A Distributed Mincut/Maxflow Algorithm Combining Path Augmentation and Push-Relabel

Alexander Shekhovtsov and Václav Hlaváč
shekhole@fel.cvut.cz hlavac@fel.cvut.cz

Czech Technical University in Prague

Abstract. We present a novel distributed algorithm for the minimum s-t cut problem, suitable for solving large sparse instances. Assuming vertices of the graph are partitioned into several regions, the algorithm performs path augmentations inside the regions and updates of the push-relabel style between the regions. The interaction between regions is considered expensive (regions are loaded into the memory one-by-one or located on separate machines in a network). The algorithm works in sweeps, which are passes over all regions. Let $\mathcal{B}$ be the set of vertices incident to inter-region edges of the graph. We present a sequential and parallel versions of the algorithm which terminate in at most $2|\mathcal{B}|^2 + 1$ sweeps. The competing algorithm by Delong and Boykov uses push-relabel updates inside regions. In the case of a fixed partition we prove that this algorithm has a tight $O(n^2)$ bound on the number of sweeps, where n is the number of vertices. We tested sequential versions of the algorithms on instances of maxflow problems in computer vision. Experimentally, the number of sweeps required by the new algorithm is much lower than for the Delong and Boykov's variant. Large problems (up to 10^8 vertices and $6 \cdot 10^8$ edges) are solved using under 1GB of memory in about 10 sweeps.

Keywords: mincut, maxflow, distributed, parallel, large-scale, streaming, augmented path, push-relabel, region

1 Introduction

Minimum s-t cut (MINCUT) is a classical combinatorial problem with applications in many areas of science and engineering. This research[1] was motivated by wide use of MINCUT/MAXFLOW in computer vision, where large sparse instances need to be solved. To deal efficiently with the large scale we consider *distributed* algorithms, dividing the computation *and* the data between computation units and assuming that passing information from one unit to another is expensive. We consider the following two practical usage modes:

- Sequential (or *streaming*) mode, which uses a single computer with a limited memory and a disk storage, reading, processing and writing back a part of

[1] A. Shekhovtsov was supported by the EU project FP7-ICT-247870 NIFTi and V. Hlavac by the EU project FP7-ICT-247525 HUMAVIPS.

data at a time. Since it is easier for analysis and implementation, this mode will be the main focus of this work.

- Parallel mode, in which the units are *e.g.* computers in a network. We show that the algorithm we propose admits full parallelization. The theoretical analysis is derived from the sequential variant. Details and preliminary experiments on a single computer with several CPUs are presented in the technical report [1].

To represent the cost of information exchange between the units, we use a special related measure of complexity. We call a *sweep* the event when all units of a distributed algorithm recalculate their data once. The number of sweeps is roughly proportional to the amount of communication in the parallel mode or disk operations in the streaming mode.

Previous Work. A variant of path augmentation algorithm was shown in [2] to have the best performance on computer vision problems among sequential solvers. There were several proposals how to parallelize it. Partially distributed implementation [3] augments paths within disjoint regions first and then merges regions hierarchically. In the end, it still requires finding augmenting paths in the whole problem. A distributed algorithm was obtained in [4] using the dual decomposition approach. The subproblems are MINCUT instances on the parts of the graph (regions) and the master problem is solved using subgradient method. This approach requires solving MINCUT subproblems with real valued capacities (rather than integer ones) and does not have a polynomial iteration bound.

The push-relabel algorithm [5] performs many local atomic operations, which makes it a good choice for a parallel or distributed implementation. A distributed version [6] runs in $O(n^2)$ time using $O(n)$ processors and $O(n^2\sqrt{m})$ messages. Delong and Boykov [7] proposed a coarser granulation, associating a subset of vertices (a region) to each processor. Push and relabel operations inside a region are decoupled from the rest of the graph. This allows to process several non-interacting regions in parallel or run in a limited memory, processing one region at a time. For the case of a fixed partition we prove that the sequential and our novel parallel versions of their algorithm have a tight $O(n^2)$ bound on the number of sweeps. We then construct a new algorithm, which works with the same partition of the data but is guided by a different distance function than push-relabel.

The New Algorithm. Given a fixed partition into regions, we introduce a distance function which counts the number of region boundaries crossed by a path to the sink. Intuitively, it corresponds to the amount of costly operations – network communications or loads-unloads of the regions in the streaming mode. The algorithm maintains a labeling, which is a lower bound on the distance function. Within a region, we first augment paths to the sink and then paths to the boundary nodes of the region in the order of their increasing labels. Thus the flow is pushed out of the region in the direction given by the distance estimate. We present a sequential and parallel versions of the algorithm which terminate in at most $2|\mathcal{B}|^2 + 1$ sweeps, where $\mathcal{B}$ is the set of all boundary nodes (incident to inter-region edges).

Other Related Work. The following works do not consider a distributed implementation but are relevant to our design. Partial Augment-Relabel algorithm (PAR) [8] in each step augments a path of length k. It may be viewed as a lazy variant of push-relabel, where actual pushes are delayed until it is known that a sequence of k pushes can be executed. The algorithm of [9] incorporates the notion of a length function and a valid labeling w.r.t. this length. It can be seen that the labeling maintained by our algorithm corresponds to the length function assigning 1 to boundary edges and 0 to intra-region edges. In [9] this generalized labeling is used in the context of blocking flow algorithm but not within push-relabel.

2 Mincut and Push-Relabel

We will be solving MINCUT problem by finding a maximum preflow[2]. In this section, we give basic definitions and introduce the push-relabel framework [5].

By a *network* we call the tuple $G = (V, E, s, t, c, e)$, where V is a set of vertices; $E \subset V \times V$, thus (V, E) is a directed graph; $s, t \in V$, $s \neq t$, are *source* and *sink*, respectively; $c\colon E \to \mathbb{N}_0$ is a capacity function; and $e\colon V \backslash \{s, t\} \to \mathbb{N}_0$ is an *excess* function. Excess can be equivalently represented as additional edges from the source, but we prefer this explicit form. For convenience we let $e(s) = \infty$ and $e(t) = 0$. We also denote $n = |V|$ and $m = |E|$.

