MaxFlow Revisited: An Empirical Comparison of Maxflow Algorithms for Dense Vision Problems

Tanmay Verma
tanmay08054@iiitd.ac.in
IIIT-Delhi
Delhi, India

Dhruv Batra
dbatra@ttic.edu
TTI-Chicago
Chicago, USA

Abstract

Algorithms for finding the maximum amount of flow possible in a network (or max-flow) play a central role in computer vision problems. We present an empirical comparison of different max-flow algorithms on modern problems. Our problem instances arise from energy minimization problems in Object Category Segmentation, Image Deconvolution, Super Resolution, Texture Restoration, Character Completion and 3D Segmentation. We compare 14 different implementations and find that the most popularly used implementation of Kolmogorov [5] is no longer the fastest algorithm available, especially for dense graphs.

1 Introduction

Over the past two decades, algorithms for finding the maximum amount of flow possible in a network (or max-flow) have become the workhorses of modern computer vision and machine learning – from optimal (or provably-approximate) inference in sophisticated discrete models [6, 11, 27, 30, 32] to enabling real-time image processing [38, 39].

Perhaps the most prominent role of max-flow is due to the work of Hammer [23] and Kolmogorov and Zabih [27], who showed that a fairly large class of energy functions – sum of submodular functions on pairs of boolean variables – can be efficiently and *optimally* minimized via a reduction to max-flow. Max-flow also plays a crucial role in approximate minimization of energy functions with multi-label variables [4, 6], triplet or higher order terms [26, 27, 35, 37], global terms [30], and terms encoding label costs [11, 32].

Given the wide applicability, it is important to ask *which* max-flow algorithm should be used. There are numerous algorithms for max-flow with different asymptotic complexities and practical run-time behaviour. For an extensive list, we refer the reader to surveys in the literature [2, 7]. Broadly speaking, there are three main families of max-flow algorithms:

1. Augmenting-Path (AP) variants: algorithms [5, 13, 14, 17, 21] that maintain a valid flow during the algorithm, *i.e.* always satisfying the capacity and flow-conservation constraints.

2. Push-Relabel (PRL) variants: algorithms [8, 20] that maintain a *preflow*, *i.e.* satisfy the capacity constraints but may violate the conservation constraints to have flow excess at nodes (but never a flow deficit).
3. Hochbaum's Pseudoflow (HPF) variants: algorithms [7, 24] that maintain a *pseudoflow*, *i.e.* satisfy the capacity constrains but may violate the conservation constrains to allow flow excess and deficit at nodes.[1]

Boykov and Kolmogorov [5] compared AP and PRL algorithms on a number of computer vision problems, and found that their own AP algorithm was the fastest algorithm in practice, even though they could only prove a very loose asymptotic complexity bound of $O(n^2mC)$, where n is the number of nodes, m is the number of edges and C is the value of the max-flow.

Goal. The central thesis of this work is that since this comparison a decade ago, the models used in computer vision and the *kinds* of inference problems we solve have changed significantly. Specifically, while [5] only considered 4-connected grid MRFs, the models today involve high-order terms [26], long-range connections [31], hierarchical MRFs [28] and even global terms [30]. The effect of all these modifications is to make the underlying max-flow graph significantly denser, thus causing the complexity of the algorithm of [5] to become a concern. It is time to revisit this comparison.

Contribution. The goal of this paper is to compare the runtimes of different max-flow algorithms, to investigate if the conclusions of Boykov and Kolmogorov [5] are still valid for current-day *dense* problems, and find out which algorithm is most suited for modern vision problems. One key contribution of our study is that it includes recently proposed algorithms – Pseudoflow [7, 24] and Incremental Breadth-First-Search (IBFS) [21] – which were not developed at the time [5] was written, and thus were absent from their comparison. At a high-level, we hope that the results of our study guide practitioners in picking the correct implementation for their problems.

Scope of the Comparison. In order to perform a controlled thorough analysis, we need to set limits on the scope of our comparison. We restrict ourselves to exact sequential max-flow solvers evaluated on energy minimization problems arising in computer vision. Specifically, we do not consider approximate max-flow solvers [9, 19]. Perhaps more importantly, we do not compare multi-core implementations [10, 33], distributed implementations [22, 33], or GPU implementations [25, 40]. However, the scope of our comparison is wide enough to be useful to a large number of practitioners.

Paper Organization. Section 2 describes previous max-flow comparisons; Section 3 describes preliminaries and relevant background; Section 4 describes our experimental setup and comparisons; Section 5 concludes with a discussion about our findings.

