# A GENERAL GRAPH DECOMPOSITION AND MINIMUM CUTS COMPOSITION FRAMEWORK FOR MIN-CUT PROBLEM*

WENBING TAO† AND XUE-CHENG TAI‡

**Abstract.** A general graph decomposition and minimum cuts composition framework is proposed in this paper. We first present a graph decomposition method. We prove that the minimum cuts of the original graph can be obtained by finding the minimum cuts of a sequence of subgraphs. Based on the graph decomposition method and minimum cuts composition theorem, we develop a general framework, recursive minimum cuts (RMC) framework, where any existing minimum cuts algorithm can be used to solve minimum cuts problem over the sub-graphs. The computational complexity of the proposed approach is $O(N^{\frac{\log 3}{\log 2}})$ . We also design a parallel algorithm and run in approximate $O(N)$ time. Some preliminary experiments are provided on some synthetic graphs from images to test the proposed RMC framework.

**Key words.** Minimum cuts problem, graph decomposition, composition, recursive framework

**AMS subject classifications.** 15A15, 15A09, 15A23

## 1. Introduction.

Graph cuts have emerged as an increasingly powerful tool for exact or approximate energy minimization for various discrete pixel labeling problems in computer vision such as image segmentation[21, 26, 24], video segmentation[23], image restoration[16], stereo[28], shape reconstruction[1], object recognition[3], texture synthesis[22], and others. One of the primary reasons behind their growing popularity is the availability of efficient algorithm with low computational complexity to solve the min-cut/max-flow problem in graphs [5], which in turn allows for the computation of globally optimal solutions for certain important energy functions in vision. In [16] Greig et al. applied graph cuts to optimize this kind of energy function and gived a globally optimal binary labeling in computer vision. Previously, exact minimization of energy functions in [16] was impossible and such energies were approached by mainly using iterative algorithms like simulated annealing, whose results usually were far from the global minimum. Boykov et al. [6] then presented two algorithms based on graph cuts that efficiently find a local minimum with respect to two types of large moves, namely expansion moves and swap moves for the optimization problem of multi-labeling. The graphs corresponding to these applications are usually huge 2D or 3D grids and the efficiency of the min-cut/max-flow algorithms is an issue that need to be considered. Recently, graph-cut algorithms have been extended to solve some minimization problems from PDE-based image processing and these problems have been traditionally solved through Euler-Lagrangian minimization approaches. In [9, 10, 11], graph-cut method was used to get fast solutions for some TV (total variation) minimization problems. In some new attempts, continuous max-flow approaches [30, 29, 2] have been proposed. These approaches is extending the max-flow problem from discrete settings to continuous settings and some primal-dual algorithms can be

*This work was supported supported by the National Natural Science Foundation of China (Grant 61073093), the Fundamental Research Funds for the Central Universities of China (HUST: 2010MS025).

† Institute for Pattern Recognition and Artificial Intelligence and State Key Laboratory for Multi-spectral Information Processing Technologies, Huazhong University of Science and Technology, Wuhan 430074, China (wenbingtao@hust.edu.cn).

‡Department of Mathematics, University of Bergen, 5007 Bergen, Norway, and Division of Mathematical Sciences, School of Physical and Mathematical Sciences, Nanyang Technological University 637616, Singapore(tai@cma.uio.no).

used to get fast numerical schemes.

The goal of this paper is to develop a recursive graph cuts method which can solve the min-cut/max-flow problem more efficiently than the existing methods. The proposed method is essentially a generic framework into which any existing min-cut/max-flow algorithm can be integrated to produce a recursive method with lower time complexity than the original one. Especially, the solution by the recursive framework is exactly the same with the original minimum cuts algorithm, not an approximation. If the time complexity of the integrated min-cut algorithms can be denoted as $O(N^p)$, we can prove the total time complexity of the proposed RMC algorithm is $O(N^{\frac{\log 3}{\log 2}})$. This is implemented based on the proposed graph decomposition method and minimum cuts composition theorem.The primary conclusions of the proposed method are as follows.