For $X, Y \subset V$ we will denote $(X, Y) = E \cap (X \times Y)$. For $C \subset V$ such that $s \in C$, $t \notin C$, the set of edges $(C, \bar{C})$, with $\bar{C} = V \backslash C$ is called an *s-t cut*. The MINCUT problem is

$$\min \Big\{ \sum_{(u,v) \in (C, \bar{C})} c(u, v) + \sum_{v \in \bar{C}} e(v) \;\Big|\; C \subset V,\ s \in C,\ t \in \bar{C} \Big\}. \tag{1}$$

The objective is called the *cost* of the cut. Without a loss of generality, we assume that E is symmetric – if not, the missing edges are added and assigned zero capacity.

A *preflow* in G is a function $f\colon E \to \mathbb{Z}$ satisfying the following constraints:

$$f(u, v) \leq c(u, v) \quad \forall (u, v) \in E \quad \text{(capacity constraint)}, \tag{2a}$$

$$f(u, v) = -f(u, v) \quad \forall (u, v) \in E \quad \text{(antisymmetry)}, \tag{2b}$$

$$e(v) + \sum_{u \,|\, (u,v) \in E} f(u, v) \geq 0 \quad \forall v \in V \quad \text{(preflow constraint)}. \tag{2c}$$

A *residual network* w.r.t. preflow f is a network $G_f = (V, E, s, t, c_f, e_f)$ with the capacity and excess functions given by

$$c_f = c - f, \tag{3a}$$

$$e_f(v) = e(v) + \sum_{u \,|\, (u,v) \in E} f(u, v), \quad \forall v \in V \backslash \{t\}. \tag{3b}$$

---

[2] A maximum preflow can be completed to a maximum flow using flow decomposition, in $O(m \log m)$ time. Because we are primarily interested in the minimum cut, we do not consider this step or whether it can be distributed.

By constraints (2) it is $c_f \geq 0$ and $e_f \geq 0$. The costs of all s-t cuts differ in G and G_f by a constant called the *flow value*, $|f| = \sum_{u \mid (u,t) \in E} f(u,t)$. Network G_f is thus up to a constant *equivalent* to network G and $|f|$ is a trivial lower bound on the cost of a cut. Dual to MINCUT is the problem of maximizing this lower bound, *i.e.* finding a maximum preflow:

$$\max_f |f| \quad \text{s.t. constraints (2).} \tag{4}$$

We say that $w \in V$ is *reachable* from $v \in V$ in network G if there is a path (possibly of length 0) from v to w composed of edges with strictly positive capacities. This relation is denoted by $v \to w$. If w is not reachable from v we write $v \not\to w$. For any $X, Y \subset V$, we write $X \to Y$ if there exist $x \in X$, $y \in Y$ such that $x \to y$. Otherwise we write $X \not\to Y$.

A preflow f is maximum iff $\{v \mid e(v) > 0\} \not\to t$ in G_f. In that case the cut $(\bar{T}, T)$ with $T = \{v \in V \mid v \to t \text{ in } G_f\}$ has value 0 in G_f. Because all cuts are non-negative it is a minimum cut.

A *Distance* function $d^*\colon V \to \mathbb{N}_0$ in G assigns to $v \in V$ the length of the shortest path from v to t, or n if no such path exists. A shortest path cannot have loops, thus its length is not greater than $n-1$. Let us denote $d^\infty = n$.

A *labeling* $d\colon V \to \{0, \dots, d^\infty\}$ is *valid* in G if $d(t) = 0$ and $d(u) \leq d(v) + 1$ for all $(u,v) \in E$ such that $c(u,v) > 0$. Any valid labeling is a lower bound on the distance d^* in G. Not every lower bound is a valid labeling. A vertex v is called *active* w.r.t. (f, d) if $e_f(v) > 0$ and $d(v) < d^\infty$.

All algorithms in this paper will use the following common initialization.

Procedure Init

```
1 f := preflow saturating all ({s}, V) edges; G := G_f; f := 0;
2 d := 0, d(s) := d^∞;
```

The generic push-relabel algorithm [5] starts with `Init` and applies the following `Push` and `Relabel` operations while possible:

- `Push`(u,v) is applicable if u is active and $c_f(u,v) > 0$ and $d(u) = d(v) + 1$. The operation increases $f(u,v)$ by Δ and decreases $f(v,u)$ by Δ, where $\Delta = \min(e_f(u), c_f(u,v))$.
- `Relabel`(u) is applicable if u is active and $\forall v \mid (u,v) \in E,\ c_f(u,v) > 0$ it is $d(u) \leq d(v)$. It sets $d(u) := \min\big(d^\infty, \min\{d(v)+1 \mid (u,v) \in E,\ c_f(u,v) > 0\}\big)$.

If u is active then either `Push` or `Relabel` operation is applicable to u. The algorithm preserves validity of labeling and stops when there are no active nodes. Then for any u such that $e_f(u) > 0$, we have $d(u) = d^\infty$ and therefore $d^*(u) = d^\infty$ and $u \not\to t$ in G_f, so f is a maximum preflow.

3 Region Discharge Revisited

We now review the approach of Delong and Boykov [7] and reformulate it for the case of a fixed graph partition. We then describe generic sequential and parallel algorithms which can be applied with both push-relabel and augmenting path approaches.

Delong and Boykov [7] introduce the following operation. The *discharge* of a *region* $R \subset V \backslash \{s,t\}$ applies `Push` and `Relabel` to $v \in R$ until there are no active vertices left in R. This localizes computations to R and its *boundary*, defined as

$$B^R = \{w \mid \exists u \in R \ (u,w) \in E, w \notin R, \ w \neq s,t\}. \tag{5}$$

When a `Push` is applied to an edge $(v,w) \in (R, B^R)$, the flow is sent out of the region. We say that two regions $R_1, R_2 \subset V \backslash \{s,t\}$ *interact* if $(R_1, R_2) \neq \emptyset$. Discharges of non-interacting regions can be performed in parallel since the computations in them do not share the data. The algorithm proposed in [7] repeats the following steps until there are no active vertices in V:

1. Select several non-interacting regions, containing active vertices.
2. Discharge the selected regions in parallel, applying region-gap and region-relabel heuristics[3].
3. Apply global gap heuristic.