2 Related Comparisons

Experimental comparisons of max-flow algorithms have typically been performed outside the scope of computer vision and machine learning [2, 3, 12]. To the best of our knowledge, most existing comparisons are lacking in one way or another. Cherkassky and Goldberg [8] compared Dinic's AP algorithm [13] with variants of PRL [20], but did not compare to HPF or the IBFS algorithms. Boykov and Kolmogorov [5] compared their own proposed algorithm BK with Dinic's algorithm and PRL but not HPF or IBFS. Chandran and Hochbaum [7]

[1] Interestingly, the key difference between Push-Relabel and Pseudoflow algorithms is not the concept of pseudoflow rather the admissibility of certain push schemes. See Section 3 for details.

compared BK, PRL and HPF but not IBFS. Goldberg *et al*. [21] compared BK and IBFS, but neither PRL nor HPF. Most related to our comparison is the (as yet) unpublished study of Fishbain *et al*. [16], but it does not compare IBFS either. Moreover, [16] suffers from the same problems as the study of Boykov and Kolmogorov [6] – it fails to test on modern vision problems with higher order factors and dense graphs with long-range connections.

3 Background: Max-Flow and Algorithms

In the interest of being self-contained, we now give a brief background of max-flow and the main families of the algorithms. Due to space limitations, we only provide a high-level overview of the algorithms and refer the reader the to respective publications for details.

Notation

For any positive integer n, let $[n]$ be shorthand for the set $\{1,2,\ldots,n\}$. Let $G=(\mathcal{V},\mathcal{E})$ be a directed graph over n nodes, *i.e.* $\mathcal{V}=[n]$, $\mathcal{E}\subseteq\{(i,j)\mid i,j\in\mathcal{V}, i\neq j\}$ Let s,t be two special nodes called source and sink respectively. Moreover, let $\mathcal{C}=\{c_{ij}\mid(i,j)\in\mathcal{E}, c_{ij}\geq 0\}$ be a set of positive integer *capacities* at edges.

A *feasible flow* in graph G is a positive number for each edge *i.e.* $f:\mathcal{E}\rightarrow\mathbb{R}_+$, such that it satisfies *capacity constraints* $f_{ij}\leq c_{ij}$, and *conservation constraints*: $\sum_{(i,j)\in\mathcal{E}} f_{ij}=\sum_{(j,k)\in\mathcal{E}} f_{jk}, \quad \forall j\in\mathcal{V}\setminus\{s,t\}$. The *value* of a flow is defined as the total outgoing flow from the source: $v=\sum_{(s,i)\in\mathcal{E}} f_{si}$. A *preflow* is a flow vector that satisfied the capacity constraints, but may have more net incoming flow than outgoing flow, *i.e.* $\sum_{(i,j)\in\mathcal{E}} f_{ij}-\sum_{(j,k)\in\mathcal{E}} f_{jk}\geq 0 \quad \forall j\in\mathcal{V}\setminus\{s,t\}$. The *excess* at a node j is defined as the total incoming flow minus the outgoing flow, *i.e.* $ex(j)=\sum_{(i,j)\in\mathcal{E}} f_{ij}-\sum_{(j,k)\in\mathcal{E}} f_{jk}$. We will refer to a positive excess as a *surplus* and a negative excess as a *deficit*. A *pseudoflow* is a flow vector that satisfies the capacity constraints, but may have both positive and negative excess, *i.e.* may violate the conservation constraints in any way. A node with strictly positive excess is called a *strong* node, and nodes with deficit (or zero excess) are called *weak* nodes.

Given a flow vector satisfying capacity constraints, the set of *residual arcs* is given by $\mathcal{R}=\{(i,j)\mid(i,j)\in\mathcal{E}, f_{ij}<c_{ij}\ OR\ (j,i)\in\mathcal{E}, f_{ji}>0\}$, *i.e.* forward arcs with flow strictly less than the capacity or backward arcs with capacity strictly greater than 0. The graph induced by these residual capacities, $G_{\mathcal{R}}=(\mathcal{V},\mathcal{R})$, is known as the *residual graph*. Moreover, the *residual capacity* of a residual arc (i,j) is given by $\tilde{c}_{ij}=c_{ij}-f_{ij}$ if $(i,j)\in\mathcal{E}$ and $\tilde{c}_{ij}=f_{ji}$ if $(j,i)\in\mathcal{E}$. An arc $(i,j)\in\mathcal{E}$ is said to be *saturated* if $\tilde{c}_{ij}=0$, and *free* if $0<f_{ij}<c_{ij}$.

Max-flow. Given an input graph G and edge-capacities $\mathcal{C}$, the max-flow problem asks for a feasible flow of maximum value.

Broadly speaking, there are three main families of max-flow algorithms: Augmenting-Path, Preflow-Push, and Pseudoflow algorithms. We now give a brief overview of the three families, and the particular algorithms we compare in our study.

Augmenting-Path (AP) Variants

AP algorithms are primal feasible algorithms that maintain a feasible flow throughout the execution of the algorithm. They iteratively search for *s-t* paths in the residual graph and push the minimum capacity of arcs on this path (bottleneck capacity) from source to sink. The algorithms terminate when no more *s-t* paths can be found.

