrial and Applied Mathematics, vol. 29, Issue 3, pp. 419-439, 2012.

[3] Fujishige, S., X. Zhang, *New algorithms for the intersection problem of submodular systems*, Japan Journal of Industrial and Applied Mathematics, vol. 9, pp.369–382, 1992.

[4] Fujishige, S., X. Zhang, *A push/relabel framework for submodular flows and its refinement for 0-1 submodular flows*, Optimization, vol. 38, Issue 2, pp. 133–154,1996.

[5] Fujishige, S., S. Iwata, *Algorithms for Submodular Flows*. IEICE TRANS. Inform. Syst., 2000.

[6] Fujishige, S., *Submodular Functions and Optimization*, 2nd edition,Annals of Discrete Mathematics, vol. 58, Elsevier, 2005.

[7] Iwata, S., L. Fleischer, S. Fujishige, *A Combinatorial, Strongly Polynomial-Time Algorithm for Minimizing Submodular Functions*, Journal of ACM, vol. 48, pp. 761–777, 2001.

[8] Iwata, S., J. B. Orlin, *A simple combinatorial algorithm for submodular function minimization*, Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2009.

[9] Olariu, E. F., *A new strategy for pus-relabel algorithm framework for matroid optimization*, TR 13-01, http://profs.info.uaic.ro/ tr/tr13-01.pdf, 2013.

[10] Schrijver, A., *A combinatorial algorithm minimizing submodular functions in strongly polynomial time*, Journal of Combinatorial Theory Series B, vol. 80, Issue 2, pp. 346–355, 2000.