While the regions in [7] are selected dynamically in each iteration trying to divide the work evenly between CPUs and cover the most of the active nodes, we restrict ourselves to a fixed collection of regions $(R_k)_{k=1}^K$ forming a partition of $V \backslash \{s,t\}$ and let each region-discharge to work on its own separate subnetwork. We define a *region network* $G^R = (V^R, E^R, s, t, c^R, e^R)$, where $V^R = R \cup B^R \cup \{s,t\}$; $E^R = (R \cup \{s,t\}, R \cup \{s,t\}) \cup (R, B^R) \cup (B^R, R)$; $c^R(u,v) = c(u,v)$ if $(u,v) \in E^R \backslash (B^R, R)$ and 0 otherwise; $e^R = e|_{R \cup \{s,t\}}$ (the restriction of function e to its subdomain $R \cup \{s,t\}$). This network is illustrated in Fig. 1(a). Note that the capacities of edges coming from the boundary, (B^R, R), are set to zero. Indeed, these edges belong to a neighboring region network. The region discharge operation of [7], which we refer to as Push-relabel Region Discharge (PRD), can now be defined as follows.

Procedure $(f,d) = \text{PRD}(G^R, d)$

/* assume $d \colon V^R \to \{0, \ldots, d^\infty\}$ valid in G^R */

while $\exists v \in R$ *active* **do**

  apply `Push` or `Relabel` to v; /* changes f and d */

  apply region gap heuristic (see [7], [1, sec.5]); /* optional */

Generic Region Discharge Algorithms. We give a sequential and a parallel algorithms in Alg. 1 and Alg. 2, resp. The later allows to discharge interacting regions in parallel, resolving conflicts in the flow similar to the asynchronous parallel push-relabel [5]. These two algorithms are generic, taking a black-box `Discharge` function. In the case `Discharge` is PRD the sequential and parallel algorithms are implementing the push-relabel approach and will be referred to as

[3] All heuristics (global-gap, region-gap, region-relabel) serve to improve the distance estimate. Details in [10,7,1]. They are very important in practice, but do not affect theoretical properties.

S-PRD and P-PRD respectively. S-PRD is a sequential variant of [7] and P-PRD is a novel variant, based on results of [5] and [7].

Algorithm 1: Sequential Region Discharge

Init;
while *there are active vertices* **do** /* a sweep */
  for $k = 1, \dots K$ **do**
    Construct G^{R_k} from G;
    $(f', d') :=$ Discharge$(G^{R_k}, d|_{V^{R_k}})$;
    $G := G_{f'}$; /* apply f' to G */
    $d|_{R_k} := d'|_{R_k}$; /* update labels */
    apply global gap heuristic (see [10], [1, sec.5]); /* optional */

Algorithm 2: Parallel Region Discharge

Init;
while *there are active vertices* **do** /* a sweep */
  $(f'_k, d'_k) :=$ Discharge$(G^{R_k}, d|_{V^{R_k}})$ $\forall k$; /* in parallel */
  $d'|_{R_k} := d'_k|_{R_k}$ $\forall k$; /* fuse labels */
  $\alpha(u,v) := [\![d'(u) \leq d'(v) + 1]\!]$ $\forall (u,v) \in (\mathcal{B}, \mathcal{B})$; /* valid pairs */
  /* fuse flow */
  $f'(u,v) := \begin{cases} \alpha(v,u) f'_k(u,v) + \alpha(u,v) f'_j(u,v) & \text{if } (u,v) \in (R_k, R_j) \\ f'_k(u,v) & \text{if } (u,v) \in (R_k, R_k) \end{cases}$;
  $G := G_{f'}$; /* apply f' to G */
  $d := d'$; /* update labels */
  global gap heuristic; /* optional */

We prove in [1] that both S-PRD and P-PRD terminate with a valid labeling in at most $2n^2$ sweeps. Parallel variants of push-relabel [11] have the same bound on the number of sweeps, so the mentioned result is not very surprising. On the other hand, the analysis in [1] allows for more general Discharge functions. We also show in [1] an example, which takes $O(n^2)$ sweeps to terminate for a partition into two regions, interacting over two edges. Hence the bound is tight.

4 Augmented Path Region Discharge

We will now use the same setup of the problem distribution, but replace the discharge operation and the labeling function. Because this is our main contribution, it is presented in full detail.

4.1 New Distance Function

Let the *boundary* w.r.t. partition $(R_k)_{k=1}^K$ be the set $\mathcal{B} = \bigcup_k B^{R_k}$. The *region distance* $d^{*\mathcal{B}}(u)$ in G is the minimal number of inter-region edges contained in a path from u to t, or $|\mathcal{B}|$ if no such path exists:

$$d^{*\mathcal{B}}(u) = \begin{cases} \min\limits_{P=((u,u_1),\dots,(u_r,t))} |P \cap (\mathcal{B}, \mathcal{B})| & \text{if } u \to t, \\ |\mathcal{B}| & \text{if } u \not\to t. \end{cases} \quad (6)$$

This distance corresponds well to the number of region discharge operations required to transfer the excess to the sink.

Statement 1. If $u \to t$ then $d^{*\mathcal{B}}(u) < |\mathcal{B}|$.