Dinic's algorithm (DF) [13] used breadth-first search to find the shortest s-t and achieved the worst case running time of $O(n^2m)$. Boykov and Kolmogorov [5] presented an algorithm (BK) that built search trees from source and sink to find s-t. paths. Although the algorithm has only a pseudo-polynomial runtime bound of $O(n^2mC)$ and no known polynomial time bound, it is very efficient in practice and outperformed all other methods in their study. More recently, Goldberg *et al.* [21] have presented an improved version of BK that is based on incremental breadth-first search (IBFS). IBFS has a worst case guarantee of $O(n^2m)$ and seems to empirically perform better than BK.

Push-Relabel (PRL) Variants

Push-Relabel algorithms [8, 20] are dual feasible algorithms; they do not maintain a feasible flow, rather a preflow. Thus nodes may have a flow surplus. The algorithms maintain a labelling for every node $d(i)$, $\forall i \in \mathcal{V}$. If $d(i) < n$, then $d(i)$ is a lower bound estimate of the distance from node i to sink in the residual graph. If $d(i) \geq n$, sink is no longer reachable from i, and $n - d(i)$ denotes a lower-bound on the distance to the source. The labels for s,t stay fixed at n and 0 respectively. The algorithms are typically initialized by saturating all outgoing arcs from the source, *i.e.* by setting $f_{si} = c_{si}$, $\forall (s,i) \in \mathcal{E}$, and labelling all nodes as 0. An iteration of the algorithm consists of performing one of two basic operations: *push* and *relabel*. The push operation involves selecting a strong node and pushing flow from it to a neighbour with lower label. If no such neighbour exists, then the relabel operation is applicable – it increases the label of such a node by 1. Phase 1 of the algorithm terminates when all strong nodes have labels $d(i) \geq n$, *i.e.* when no more excess flow can be transferred to the sink. At this point, an s-t cut in the graph can be extracted. Specifically, the source set consists of all nodes with labels $d(i) \geq n$. Phase 2 of the algorithm transforms this maximum preflow into a maximum flow by sending surplus flow back to the source (via the same basic operations push and relabel).

As with AP algorithms, the efficiency of push-relabel algorithms depend on the order in which the basic operations are performed. Goldberg and Tarjan [20] showed that a generic version of the algorithm that does not order these operations in any smart way runs in $O(n^2m)$ time. Moreover, two faster variants were proposed: highest-label-first, and first-in-first-out (FIFO). The highest-label-first variant always choses a strong node with the highest label for processing and can be shown to run in $O(n^2\sqrt{m})$ time. The FIFO variant maintains a queue in which strong nodes enter from the back and popped from the front. This variant runs in $O(n^3)$ time.

Pseudoflow (HPF) Variants

Pseudoflow algorithms [7, 24] share a number of characteristics with push-relabel. They too are dual feasible algorithms and do not maintain a feasible flow, rather a pseudoflow. They too proceed in two phases: the first phase finds an s-t cut and the second phase extracts the maximum feasible flow from the current pseudoflow. Pseudoflow algorithms maintain a few invariants throughout Phase 1. First, all source-outgoing and sink-incoming arcs stay saturated. Second, the free arcs do not contain any cycles. This induces a forest with multiple components. Third, only nodes with strictly positive or negative excess are the roots of these components. If the root is strong, the component is called a strong component, and if the root is weak the component is called a weak component. In each iteration, the algorithm attempts to find a residual arc from a node in the strong component to a node in the weak component. If such an arc does not exists the algorithm finishes Phase 1. It can be shown

that in each iteration, either the total excess of strong nodes is strictly decreased or at least one weak node becomes a strong node.

Contrast.

At a high-level, augmenting path algorithms perform completely "global" operations, *i.e.* search the residual graph to find a complete *s-t* path and push flow along this entire path, while push-relabel algorithms perform completely "local" operations, *i.e.* only push flow along single edges towards nodes closer to sink. Pseudoflow algorithms lie somewhere in between by maintaining components of nodes. Moreover, the key difference between pseudoflow and push-relabel algorithms is that pseudoflow algorithms allow for flow to be pushed between two nodes at the same label, while push-relabel algorithms do not.

4 Experimental Setup

We now describe our experimental setup in detail.

Problems. We tested a number of max-flow algorithms on the following problems:

