Parallel and Distributed Graph Cuts

Strandmark, Petter; Kahl, Fredrik

2010

Parallel and Distributed Graph Cuts

Petter Strandmark, Fredrik Kahl
Centre for Mathematical Sciences, Lund University
Email: {petter,fredrik}@maths.lth.se

***Abstract*—Graph cuts methods are at the core of many state-of-the-art algorithms in computer vision due to their efficiency in computing globally optimal solutions. In this paper, we solve the maximum flow/minimum cut problem in parallel by splitting the graph into multiple parts and hence, further increase the computational efficacy of graph cuts. Optimality of the solution is guaranteed by dual decomposition, or more specifically, the solutions to the subproblems are constrained to be equal on the overlap with dual variables.**

We demonstrate that our approach both allows (i) faster processing on multi-core computers and (ii) the capability to handle larger problems by splitting the graph across multiple computers on a distributed network. Even though our approach does not give a theoretical guarantee of speed-up, an extensive empirical evaluation on several applications with many different data sets consistently shows good performance.

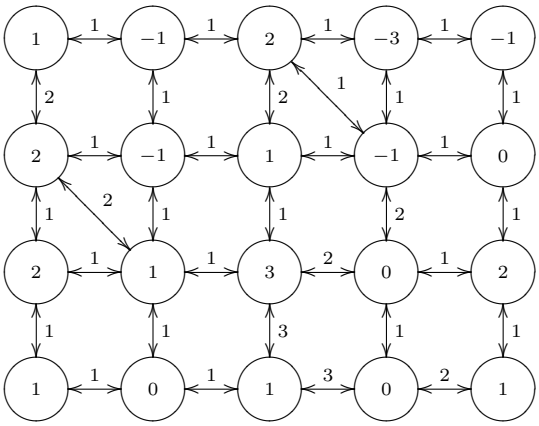
I. Introduction

Maximum flow algorithms are the foundations for many algorithms in computer vision. Examples include segmentation, image restoration, dense stereo estimation and shape matching, see [1], [2], [3], [4]. The approach is also useful for inferring the maximum *a posteriori* solution of a discrete MRF [2], [5].

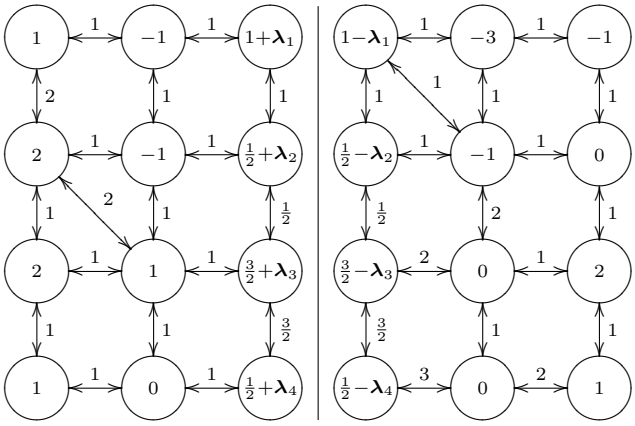
Our work builds on the following two trends: the ubiquity of maximum flow computations in computer vision and the tendency of modern microprocessor manufacturers to increase the number of cores in mass-market processors. This implies that an efficient way of parallelizing maximum flow algorithms would be of great use to the community. Due to a result from Goldschlager et al. [6], there is little hope in finding a general algorithm for parallel maximum flow with guaranteed performance gains. However, the graphs encountered in computer vision problems are often sparse with much fewer edges than the maximum $n^2 - n$ in a graph with n vertices. The susceptibility to parallelization depends on the structure and costs of the graph.

II. Graph cuts as a linear program

Finding the maximum flow, or, by duality, the minimum cut in a graph can be formulated as a linear program. Let $G = (V, \boldsymbol{c})$ be a graph where $V = \{s, t\} \cup \{1, 2, \dots, n\}$ are the source, sink and vertices, respectively, and $\boldsymbol{c}$ the edge costs. A cut is a partition S, T of V such that $s \in S$ and $t \in T$. The minimum cut problem is finding the partition where the sum of all costs of edges between the two sets is minimal. It can be formulated as [7]:



(a) Original graph. Numbers indicate s/t connection and edge costs.



(b) Subproblems with vertices in M and N, respectively.