Proof. Let P be a path from u to t given as a sequence of edges. If P contains a loop then it can be removed from P and $|P \cap (\mathcal{B}, \mathcal{B})|$ will not increase. A path without loops goes through each vertex at most once. For $\mathcal{B} \subset V$ there is at most $|\mathcal{B}| - 1$ edges in the path which have both endpoints in $\mathcal{B}$. □

We now let $d^\infty = |\mathcal{B}|$ and redefine a valid labeling w.r.t. to the new distance. A labeling $d\colon V \to \{0, \dots, d^\infty\}$ is *valid* in G if $d(t) = 0$ and for all $(u, v) \in E$ such that $c(u, v) > 0$:

$$d(u) \le d(v) + 1 \qquad \text{if } (u, v) \in (\mathcal{B}, \mathcal{B}), \tag{7}$$

$$d(u) \le d(v) \qquad \text{if } (u, v) \notin (\mathcal{B}, \mathcal{B}). \tag{8}$$

Statement 2. A valid labeling d is a lower bound on $d^{*\mathcal{B}}$.

Proof. If $u \nrightarrow t$ then $d(u) \le d^{*\mathcal{B}}$. Otherwise, let $P = ((u, v_1), \dots, (v_l, t))$ be a shortest path w.r.t. $d^{*\mathcal{B}}$, *i.e.* $d^{*\mathcal{B}}(u) = |P \cap (\mathcal{B}, \mathcal{B})|$. Applying the validity property to each edge in this path, we have $d(u) \le d(t) + |P \cap (\mathcal{B}, \mathcal{B})| = d^{*\mathcal{B}}(u)$. □

4.2 New Region Discharge

In this subsection, reachability relations "$\to$", "$\nrightarrow$", residual paths, and labeling validity will be understood in the region network G^R or its residual G^R_f.

The new `Discharge` operation, called Augmented Path Region Discharge (ARD), works as follows. It first pushes excess to the sink along augmenting paths inside the network G^R. When it is no longer possible, it continues to augment paths to nodes in the region boundary, B^R, in the order of their increasing labels. This is represented by the sequence of nested sets $T_0 = \{t\}$, $T_1 = \{t\} \cup \{v \in B^R \mid d(v) = 0\}$, …, $T_{d^\infty} = \{t\} \cup \{v \in B^R \mid d(v) < d^\infty\}$. Set T_k is the destination of augmentations in stage k. As we prove below, in stage $k > 0$ residual paths may exist only to the set $T_k \backslash T_{k-1} = \{v \mid d(v) = k - 1\}$. Algorithm 1 and 2 with this new discharge operation will be referred to as S-ARD and P-ARD, respectively.

Procedure $(f, d) = \text{ARD}(G^R, d)$

`/*` assume $d\colon V^R \to \{0, \dots, d^\infty\}$ valid in G^R `*/`
for $k = 0, 1, \dots, d^\infty$ **do** `/*` stage k `*/`
  $T_k = \{t\} \cup \{v \in B^R \mid d(v) < k\}$
  `/* Augment(`R, T_k`) */`
  while $\exists$ *a residual path* $(v_0 \in R, \dots, v_l \in T_k)$, $e_f(v_0) > 0$ **do**
    augment $\Delta = \min(e_f(v_0), \min_i c_f(v_{i-1}, v_i))$ along the path.
`/* Region-relabel */`

$$d(u) := \begin{cases} \min\{k \mid u \to T_k\} & u \in R, u \to T_{d^\infty}, \\ d^\infty & u \in R, u \nrightarrow T_{d^\infty}, \\ d(u) & u \in B^R. \end{cases}$$

The labels on the boundary, $d|_{B^R}$, remain fixed during the algorithm and the labels $d|_R$ inside the region do not participate in augmentations and therefore are updated only in the end.

We claim that ARD terminates with no active nodes inside the region, preserves validity and monotonicity of the labeling, and pushes flow from higher labels to lower labels w.r.t. the new labeling. These properties will be required to prove finite termination and correctness of S-ARD. Before we prove them (Statement 6) we need the following intermediate results:

- Properties of the network G_f^R maintained by the algorithm (Statement 3, Corollaries 1 and 2).
- Properties of valid labellings in the network G_f^R (Statement 4).
- Properties of the labeling constructed by region-relabel (line 5 of `ARD`) in the network G_f^R (Statement 5).

Lemma 1. Let $X, Y \subset V^R$, $X \cap Y = \emptyset$, $X \not\rightarrow Y$. Then $X \not\rightarrow Y$ is preserved after i) augmenting a path $(x, \dots, v)$ with $x \in X$ and $v \in V^R$; ii) augmenting a path $(v, \dots, y)$ with $y \in Y$ and $v \in V^R$.

Proof. Let $\mathcal{X}$ be the set of vertices reachable from X. Let $\mathcal{Y}$ be the set of vertices from which Y is reachable. Clearly $\mathcal{X} \cap \mathcal{Y} = \emptyset$, otherwise $X \rightarrow Y$. We have that $(\mathcal{X}, \bar{\mathcal{X}})$ is a cut separating X and Y and having all edge capacities zero. Any residual path starting in X or ending in Y cannot cross the cut and its augmentation change the edges of the cut. Hence, X and Y will stay separated. □

Statement 3. Let $v \in V^R$ and $v \not\rightarrow T_a$ in G_f in the beginning of stage k_0, where $a, k_0 \in \{0, 1, \dots d^\infty\}$. Then $v \not\rightarrow T_a$ holds until the end of the algorithm.

Proof. We need to show that $v \not\rightarrow T_a$ is not affected by augmentations performed by the algorithm. If $k_0 \leq a$, we first prove $v \not\rightarrow T_a$ holds during stages $k_0 \leq k \leq a$. Consider augmentation of a path $(u_0, u_1, \dots, u_l)$, $u_0 \in R$, $u_l \in T_k \subset T_a$, $e_f(u_0) > 0$. Assume $v \not\rightarrow T_a$ before augmentation. By Lemma 1 with $X = \{v\}$, $Y = T_a$ (noting that $X \not\rightarrow Y$ and the augmenting path ends in Y), after the augmentation $v \not\rightarrow T_a$. By induction, it holds till the end of stage a and hence in the beginning of stage $a + 1$.