1. **Synthetic.** We constructed 10 synthetic 2-label segmentation graphs, where we started with a basic grid structure and randomly added long-range edges depending on a density parameter. The edge-capacities were sampled from Gaussians.
2. **ALE Graphs.** The Automatic Labeling Environment (ALE) of Ladicky *et al.* [29] implements the Associative Hierarchical CRF framework of Ladicky *et al.* [28], which achieved competitive results in recent years' PASCAL object category segmentation challenges. ALE includes many kinds of potentials: unary potentials based on textonboost features, P^n Potts terms (between superpixel nodes and pixel nodes) and a global co-occurrence potential [30]. Inference in ALE is performed with graph-cuts (BK) based alpha-expansion. We ran ALE on 10 images from PASCAL VOC 2010 segmentation dataset [15] and modified the code to save max-flow graphs in each iteration of alpha-expansion resulting in 337 graphs.
3. **Deconvolution.** Given a blurry and noisy (binary) input image the task in image deconvolution is to reconstruct the original sharp image. We use the "CVPR" instance from [36]. Given an $n \times n$ blur-kernel, the MRF connectivity is $(2n-1) \times (2n-1) - 1$. For a 3x3 blur-kernel, this model contains ∼45,000 triplets, and for a 5x5 blur-kernel ∼450,000 triplets. This is an extremely dense graph and the problem is not submodular. We saved the QPBO graph to file and ran all methods on this graph.
4. **Super Resolution.** The goal in image super resolution is to predict a high-res image given a low-res image. Freeman [18] showed how to formulate this as a labelling problem on patches, where the labels index into a dictionary of patches. We use the models of [36], and again saved the QPBO graphs to file.
5. **Texture Restoration.** In this task, the goal is to denoise a noisy texture image. We used the Brodatz texture D103 model from [36] and again saved the QPBO graphs.
6. **DTF Graphs.** Decision Tree Field (DTF) [34] is a recently introduced model that combines random forests and conditional random fields. DTF models tend to involve particularly difficult inference problem. We used the 100 instances provided by Nowozin *et al.* [34] and saved the QPBO-graphs to file.

7. **3D Segmentation.** Finally, we also evaluated all algorithms on the standard benchmark for such studies, the binary 3D (medical) segmentation instances from the University of Western Ontario http://vision.csd.uwo.ca/maxflow-data/.

We note that all of the previous studies have been restricted to 3D segmentation, and problems 2-6 have never been used to evaluate max-flow algorithm, yet these are in some sense more representative of modern problems.

Implementations. We compare the following implementations:

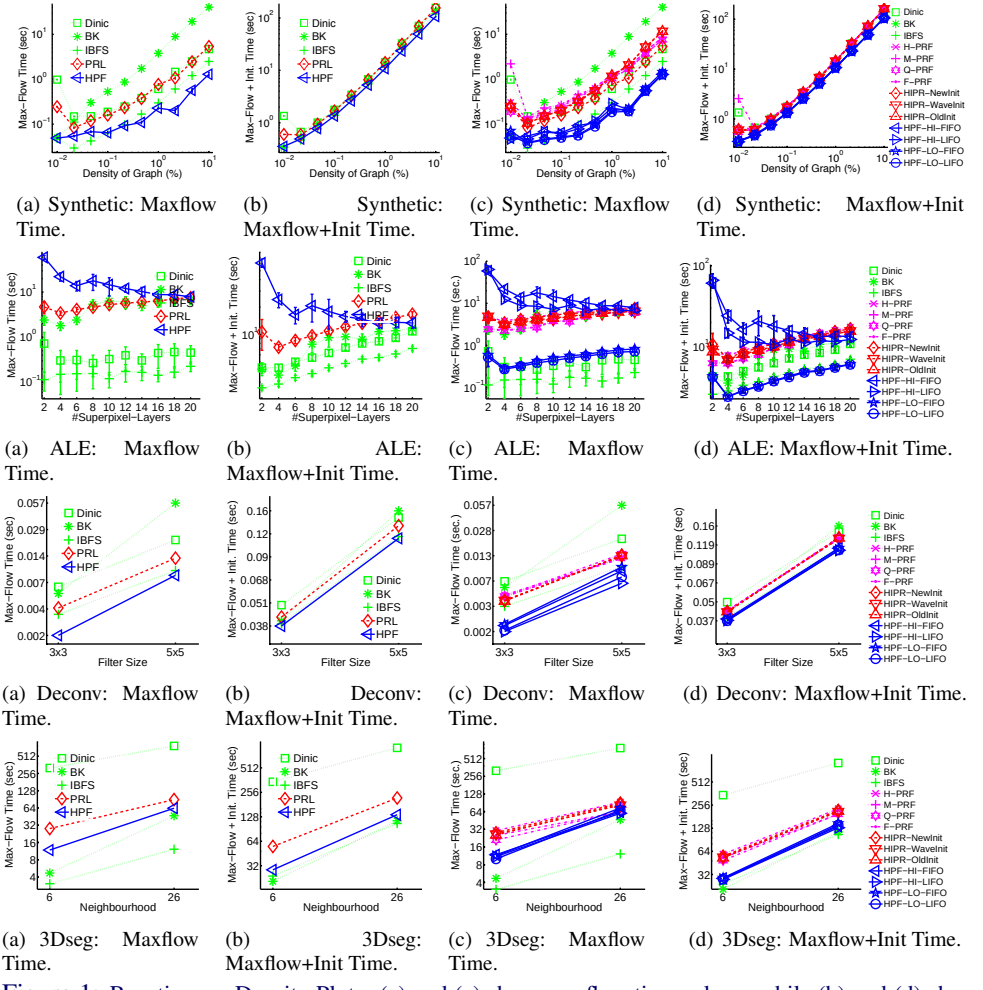
1. Augmenting-Path (AP) variants.
   - Dinic's algorithm (DF) [13]: We used the C implementation provided by Goldberg http://www.avglab.com/andrew/soft.html.
   - Boykov-Kolmogorov (BK): We used the C++ implementation provided by Kolmogorov http://pub.ist.ac.at/~vnk/software/maxflow-v3.02.src.tar.gz.
   - Incremental Breadth First Search (IBFS). We used the C++ implementation provided by Hed http://www.cs.tau.ac.il/~sagihed/ibfs/.
2. Push-Relabel (PRL). For sake of completeness, we used two different C implementations of PRL provided by Goldberg http://www.avglab.com/andrew/soft.html – called PRF and HIPR. PRF uses both highest-label first and FIFO push orderings, with and without certain relabeling heuristics (see [8] for details), and results in 4 variants H-PRF, M-PRF (highest-label first) and Q-PRF, F-PRF. Within the HIPR implementation, the choice of preflow initialization seems to matter, resulting in three different variants HIPR-New, HIPR-Old and HIPR-Wave (see [8] for details).
3. Pseudoflow. We used the C++ implementation provided by Chandran and Hochbaum http://riot.ieor.berkeley.edu/Applications/Pseudoflow/maxflow.html, and compared the following variants: highest-label-first (HI) and lowest-label-first (LO) processing of root nodes; and FIFO and LIFO ordering of roots among roots of same label. This resulted in 4 variants: HPF-HI-FIFO, HPF-HI-LIFO, HPF-LO-FIFO and HPF-LO-LIFO.