Fig. 1: The graph decomposition into sets M and N. The pairwise energies in $M \cap N$ are part of both E_M and E_N and has to be weighted by $\frac{1}{2}$. Four dual variables $\boldsymbol{\lambda}_1 \dots \boldsymbol{\lambda}_4$ are introduced.

$$\begin{aligned} \underset{\boldsymbol{x}}{\text{minimize}} \quad & \sum_{i,j \in V} \boldsymbol{c}_{i,j} \boldsymbol{x}_{i,j} \\ \text{subject to} \quad & \boldsymbol{x}_{i,j} + \boldsymbol{x}_i - \boldsymbol{x}_j \geq 0, \quad i,j \in V \\ & \boldsymbol{x}_s = 0, \qquad \boldsymbol{x}_t = 1 \\ & \boldsymbol{x} \geq 0. \end{aligned} \tag{1}$$

The variable $\boldsymbol{x}_i$ indicates whether vertex i is part of S or T ($\boldsymbol{x}_i = 0$ or 1, respectively) and $\boldsymbol{x}_{i,j}$ indicates whether the edge (i, j) is cut or not. The variables are not constrained to be 0 or 1, but there always exists one such solution, according to the duality between maximum flow and minimum cut, cf. [8, p. 119]. We write $\mathcal{D}_V$ for the convex set defined by the constraints in (1).

A. Splitting the graph

Now pick two sets M and N such that $M \cup N = V$ and $\{s,t\} \subset M \cap N$. We assume that when $i \in M \setminus N$ and $j \in N \setminus M$, $\boldsymbol{c}_{i,j} = \boldsymbol{c}_{j,i} = 0$. That is, every edge is either within M or N, or within both. See Fig. 1.

We now observe that the objective function in (1) can be rewritten as:

$$\sum_{i,j \in V} \boldsymbol{c}_{i,j}\boldsymbol{x}_{i,j} = \sum_{i,j \in M} \boldsymbol{c}_{i,j}\boldsymbol{x}_{i,j} + \sum_{i,j \in N} \boldsymbol{c}_{i,j}\boldsymbol{x}_{i,j} - \sum_{i,j \in M \cap N} \boldsymbol{c}_{i,j}\boldsymbol{x}_{i,j}. \quad (2)$$

We define two energy functions on M and N:

$$\begin{aligned} E_M(\boldsymbol{x}) &= \sum_{i,j \in M} \boldsymbol{c}_{i,j}\boldsymbol{x}_{i,j} - \frac{1}{2} \sum_{i,j \in M \cap N} \boldsymbol{c}_{i,j}\boldsymbol{x}_{i,j} \\ E_N(\boldsymbol{y}) &= \sum_{i,j \in N} \boldsymbol{c}_{i,j}\boldsymbol{y}_{i,j} - \frac{1}{2} \sum_{i,j \in M \cap N} \boldsymbol{c}_{i,j}\boldsymbol{y}_{i,j}. \end{aligned} \quad (3)$$

This leads to the following equivalent linear program:

$$\begin{aligned} \underset{\substack{\boldsymbol{x} \in \mathcal{D}_M \\ \boldsymbol{y} \in \mathcal{D}_N}}{\text{minimize}} \quad & E_M(\boldsymbol{x}) + E_N(\boldsymbol{y}) \\ \text{subject to} \quad & \boldsymbol{x}_i = \boldsymbol{y}_i, \quad i \in M \cap N. \end{aligned} \quad (4)$$

Here $\boldsymbol{x}$ is the variable belonging to the set M (left in Fig. 1b) and $\boldsymbol{y}$ belongs to N. The two variables $\boldsymbol{x}$ and $\boldsymbol{y}$ are constrained to be equal in the overlap. The Lagrange dual function of this optimization problem is:

$$\begin{aligned} g(\boldsymbol{\lambda}) = & \min_{\substack{\boldsymbol{x} \in \mathcal{D}_M \\ \boldsymbol{y} \in \mathcal{D}_N}} \left(E_M(\boldsymbol{x}) + E_N(\boldsymbol{y}) + \sum_{i \in M \cap N} \boldsymbol{\lambda}_i(\boldsymbol{x}_i - \boldsymbol{y}_i) \right) = \\ & \min_{\boldsymbol{x} \in \mathcal{D}_M} \left(E_M(\boldsymbol{x}) + \sum_{i \in M \cap N} \boldsymbol{\lambda}_i \boldsymbol{x}_i \right) + \\ & \min_{\boldsymbol{y} \in \mathcal{D}_N} \left(E_N(\boldsymbol{y}) - \sum_{i \in M \cap N} \boldsymbol{\lambda}_i \boldsymbol{y}_i \right). \end{aligned} \quad (5)$$