We can assume now that $k_0 > a$. Let $A = \{u \in R \,|\, e_f(u) > 0\}$. At the end of stage $k_0 - 1$ we have $A \not\rightarrow T_{k_0-1} \supset T_a$ by construction. Consider augmentation in stage k_0 on a path $(u_0, u_1 \dots, u_l)$, $u_0 \in R$, $u_l \in T_{k_0}$, $e_f(u_0) > 0$. By construction, $u_0 \in A$. Assume $\{v\} \cup A \not\rightarrow T_a$ before augmentation. Apply Lemma 1 with $X = \{v\} \cup A$, $Y = T_a$ (we have $X \not\rightarrow Y$ and $u_0 \in A \subset X$). After augmentation it is $X \not\rightarrow T_a$. By induction, $X \not\rightarrow T_a$ till the end of stage k_0. By induction on stages, $v \not\rightarrow T_a$ until the end of the algorithm. □

Corollary 1. If $w \in B^R$ then $w \not\rightarrow T_{d(w)}$ throughout the algorithm.

Proof. At initialization, it is fulfilled by construction of G^R due to $c^R(B^R, R) = 0$. It holds then during the algorithm by Statement 3. □

In particular, we have $B^R \not\rightarrow t$ during the algorithm.

Corollary 2. Let $(u, v_1 \dots v_l, w)$ be a residual path in G_f^R from $u \in R$ to $w \in B^R$ and let $v_r \in B^R$ for some r. Then $d(v_r) \leq d(w)$.

Proof. We have $v_r \nrightarrow T_{v_r}$. Suppose $d(w) < d(v_r)$, then $w \in T_{v_r}$ and because $v_r \to w$ it is $v_r \to T_{v_r}$ which is a contradiction. □

Statement 4. Let d be a valid labeling, $d(u) \geq 1$, $u \in R$. Then $u \nrightarrow T_{d(u)-1}$.

Proof. Suppose $u \to T_0$. Then there exist a residual path $(u, v_1 \dots v_l, t)$, $v_i \in R$ (by Corollary 1 it cannot happen that $v_i \in B^R$). By validity of d we have $d(u) \leq d(v_1) \leq \dots \leq d(v_l) \leq d(t) = 0$, which is a contradiction.

Suppose $d(u) > 1$ and $u \to T_{d(u)-1}$. Because $u \nrightarrow T_0$, it must be that $u \to w$, $w \in B^R$ and $d(w) < d(u) - 1$. Let $(v_0 \dots v_l)$ be a residual path with $v_0 = u$ and $v_l = w$. Let r be the minimal number such that $v_r \in B^R$. By validity of d we have $d(u) \leq d(v_1) \leq \dots \leq d(v_{r-1}) \leq d(v_r) + 1$. By corollary 2 we have $d(v_r) \leq d(w)$, hence $d(u) \leq d(w) + 1$ which is a contradiction. □

Statement 5. For d computed on line 5 and any $u \in R$ it holds:

1. d is valid;
2. $u \nrightarrow T_a \Leftrightarrow d(u) \geq a + 1$.

Proof. 1. Let $(u, v) \in E^R$ and $c(u, v) > 0$. Clearly $u \to v$. Consider four cases:
   - case $u \in R$, $v \in B^R$: Then $u \to T_{d(v)+1}$, hence $d(u) \leq d(v) + 1$.
   - case $u \in R$, $v \in R$: If $v \nrightarrow T_{d^\infty}$ then $d(v) = d^\infty$ and $d(u) \leq d(v)$. If $v \to T_{d^\infty}$, then $d(v) = \min\{k \mid v \to T_k\}$. Let $k = d(v)$, then $v \to T_k$ and $u \to T_k$, therefore $d(u) \leq k = d(v)$.
   - case $u \in B^R$, $v \in R$: By Corollary 1, $u \nrightarrow T_{d(u)}$. Because $u \to v$, it is $v \nrightarrow T_{d(u)}$, therefore $d(v) \geq d(u) + 1$ and $d(u) \leq d(v) - 1 \leq d(v) + 1$.
   - case when $u = t$ or $v = t$ is trivial.

2. The "$\Leftarrow$" direction follows by Statement 4 applied to d, which is a valid labeling. The "$\Rightarrow$" direction: we have $u \nrightarrow T_a$ and $d(u) \geq \min\{k \mid u \to T_k\} = \min\{k > a \mid u \to T_k\} \geq a + 1$. □

Statement 6 (Properties of ARD)**.** Let d be a valid labeling in G^R. The output (f', d') of ARD satisfies:

1. There are no active vertices in R w.r.t. (f', d'); (*optimality*)
2. $d' \geq d$, $d'|_{B^R} = d|_{B^R}$; (*labeling monotonicity*)
3. d' is valid in $G^R_{f'}$; (*labeling validity*)
4. f' is a sum of path flows, where each path is from a vertex $u \in R$ to a vertex $v \in \{t\} \cup B^R$ and it is $d'(u) > d(v)$ if $v \in B^R$. (*flow direction*)

Proof. 1. In the last stage, the algorithm augments all paths to T_{d^∞}. After this augmentation a vertex $u \in R$ either has excess 0 or there is no residual path to T_{d^∞} and hence $d'(u) = d^\infty$ by construction.

2. For $d(u) = 0$, we trivially have $d'(u) \geq d(u)$. Let $d(u) = a + 1 > 0$. By Statement 4, $u \nrightarrow T_a$ in G^R and it holds also in G^R_f by Statement 3. From Statement 5.2, we conclude that $d'(u) \geq a + 1 = d(u)$. The equality $d'|_{B^R} = d|_{B^R}$ is by construction.
3. Proven by Statement 5.1.
4. Consider a path from u to $v \in B^R$, augmented in stage $k > 0$. It follows that $k = d(v)+1$. At the beginning of stage k it is $u \nrightarrow T_{k-1}$. By Statement 3, this is preserved till the end of the algorithm. By Statement 5.2, $d'(u) \geq k = d(v) + 1 > d(v)$. □

4.3 Complexity of the Sequential ARD

Let us first verify that the labeling in S-ARD is globally valid.

Statement 7. For a labeling d valid in G and $(f', d') = \texttt{ARD}(G^R, d)$, the extension of d' to V defined by $d'|_{\bar{R}} = d|_{\bar{R}}$ is valid in $G_{f'}$.