In total, we compared 14 (3AP+7PRL+4HPF) different algorithm-variant pairs.

Common setup All implementations were converted into C++ and we wrote a common interface. For all experiments, we saved the max-flow problems in a standard DIMACS file format [1], which was then fed as input to all methods. PRL and HPF implementations already contained DIMACS readers, but for BK and IBFS this was written by us.

In each experiment, we measured the following quantities: time taken to read input files (read-time); time taken to initialize internal graph representations and other data-structures (init-time); time taken to compute the max-flow (maxflow-time). The read-time was approximately the same for all methods and is thus not reported. Different methods use vastly different internal representations and the init-time varied between families of algorithms. Note that PRL and HPF methods first compute a cut and then compute the max-flow. For a number of vision problems, the cut is of principal interest and not the actual flow values. In such applications, one would expect further saving over the maxflow-time. However, these reductions for all our experiments were a very small fraction of the maxflow-time, and we do not report them here.

All experiments were performed on a 64-bit 4-core 1.6 GHz Intel(R) Core i7 GNU/Linux machine with 8GB RAM. All times were measured via the C++ `clock()` function that



(a) Synthetic: Maxflow Time. (b) Synthetic: Maxflow+Init Time. (c) Synthetic: Maxflow Time. (d) Synthetic: Maxflow+Init Time.

(a) ALE: Maxflow Time. (b) ALE: Maxflow+Init Time. (c) ALE: Maxflow Time. (d) ALE: Maxflow+Init Time.

(a) Deconv: Maxflow Time. (b) Deconv: Maxflow+Init Time. (c) Deconv: Maxflow Time. (d) Deconv: Maxflow+Init Time.

(a) 3Dseg: Maxflow Time. (b) 3Dseg: Maxflow+Init Time. (c) 3Dseg: Maxflow Time. (d) 3Dseg: Maxflow+Init Time.

Figure 1: Run-time vs Density Plots: (a) and (c) show maxflow times alone, while (b) and (d) show maxflow+initialization times. (a) and (b) show a subset of algorithms, while (c) and (d) show all 14.

measures process time with a typical resolution of 1 millisecond. Higher resolutions in measuring time are possible with other (non-portable) functions but none of our comparisons required such fine resolution.

5 Results

Fig. 1,2,3 show the results of all our comparisons. We try to cluster the results so that a meaningful picture might emerge.

Fig. 1 compares run-time as a function of some measure of "density" of the graphs. For synthetic graphs, we can directly control the number of edges present. However, for other applications we cannot directly change the density of the max-flow graphs; thus we resort to varying some secondary application-specific quantities that ultimately result in increased density of the max-flow graph. For ALE graphs, we varied the number of superpixels lay-

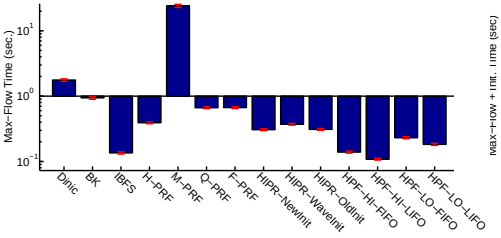
ers used by ALE, which increases the number of P^n-Potts terms, thus ultimately increasing the density. For Deconvolution, we vary the filter-size from 3x3 to 5x5, which significantly increases the density of the graph. For 3D segmentation, we vary the neighbourhood connectivity from 6 neighbourhood to 26 neighbourhood, which also increases the density. The synthetic results are averaged over 10 graphs and ALE results over 22 graphs (all alpha-expansion iterations of 1 image). In order to clearly see the curves, we show two sets of plots: one containing all 14 methods, and another containing just DF, BK, IBFS, HIPF-New and HPF-HI-FIFO, which are overall the most competitive variants.