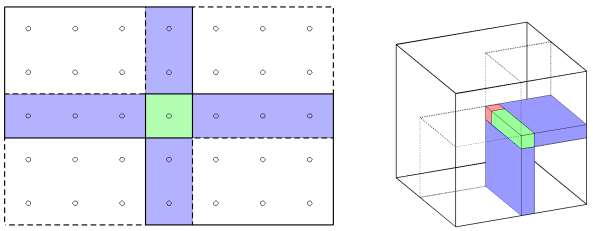
We now see that evaluating the dual function g amounts to solving two independent minimum cut problems of the type (1). Let $\boldsymbol{x}^*, \boldsymbol{y}^*$ be the solution to (4) and let $\boldsymbol{\lambda}^*$ maximize the dual function g. Because strong duality holds, we have $g(\boldsymbol{\lambda}^*) = E_M(\boldsymbol{x}^*) + E_N(\boldsymbol{y}^*)$. The solution to the subproblems may in general have multiple solutions, so to obtain a unique solution we always connect a vertex to the sink, if possible.

B. Algorithm

For an optimal choice of dual variables, the constraints in (4) will be satisfied. Solving the original problem (4) then amounts to finding the maximum value of the dual function. Since the dual function is concave, it can be maximized with an ascent method. However, it is not differentiable, but we use the supergradient[1] of g to find an ascent direction. The supergradient is given by $\boldsymbol{x}_i - \boldsymbol{y}_i$ for each i on the overlap $M \cap N$. The following maximization method[2] was found to converge quickly:

Start with $\boldsymbol{\lambda}_i = \mathbf{0}$ and a step length τ.
repeat
  while $g(\boldsymbol{\lambda} + \tau(\boldsymbol{x}_i - \boldsymbol{y}_i)) > g(\boldsymbol{\lambda})$ **do**
    $\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda} + \tau(\boldsymbol{x}_i - \boldsymbol{y}_i)$
    Update $\boldsymbol{x}$ and $\boldsymbol{y}$
  end
  $\tau \leftarrow \tau/2$
until $\boldsymbol{x}_i = \boldsymbol{y}_i$, $i \in M \cap N$

This iterative scheme is very efficient, since the search trees of the already solved graphs can be reused. Only a small number of costs are changed between iterations and our experiments show that these subsequent max-flow computations can be completed within microseconds, see Table I.



(a) 2×2 (b) $2 \times 2 \times 2$

Fig. 2: Splitting a graph into several components. The blue, green and red parts are weighted by $1/2$, $1/4$ and $1/8$, respectively.

C. More than two subproblems

Splitting a graph into more than two components can be achieved with the same approach. The energy functions analogous to (3) might then contain terms weighted by $1/4$ and $1/8$, depending on the geometry of the split. See Fig. 2.

III. Experiments on a single machine

In this section we describe experiments performed in parallel on a single machine executed across multiple threads. All times include any overhead associated with starting and stopping threads, allocation of extra memory etc. We have only considered the time for actual maximum flow calculations, i.e. the time required to construct the graphs is not taken into account. We note, however, that graph construction trivially benefit from parallel processing.

A. Image segmentation

We applied our parallel method for the 301 images in the Berkeley segmentation database [9], see Fig. 5 for examples. The segmentation model used was a piecewise constant model with the boundary length as a regulating term. The boundary length can be approximated with a neighborhood of edges around each pixel, usually of sizes 4, 8 or 16 in the two-dimensional case.

[1] completely analogous to subgradients for convex functions

[2] For our experiments in this paper we used an individual step length for each i, which led to slightly better performance.

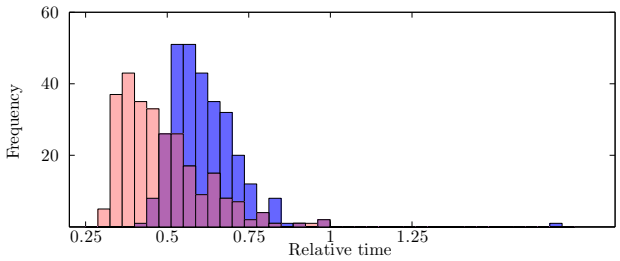


Fig. 3: Relative times with 2 (blue) and 4 (red) computational threads for the 301 images in the Berkeley segmentation database, using 4-connectivity. The medians are 0.596 and 0.455. See sections III-A and III-D.

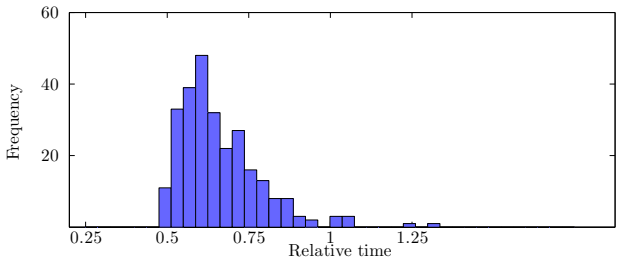


Fig. 4: Relative times using 8-connectivity and 2 computational threads. The median is 0.628.