Proof. Statement 5 established validity of d' in $G^R_{f'}$. For edges $(u, v) \in (V\backslash R, V\backslash R)$ labeling d' coincides with d and $f'(u, v) = 0$. It remains to verify validity on edges $(v, u) \in (B^R, R)$ in the case $c^R_f(v, u) = 0$ and $c_f(v, u) > 0$. Because $0 = c^R_f(v, u) = c^R(v, u) - f(v, u) = -f(v, u)$, we have $c_f(v, u) = c(v, u)$. Since d was valid in G, $d(v) \leq d(u) + 1$. The new labeling d' satisfies $d'(u) \geq d(u)$ and $d'(v) = d(v)$. It follows that $d'(v) = d(v) \leq d(u) + 1 \leq d'(u) + 1$. Hence d' is valid in $G_{f'}$. □

Theorem 1. The sequential ARD terminates in at most $2|\mathcal{B}|^2 + 1$ sweeps.

Proof. The value of $d(v)$ does not exceed $|\mathcal{B}|$ and d is non-decreasing. The total increase of $d|_\mathcal{B}$ during the algorithm is at most $|\mathcal{B}|^2$.

After the first sweep, active vertices are only in $\mathcal{B}$. Indeed, discharging region R_k makes all vertices $v \in R_k$ inactive and only vertices in $\mathcal{B}$ may become active. So by the end of the sweep, all vertices $V\backslash\mathcal{B}$ are inactive.

Let us introduce the quantity

$$\Phi = \max\{d(v) \mid v \in \mathcal{B},\ v \text{ is active in } G\}. \tag{9}$$

We will prove the following two cases for each sweep after the first one:

1. If $d|_\mathcal{B}$ is increased then the increase in Φ is no more than total increase in $d|_\mathcal{B}$. Consider discharge of R_k. Let Φ be the value before ARD on R_k and Φ' the value after. Let $\Phi' = d'(v)$. It must be that v is active in G'. If $v \notin V^R$, then $d(v) = d'(v)$ and $e(v) = e_{f'}(v)$ so $\Phi \geq d(v) = \Phi'$.
   Let $v \in V^R$. After the discharge, vertices in R_k are inactive, so $v \in B_k$ and it is $d'(v) = d(v)$. If v was active in G then $\Phi \geq d(v)$ and we have $\Phi' - \Phi \leq d'(v) - d(v) = 0$. If v was not active in G, there must exist an augmenting path from a vertex v_0 to v such that $v_0 \in R_k \cap \mathcal{B}$ was active in G. For this path, the flow direction property implies $d'(v_0) \geq d(v)$. We now have $\Phi' - \Phi \leq d'(v) - d(v_0) = d(v) - d(v_0) \leq d'(v_0) - d(v_0) \leq \sum_{v \in R_k \cap \mathcal{B}}[d'(v) - d(v)]$. Summing over all regions, we get the result.
2. If $d|_\mathcal{B}$ is not increased then Φ is decreased at least by 1. We have $d' = d$. Let us consider the set of vertices having the highest active label or above, $H = \{v \mid d(v) \geq \Phi\}$. These vertices do not receive flow during all discharge operations due to the flow direction property. After the discharge of R_k there are no active vertices left in $R_k \cap H$ (property 6.1). After the full sweep, there are no active vertices in H.

In the worst case, starting from sweep 2, Φ can increase by one $|\mathcal{B}|^2$ times and decrease by one $|\mathcal{B}^2|$ times. In at most $2|\mathcal{B}|^2 + 1$ sweeps, there are no active vertices left. □

On termination we have that the labeling is valid and there are no active vertices in G. The proof that P-ARD terminates is similar and is given in [1].

5 Experiments

We tested the algorithms on synthetic and real problems. The machine had Intel Core 2 Quad CPU@2.66Hz, 4GB memory, Windows XP 32bit and Microsoft VC compiler. All tested algorithms are sequential, 32bit and use only one core of the CPU. The memory limit for the algorithms is 2GB.

As a baseline we used augmenting path implementation [2] v3.0 (**BK**) and the highest level push-relabel implementation [10] v3.6 (**HIPR**α, where α is a parameter denoting frequency of global relabels, 0.5 is the default value).

ARD was implemented[4] using BK as a core solver. PRD is based on our reimplementation of the highest level push-relabel for the case of a given labeling on the boundary. This reimplementation (denoted **HPR**) uses linked list of buckets (rather than array) to achieve the time and space complexity independent of n and otherwise is similar to HIPR. The sequential Alg. 1 for each region loads and saves all the internal data of the core solver, so that discharge is always warm-started. Please see [1] for details of implementation and more experimental results.

5.1 Synthetic Instances

We generated simple synthetic 2D grid problems with a regular connectivity structure. Fig. 1(b) shows an example of such a network. Nodes are arranged into 2D grid and edges are added at the the following relative displacements: $(0, 1)$, $(1, 0)$, $(1, 2)$, $(2, 1)$, so the number of edges incident to a node far enough from the boundary (*connectivity*) is equal to 8. Each node is given an integer excess/deficit distributed uniformly in the interval $[-500\ 500]$. A positive number means a source link and a negative number a sink link. All edges in the graph, except of terminal links, are assigned a constant capacity, called *strength*. The network is partitioned into regions by slicing it in s equal parts in both dimensions. Let us first look at the dependence on the strength, shown in Fig. 1(c). Problems with small strength are easy because they are very local – long augmentation paths do not occur. For problems with large strength long paths needs to be augmented. However, finding them is easy because bottlenecks are unlikely. Therefore BK and S-ARD have a maximum in the computation time somewhere in the middle. It is more difficult to transfer the flow over long distances for push-relabel algorithms. This is where the global relabel heuristic becomes efficient and HIPR0.5 outperforms HIPR0. The region-relabel heuristic of S-PRD allows it to outperform other push-relabel variants.

As the function of the number of regions (Fig. 2(a)), both the number of sweeps and the computation time grow slowly.