For problems where we could not vary the density, Fig. 2 shows just the bar plots. In case of DTF, these are averaged over 100 instances. Fig. 3 shows the change in run-time with different expansions within a sweep of alpha-expansion.

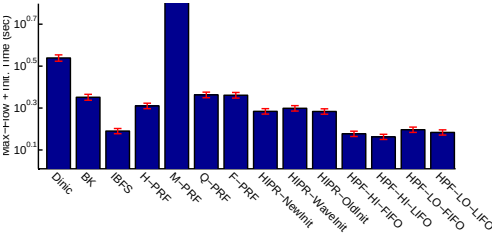
Finally, note that most timing axes are in log-scale and thus any differences are fairly significant. We now summarize our findings.

5.1 Take Home Messages

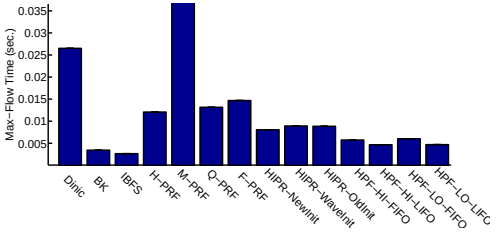
1. **Choice of Algorithm *Matters*.** In all applications, the fastest algorithms is *orders of magnitude* faster than the slowest algorithm.
2. **BK Scales Poorly with Density.** Fig. 1 shows that the motivating hypothesis of this study is correct. In a number of cases (Synthetic, Deconv, 3Dseg) BK starts out fairly competitive at low densities but very quickly becomes the slowest algorithm, sometimes even slower than DF.
3. **New Kids in Town: IBFS and HPF.** In a number of applications we considered, both IBFS and HPF significantly outperform BK both in terms of maxflow time and maxflow+init time, IBFS more consistently so than HPF. For instance, our experiments seem to suggest that the ALE implementation could be made 2x faster simply by switching to IBFS or HPF, at no loss in accuracy. While the number of instances in each application might be limited, the consistency of the results certainly provides evidence for this claim.
4. **Clever Data-structures Matter.** Different implementations use vastly different internal representations and thus initialization times vary. In a number of applications, the initialization times dominate maxflow times, especially as the graphs become denser and the data-structures become heavy. We found the data-structures used by BK to be particularly efficient. In a number of applications (see *e.g.* SuperRes, Texture-Restoration), BK maxflow time is longer than IBFS but the maxflow+init time is shorter. However, this may not be too relevant for applications in video where the data-structures may be initialized once and maxflow called repeatedly.
5. **Counter-Intuitive Findings.** We report a couple of counter-intuitive findings. Chandran and Hochbaum [7] found HPF-LO (LIFO and FIFO) to consistently outperform HPF-HI (LIFO and FIFO). In ALE experiments, we find the opposite to be the case, by a significant margin. We believe this has to do with the particular kind of hierarchical graph structure constructed in ALE, but this requires further examination. We also found DF to perform surprisingly well on ALE.



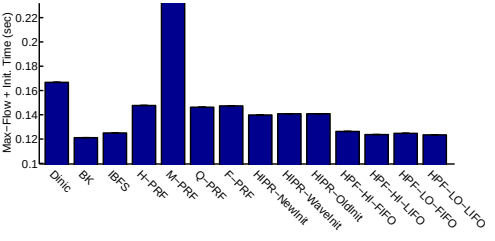
(a) DTF Graphs: Maxflow Time.



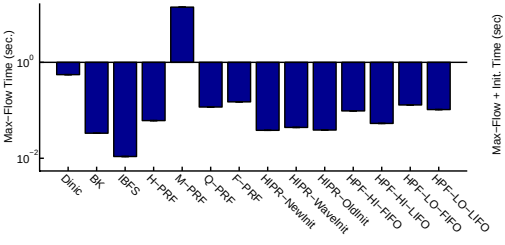
(b) DTF Graphs: Maxflow+Init Time.



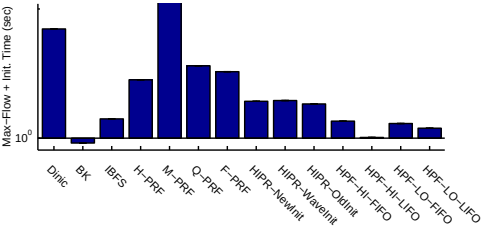
(a) SuperRes Graphs: Maxflow Time.



(b) SuperRes Graphs: Maxflow+Init Time.

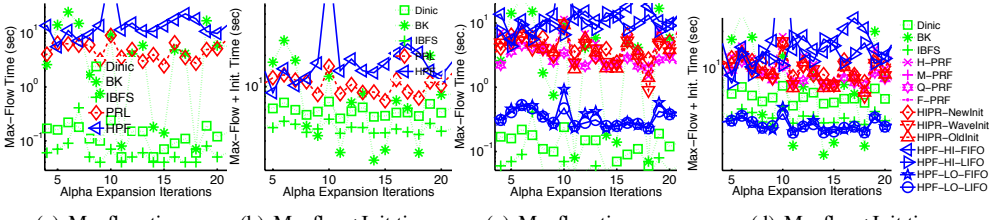


(a) Texture-Restoration Graphs: Maxflow Time.