The relative times ($t_{\text{multi-thread}}/t_{\text{single}}$) using two computational threads are shown in Figs. 3 and 4. Since the images in the database are quite small, the total processing time for a single image is around 10 milliseconds. Even with the overhead of creating threads and iterating to find the global minimum, we were able to get a significant speed improvement for almost all images. The exceptions are discussed in Section III-D.

Table I shows how the processing time varies with each iteration. In the last steps, very few vertices change and solving the maximum flow problems can therefore be done very quickly within microseconds.

It is very important to note that the problem complexity depends heavily on the amount of regularization used. That is, a segmentation problem in which boundary length is given a low cost is easy to solve and to parallelize. In the extreme case where no regularization is used, the problem reduces to simple thresholding, which of course is trivial to parallelize. Therefore, it is relevant to investigate how the algorithm performs with different amounts of regularization. We have done this and can conclude that our graph decomposition scheme performs well for a wide range of different settings, see Fig. 7. We see that the relative improvement in speed remains roughly constant over a large interval of different regularizations, whereas the absolute processing times vary by an order of magnitude.

When the number of computational threads increase, the computation times decrease as shown in Fig. 3.

B. Stereo problems

The "Tsukuba" dataset (which we obtained from [10]) consists of a sequence of max-flow instances corresponding to the first iteration of α-expansions [2]. First, we solved the 16 problems without any parallelization and then, with two computational threads. The relative times ranged from 0.51 to 0.72, with the average being 0.61.

Fig. 5: Examples from the Berkeley database [9].

Fig. 6: "Worst-case" test. The left and right side of the image is connected to the source and sink, respectively. The edge costs are determined by the image gradient. All flow must be communicated between the two computational threads when splitting the graph vertically. Even with this poor choice of split, the dual approach finished 30% faster.

C. Three-dimensional graphs

We used the graph construction described in [4] with data downloaded from [10] to evaluate the algorithm in three dimensions. For the "bunny" dataset the relative time was 0.67 with two computational threads.

D. Analysis of limitations

To see how our algorithm performs when the choice of split is very poor, we took a familiar image and split it in half from top to bottom as depicted in Fig. 6. We then attached the leftmost pixel column to the source and the rightmost to the sink. Splitting horizontally would have been much more preferable, since splitting vertically severs every possible s-t path and all flow has to be communicated between the threads. Still, the parallel approach finished processing the graph 30% *faster* than the single-threaded approach. This is a good indication that the choice of the split is not crucial for a speed improvement.

Figs. 3 and 4 contain a few examples ($< 1\%$) where the multi-threaded algorithm actually performs slower or almost the same as the single-threaded algorithm. The single example in Fig. 3 is interesting, because solving one of the subgraphs *once* takes significantly *longer* than solving the entire graph. This can happen for the BK algorithm, but is very uncommon in practice. We have noted that slightly perturbing any of the problem parameters (regularization, image model etc.) makes the multi-threaded algorithm faster also for this example.

The other slow examples have a simpler explanation: there is simply nothing interesting going on in one of the two halves of the graph, see e.g. the first image in Fig. 5. Therefore, the overhead of creating and deallocating the threads and extra memory gives the multi-threaded algorithm a slight disadvantage.

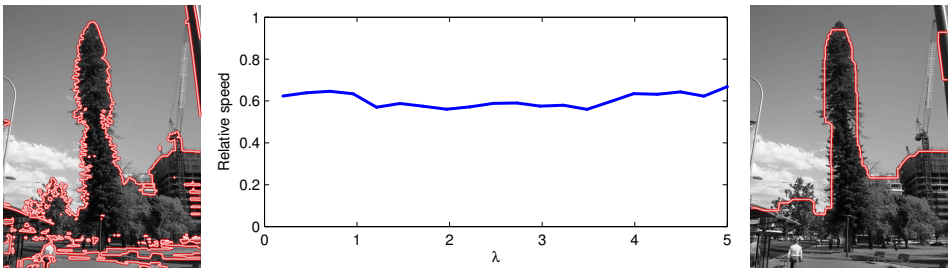


Fig. 7: Relative improvement in speed with two computational threads when the regularization parameter changes. Although the processing time ranged from 230 ms to 4 seconds, but the relative improvement was not affected.