As the function of the problem size (Fig. 2(b)), computation efforts of all algorithms grow proportionally. However, the number of sweeps shows different asymptotes. It is almost constant for S-ARD but grows significantly for S-PRD.

[4] Implementations are available at `cmp.felk.cvut.cz/~shekhovt/d_maxflow`.

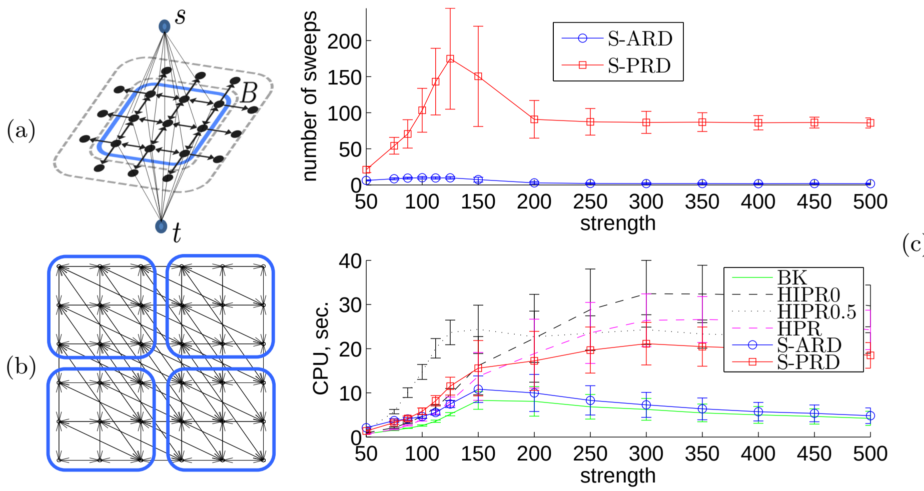

Fig. 1. (a) Region Network. (b) Example of a synthetic problem: a network of size 6×6, connectivity 8, partition into 4 regions. The source and sink are not shown. (c) Dependence on the interaction strength for size 1000×1000, connectivity 8 and 4 regions. Plots show the mean values over 100 random samples and intervals containing 70% of the samples.

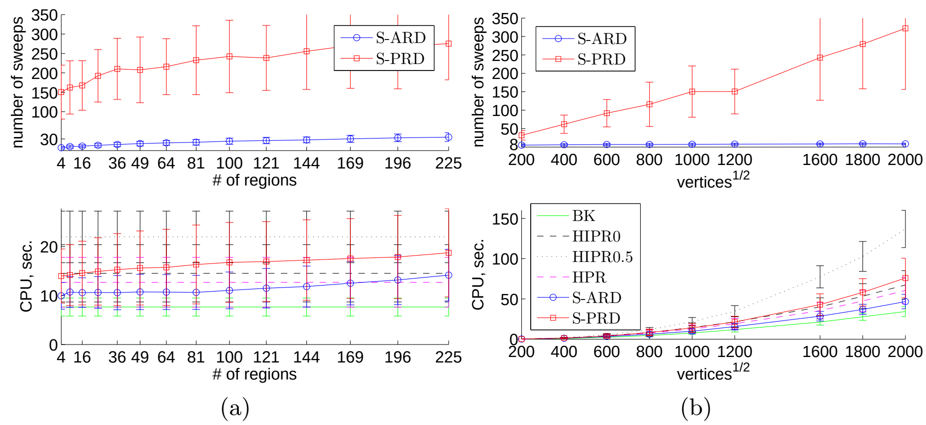

Fig. 2. (a) Dependence on the number of regions, for size 1000×1000, strength 150. (b) Dependence on the problem size, for strength 150, 4 regions.

5.2 Real Instances

We tested our algorithms on the MAXFLOW problem instances published by the Computer Vision Research Group at the University of Western Ontario (`http://vision.csd.uwo.ca/maxflow-data/`). The data consists of typical maxflow problems in computer vision, graphics, and biomedical image analysis, including 2D, 3D and 4D grids of various connectivity. The results are presented in Table 1.

We select the regions by slicing the problems in 4 parts in each dimension: into 16 regions for 2D BVZ grids and into 64 regions for 3D segmentation instances.

Problems KZ2 are not regular grids, so we sliced them into 16 regions just by the node number. The same we did for the multiview LB06 instances, for which we do not know the grid layout. In 3D segmentation instances the arcs which are reverse of each other are spread in the file. Because we did not match them, we had to create parallel arcs in the graph (multigraph). This is seen, *e.g.* in `babyface.n26c100`, which is 26-connected, but we construct a multigraph with average node degree of 49. For some other instances, however, this is not visible because there are many zero arcs.

Table 1. Real instances. CPU – the time spent purely for computation, excluding the time for parsing, construction and disk I/O. The total time to solve the problem is not shown. K – number of regions. RAM – memory taken by the solver; for BK in the case it exceeds 2GB limit, the expected required memory; for streaming solvers the sum of shared and region memory. I/O – total bytes read or written to disk.