(b) Texture-Restoration Graphs: Maxflow+Init Time.

Figure 2: Run-times on various applications: (a) shows maxflow time and (b) shows maxflow+initialization time.



(a) Maxflow time. (b) Maxflow+Init time. (c) Maxflow time. (d) Maxflow+Init time.

Figure 3: Run-time on ALE as a function of α-Iterations.

6 Conclusions and Future Work

In summary, we compared a number of max-flow algorithms on representative computer vision problems. Among other things, we found that the most popularly used implementation of Kolmogorov [5] is not in fact the fastest algorithm available, which was surprising at least to the authors. We hope our findings will be useful to the community at large. There are a number of ways in which this comparison could be extended – most notably to include multi-core and distributed implementation like [10, 22, 33, 34], which become necessary as soon as the graphs become too large to hold in memory. We plan to pursue this direction.

Acknowledgements. We thank Pushmeet Kohli for complaining about the lack of such a comparison; Lubor Ladicky for answering our questions about ALE; Andrew Delong for answering our questions about the 3D segmentation instances; and Sameh Khamis for answering our questions about Region Push-Relabel. A significant portion of this work was done while Tanmay Verma was an intern at TTI-Chicago.

References

[1] The DIMACS File Format for Max-Flow Problems. http://lpsolve.sourceforge.net/5.5/DIMACS_maxf.htm.

[2] Ravindra K. Ahuja, Murali Kodialam, Ajay K. Mishra, and James B. Orlin. Computational investigations of maximum flow algorithms. *European Journal of Operational Research*, 97(3): 509 – 542, 1997.

[3] R. J. Anderson and J. C. Setuba. Goldberg's algorithm for the maximum flow in perspective: a computational study. In *Network Flows and Matching: First DIMACS Implementation Challenge*, 1993.

[4] Dhruv Batra and Pushmeet Kohli. Making the right moves: Guiding alpha-expansion using local primal-dual gaps. In *CVPR*, 2011.

[5] Yuri Boykov and Vladimir Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *PAMI*, 26(9):1124–1137, 2004.

[6] Yuri Boykov, Olga Veksler, and Ramin Zabih. Efficient approximate energy minimization via graph cuts. *PAMI*, 20(12):1222–1239, 2001.

[7] Bala G. Chandran and Dorit S. Hochbaum. A computational study of the pseudoflow and push-relabel algorithms for the maximum flow problem. *Oper. Res.*, 57:358–376, March 2009. ISSN 0030-364X. doi: 10.1287/opre.1080.0572.

[8] Boris V. Cherkassky and Andrew V. Goldberg. On implementing the push-relabel method for the maximum flow problem. *Algorithmica*, 19(4):390–410, 1997.

[9] Paul Christiano, Jonathan A. Kelner, Aleksander Madry, Daniel A. Spielman, and Shang-Hua Teng. Electrical flows, laplacian systems, and faster approximation of maximum flow in undirected graphs. In *STOC*, pages 273–282, 2011.

[10] Andrew Delong and Yuri Boykov. A scalable graph-cut algorithm for n-d grids. In *CVPR*, 2008.

[11] Andrew Delong, Anton Osokin, Hossam N. Isack, and Yuri Boykov. Fast approximate energy minimization with label costs. In *CVPR*, pages 2173–2180, 2010.

[12] U. Derigs and W. Meier. Implementing goldberg's max-flow-algorithm âĂŤ a computational investigation. *Mathematical Methods of Operations Research*, 33:383–403, 1989. ISSN 1432-2994. URL http://dx.doi.org/10.1007/BF01415937. 10.1007/BF01415937.

[13] E. A. Dinic. Algorithm for Solution of a Problem of Maximum Flow in a Network with Power Estimation. *Soviet Math Doklady*, 11:1277–1280, 1970.

[14] Jack Edmonds and Richard M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM*, 19:248–264, April 1972. ISSN 0004-5411.

[15] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2010 (VOC2010) Results. http://www.pascal-network.org/challenges/VOC/voc2010/workshop/index.html.

[16] Barak Fishbain, Dorit S. Hochbaum, and Stefan Mueller. Competitive analysis of minimum-cut maximum flow algorithms in vision problems. *CoRR*, abs/1007.4531, 2010.

[17] L. R. Ford and D. R. Fulkerson. Maximal Flow through a Network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[18] William T. Freeman, Egon C. Pasztor, and Owen T. Carmichael. Learning low-level vision. *International Journal of Computer Vision*, 40(1):25–47, 2000.

[19] Andrew V. Goldberg and Satish Rao. Beyond the flow decomposition barrier. *J. ACM*, 45(5): 783–797, September 1998. ISSN 0004-5411. doi: 10.1145/290179.290181. URL http://doi.acm.org/10.1145/290179.290181.