Iteration	1	2	3	4	5	6	7	8	9	10	11
Differences	108	105	30	33	16	16	16	16	9	9	0
Time (ms)	245	1.5	1.2	0.1	0.08	0.09	0.07	0.15	0.06	0.07	0.47

TABLE I: Detailed information about the processing time for each iteration for a 1152×1536 example image (shown in Fig. 7). The number of overlapping pixels ($M \cap N$) was 1536 (one column). Deallocating memory and terminating threads is the cause of the processing time increase in the last iteration. The advantage of reusing search trees is clearly seen in the short processing times after the first iteration.

IV. Splitting across different machines

We now turn to another application of graph decomposition. Instead of assigning each part of the graph to a computational thread, one may assign each subgraph to a different *machine* and let the machines communicate the flow over a network.

Memory is often a limiting factor for maximum flow calculations. Using splitting we were able to segment 4-dimensional (space+time) MRI heart data with $95 \times 98 \times 30 \times 19 = 5.3\text{M}$ voxels. The connectivity used was 80, requiring 12.3 GB memory for the graph representation. By dividing this graph among 4 (2-by-2) different machines and using MPI for communication, we were able to solve this graph in 1980 seconds. Since only a small amount of data (54 kB in this case) needs to be transmitted between machines each iteration, this is an efficient way of processing large graphs. On the system we used (LUNARC Iris), the communication time was about 7-10 ms per iteration, for a total of 68 iterations until convergence.

The largest data set we used was a $256 \times 256 \times 1159 = 76\text{M}$ voxel CT scan with 6-connectivity. Solving this dataset required 16.6 GB of memory divided among 8 machines. We are not aware of any previous methods designed to solve problems of this magnitude.

We also evaluated the algorithms for some of the big problems available at [10]. The largest version of the "bunny" dataset is $401 \times 396 \times 312 = 50\text{M}$ with 300M edges was solved in 7 seconds across 4 machines. As a reference, a slightly larger version of the same dataset (not publicly available) was solved in over a minute with an (iterative) touch-and-expand approach in [4].

Splitting graphs across multiple machines also saves computation time, even though the MPI introduces some overhead. A single machine solved the small version of the "bunny" dataset [10] in 268 milliseconds, while two machines used 152 ms. Four machines (2-by-2) required 105 ms. The elapsed times were 2.3, 1.34 and 0.84 seconds, for the medium sized version respectively.

V. Conclusions

We have shown that it is possible to split a graph and obtain the global maximum flow by iteratively solving subproblems in parallel. Two applications of this technique were demonstrated:

- Faster maximum flow computations when multiple CPU cores are available (Section III).
- The ability to handle graphs which are too big to fit in the computer's RAM, by splitting the graph across multiple machines (Section IV).

Good results were demonstrated even if the split severs many, or even all s-t paths of the graph (Fig. 6). Experiments with different amounts of regularization suggest that the speed-up is relatively insensitive to regularization (Fig. 7).

References

[1] Y. Boykov and V. Kolmogorov, "An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 26, no. 9, pp. 1124–1137, Sept. 2004. 1

[2] Y. Boykov, O. Veksler, and R. Zabih, "Markov random fields with efficient approximations," in *Conf. Computer Vision and Pattern Recognition*, 1998. 1, 3

[3] C. Rother, V. Kolmogorov, and A. Blake, "Grabcut -interactive foreground extraction using iterated graph cuts," *ACM Transactions on Graphics (SIGGRAPH)*, 2004. 1

[4] V. Lempitsky and Y. Boykov, "Global optimization for shape fitting," in *Conf. Computer Vision and Pattern Recognition*, Minneapolis, USA, June 2007. 1, 3, 4

[5] D. M. Greig, B. T. Porteous, and A. H. Seheult, "Exact maximum a posteriori estimation for binary images," *Journal of the Royal Statistical Society*, 1989. 1

[6] L. M. Goldschlager, R. A. Shaw, and J. Staples, "The maximum flow problem is log space complete for P," vol. 21, no. 1, pp. 105–111, Oct. 1982. 1

[7] Wikipedia, "Max-flow min-cut theorem — Wikipedia, the Free Encyclopedia," [Online; accessed 9-January-2010]. 1

[8] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization; Algorithms and Complexity*. Dover Publications, 1998. 1

[9] D. Martin, C. Fowlkes, D. Tal, and J. Malik, "A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics," in *Int. Conf. Computer Vision*, 2001. 2, 3

[10] U. of Western Ontario, "Max-flow problem instances in vision," http://vision.csd.uwo.ca/maxflow-data. 3, 4