problem		BK	HI-PR0	HI-PR0.5	HPR	S-ARD			S-PRD		
name		CPU	CPU	CPU	CPU	CPU	sweeps	K	CPU	sweeps	K
$n(10^6)$	m/n	RAM	RAM	RAM	RAM	RAM		I/O	RAM		I/O
stereo											
BVZ-sawtooth(20)		0.68s	3.0s	7.7s	3.8s	0.68s	6	16	2.7s	26	16
0.2	4.0	14MB			17MB	0.3+0.9MB		91MB	0.7+1.1MB		0.6GB
BVZ-tsukuba(16)		0.36s	1.9s	4.9s	2.6s	0.40s	5	16	1.7s	23	16
0.1	4.0	9.7MB			11MB	0.2+0.6MB		55MB	0.5+0.7MB		349MB
BVZ-venus(22)		1.2s	5.7s	15s	6.2s	1.6s	6	16	5.8s	29	16
0.2	4.0	15MB			17MB	0.3+0.9MB		94MB	0.7+1.1MB		0.8GB
KZ2-sawtooth(20)		1.8s	7.1s	22s	6.1s	2.2s	6	16	6.0s	21	16
0.3	5.8	33MB			36MB	1.2+2.0MB		212MB	1.5+2.3MB		1.1GB
KZ2-tsukuba(16)		1.1s	5.3s	20s	4.4s	1.8s	6	16	5.4s	15	16
0.2	5.9	23MB			25MB	1.1+1.4MB		148MB	1.1+1.6MB		0.5GB
KZ2-venus(22)		2.8s	13s	39s	10s	4.0s	7	16	12s	29	16
0.3	5.8	34MB			37MB	1.2+2.1MB		255MB	1.5+2.4MB		1.5GB
multiview											
BL06-camel-lrg		81s				116s	11	16	308s	418	16
18.9	4.0	1.6GB				19+116MB		25GB	86+122MB		0.6TB
BL06-camel-med		25s	29s	77s	59s	36s	12	16	118s	227	16
9.7	4.0	0.8GB			1.0GB	13+60MB		13GB	46+63MB		225GB
BL06-gargoyle-lrg		245s			91s	191s	20	16	318s	354	16
17.2	4.0	1.5GB			1.7GB	23+106MB		33GB	82+112MB		0.8TB
BL06-gargoyle-med		115s	17s	58s	37s	91s	14	16	143s	340	16
8.8	4.0	0.8GB			0.9GB	15+55MB		12GB	44+58MB		235GB
surface											
LB07-bunny-lrg						16min	6	64	416s	43	64
49.5	6.0	5.7GB				49+101MB		34GB	226+99MB		276GB
LB07-bunny-med		1.6s	20s	41s	26s	20s	8	64	16s	25	64
6.3	6.0	0.7GB			0.8GB	14+14MB		4.1GB	34+13MB		24GB
segm											
liver.n26c100		12s	26s	28s	39s	26s	15	64	35s	98	64
4.2	11.1	0.8GB			0.7GB	18+15MB		11GB	30+14MB		66GB
liver.n6c100		15s	30s	34s	44s	25s	17	64	32s	94	64
4.2	10.5	0.8GB			0.7GB	16+14MB		11GB	29+13MB		70GB
babyface.n26c100						264s	36	64	262s	116	64
5.1	49.0	3.8GB				165+72MB		95GB	180+57MB		0.6TB
babyface.n6c100		13s	71s	65s	87s	32s	17	64	74s	191	64
5.1	11.5	1.0GB			0.9GB	22+19MB		17GB	37+17MB		189GB
adhead.n26c100						185s	16	64	269s	129	64
12.6	31.6	6.3GB				154+106MB		70GB	196+86MB		0.8TB
adhead.n6c100						48s	13	64	121s	165	64
12.6	11.7	2.5GB				35+44MB		29GB	77+39MB		354GB
bone.n26c100						32s	15	64	68s	124	64
7.8	32.4	4.0GB				122+79MB		31GB	147+63MB		321GB
bone.n6c10		7.7s	5.7s	17s	12s	7.8s	10	64	37s	195	64
7.8	11.5	1.5GB			1.4GB	27+28MB		10GB	52+25MB		188GB

Continued on next page

Table 1 – continued from previous page.

bone.n6c100		9.1s	9.1s	22s	14s	9.8s	11	64	23s	65	64
7.8	11.6	1.6GB			1.5GB	27+28MB		12GB	52+25MB		104GB
abdomen_long.n6c10						179s	11	64		> 35	64
144.4	11.8	29GB				170+497MB		196GB			>1TB
abdomen_short.n6c10						82s	11	64			64
144.4	11.8	29GB				170+497MB		138GB			

6 Conclusion

We have developed a new distributed algorithm for MINCUT problem on sparse graphs and proved an $O(|\mathcal{B}|^2)$ bound on the number of sweeps. Both in theory and practice (randomized tests) the required number of sweeps is asymptotically better than for a variant of parallel push-relabel. Experiments on real instances showed that S-ARD, while sometimes doing more computations than S-PRD or BK, uses significantly fewer disk operations.

We proposed a sequential and a parallel version of the algorithm. The best practical solution could be a combination of the two, depending on the usage mode and hardware (several CPUs, several network computers, sequential with storage on Solid State Drive, using GPU for region discharge, *etc.*).

There is the following simple way how to allow region overlaps in our framework. A sequential algorithm can keep 2 regions in memory at a time and alternate between them until both are discharged. With PRD this is efficiently equivalent to discharging twice larger regions with a 1/2 overlap and may significantly decrease the number of sweeps required.

References

1. Shekhovtsov, A., Hlavac, V.: A distributed mincut/maxflow algorithm combining path augmentation and push-relabel. Research Report K333–43/11, CTU–CMP–2011–03, Czech Technical University in Prague (2011)
2. Boykov, Y., Kolmogorov, V.: An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. In: PAMI. Volume 26. (2004)
3. Liu, J., Sun, J.: Parallel graph-cuts by adaptive bottom-up merging. In: CVPR. (2010)
4. Strandmark, P., Kahl, F.: Parallel and distributed graph cuts by dual decomposition. In: CVPR. (2010)
5. Goldberg, A.V., Tarjan, R.E.: A new approach to the maximum flow problem. Journal of the ACM **35** (1988)
6. Goldberg, A.V.: Processor-efficient implementation of a maximum flow algorithm. Inf. Process. Lett. **38** (1991)
7. Delong, A., Boykov, Y.: A scalable graph-cut algorithm for N-D grids. In: CVPR. (2008)
8. Goldberg, A.V.: The partial augment–relabel algorithm for the maximum flow problem. In: Proceedings of the 16th annual European symposium on Algorithms. (2008)
9. Goldberg, A.V., Rao, S.: Beyond the flow decomposition barrier. J. ACM (1998)
10. Cherkassky, B.V., Goldberg, A.V.: On implementing push-relabel method for the maximum flow problem. Technical report (1994)
11. Goldberg, A.: Efficient graph algorithms for sequential and parallel computers. PhD thesis, Massachusetts Institute of Technology (1987)