[20] Andrew V. Goldberg and Robert E. Tarjan. A new approach to the maximum-flow problem. *J. ACM*, 35:921–940, October 1988. ISSN 0004-5411. doi: http://doi.acm.org/10.1145/48014.61051. URL http://doi.acm.org/10.1145/48014.61051.

[21] Andrew V. Goldberg, Sagi Hed, Haim Kaplan, Robert E. Tarjan, and Renato F. Werneck. Maximum flows by incremental breadth-first search. In *Proceedings of the 19th European conference on Algorithms*, ESA'11, pages 457–468, 2011. ISBN 978-3-642-23718-8.

[22] Felix Halim, Roland H. C. Yap, and Yongzheng Wu. A mapreduce-based maximum-flow algorithm for large small-world network graphs. In *Proceedings of the 2011 31st International Conference on Distributed Computing Systems*, ICDCS '11, pages 192–202, Washington, DC, USA, 2011. IEEE Computer Society. ISBN 978-0-7695-4364-2. doi: 10.1109/ICDCS.2011.62. URL http://dx.doi.org/10.1109/ICDCS.2011.62.

[23] P.L. Hammer. Some network flow problems solved with pseudo-boolean programming. *Operations Research*, 13:388–399, 1965.

[24] Dorit S. Hochbaum. The pseudoflow algorithm: A new algorithm for the Maximum-Flow problem. *Operations Research*, 56(4):992–1009, July 2008.

[25] M. Hussein, A. Varshney, and L. Davis. On implementing graph cuts on CUDA. In *First Workshop on General Purpose Processing on Graphics Processing Units*, 2007.

[26] Pushmeet Kohli, L'ubor Ladicky, and Philip H. S. Torr. Robust higher order potentials for enforcing label consistency. In *CVPR*, 2008. ISBN 978-1-4244-2242-5. doi: http://doi.ieeecomputersociety.org/10.1109/CVPR.2008.4587417.

[27] Vladimir Kolmogorov and Ramin Zabih. What energy functions can be minimized via graph cuts? *PAMI*, 26(2):147–159, 2004.

[28] L. Ladickỳ, C. Russell, P. Kohli, and P. H. S. Torr. Associative hierarchical CRFs for object class image segmentation. *ICCV*, 2009.

[29] Lubor Ladicky and Philip H.S. Torr. The automatic labelling environment. http://cms.brookes.ac.uk/staff/PhilipTorr/ale.htm.

[30] Lubor Ladicky, Chris Russell, Pushmeet Kohli, and Philip H. S. Torr. Graph cut based inference with co-occurrence statistics. In *ECCV*, pages 239–253, 2010. ISBN 3-642-15554-5, 978-3-642-15554-3. URL http://dl.acm.org/citation.cfm?id=1888150.1888170.

[31] Yunpeng Li and Daniel P. Huttenlocher. Sparse long-range random field and its application to image denoising. In *ECCV*, pages 344–357, 2008. ISBN 978-3-540-88689-1. doi: http://dx.doi.org/10.1007/978-3-540-88690-7_26. URL http://dx.doi.org/10.1007/978-3-540-88690-7_26.

[32] Yongsub Lim, Kyomin Jung, and Pushmeet Kohli. Energy minimization under constraints on label counts. In *ECCV*, pages 535–551, 2010. ISBN 3-642-15551-0, 978-3-642-15551-2. URL http://dl.acm.org/citation.cfm?id=1888028.1888070.

[33] Jiangyu Liu and Jian Sun. Parallel graph-cuts by adaptive bottom-up merging. In *CVPR*, pages 2181–2188, 2010.

[34] Sebastian Nowozin, Carsten Rother, Shai Bagon, Toby Sharp, Bangpeng Yao, and Pushmeet Kholi. Decision tree fields. In *ICCV*, 2011.

[35] Srikumar Ramalingam, Pushmeet Kohli, Karteek Alahari, and Philip H. S. Torr. Exact inference in multi-label CRFs with higher order cliques. *CVPR*, 2008.

[36] C. Rother, V. Kolmogorov, V. Lempitsky, and M. Szummer. Optimizing binary MRFs via extended roof duality. In *CVPR*, June 2007. doi: 10.1109/CVPR.2007.383203.

[37] C. Rother, P. Kohli, W. Feng, and J.Y. Jia. Minimizing sparse higher order energy functions of discrete variables. In *CVPR*, pages 1382–1389, 2009.

[38] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. "Grabcut": interactive foreground extraction using iterated graph cuts. *SIGGRAPH*, 2004.

[39] Michael Rubinstein, Ariel Shamir, and Shai Avidan. Improved seam carving for video retargeting. *ACM Transactions on Graphics (SIGGRAPH)*, 27(3):1–9, 2008.

[40] Vibhav Vineet and P J Narayanan. Cuda cuts: Fast graph cuts on the gpu. In *CVPR Workshop on Computer Vision on GPUs*, 2008. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4563095.