1) Given an arbitrary cut $C_0$ of $G$, we construct subgraphs $G_1$ and $G_2$ as in Sec. 2. Presume that the min-cuts of $G_1$, $G_2$ and $G$ are $c_1(S_1, T_1)$, $c_2(S_2, T_2)$ and $c(S, T)$ respectively. We prove that

$$S_1 \subseteq S \text{ and } T_2 \subseteq T.$$

This justifies that the min-cut $C_{\min}$ of $G$ lies in the gray zone in Fig.1 (row 1) as was pointed out in [17].

2) We further construct subgraph $G_3$ according to the min-cuts $c_1(S_1, T_1)$ and $c_2(S_2, T_2)$ of subgraphs $G_1$ and $G_2$. Details will be given in Sec. 2. Presume that the min-cut of $G_3$ is $c_3(S_3, T_3)$. We prove that

$$S_3 \subseteq S \text{ and } T_3 \subseteq T.$$

This means that the min-cut of $G_3$ is a subset of min-cut of $G$. This argument directly leads to the conclusion that

$$S = S_1 + S_3 - s \text{ and } T = T_2 + T_3 - t.$$

Therefore, we can obtain $c(S, T)$ by computing $c_1(S_1, T_1)$, $c_2(S_2, T_2)$ and $c_3(S_3, T_3)$.

3) Our proof and the recursive cut method are independent of the minimum cut algorithms. We don't need to convert between the excess or the deficit and t-links as in [17] which is just suitable to Push- Relabel algorithms. We can apply any min-cut/max-flow algorithm, including "augmenting paths" style algorithms [13] and "push-relabel" style algorithms [15] to our general graph decomposition and minimum cuts composition framework.

4) Since our proposed method is independent of the min-cut/max-flow algorithm, we can further apply this method to the sub-graphs $G_1$, $G_2$ and $G_3$ of $G$. Therefore, a recursive frame can be designed to solve the min-cut/max-flow problems. Theoretically, the recursive process can be continued until the constructed sub-graphs include only one node in addition to the source and sink nodes. We develop a new recursive minimum cut (RMC) framework to solve max-flow/min-cut problem. We show that the mean time complexity is $O(N^{\frac{\log 3}{\log 2}})$.

5) Based on the graph decomposition method and minimum cuts composition theorem, a parallel algorithm can be developed to solve the min-cut problem with time complexity of order $O(N)$ using $N$ processors.

The rest of the paper is organized as follows. Section 2 outlines the basic min-cut/max-flow problems and algorithm and the motivation and overview of the proposed method in this paper. Section 3 provides the graph decomposition method to

decompose the original graph $G$ into three subgraphs $G_1$, $G_2$ and $G_3$ of $G$ given an arbitrary initial cut $c_0(S_0, T_0)$ of $G$. In Section 4 we will provide the minimum cuts composition theorems and their proofs based on graph theory. The recursive minimum cuts framework based on the proposed graph decomposition and minimum cuts composition method is developed in Section 5. In Section 6 the time complexity of the recursive minimum cuts framework is analyzed. The parallel algorithm design and analysis is given in Section 7. Some experiments are shown in Section 8, followed by a brief conclusion section in section 9.

## 2. Related Works.

**2.1. Min-Cut and Max-flow Problems.** A flow network $G = (V, E)$ is a directed graph in which each edge $(u, v) \in E$ has a nonnegative capacity $c(u, v) \geq 0$. Two vertices are distinguished in a flow network: a source $s$ and a sink $t$. We assume that every vertex lies on some path from the source to the sink. That is, for every vertex $v \in E$, there is a path $s \sim v \sim t$. We now define flows more formally. Let $G = (V, E)$ be a flow network with a capacity function $c$. Let $s$ be the source of the network, and let $t$ be the sink. A flow in $G$ is a real-valued function $f : V \times V \to R$ that satisfies the following three properties:

Capacity constraint: For all $u, v \in V$, $f(u, v) \leq c(u, v)$

Skew symmetry: For all $u, v \in V$, $f(u, v) = -f(v, u)$

Flow conservation: For all $u \in V - \{s, t\}$, $\sum_{u \in V} f(u, v) = 0$

The quantity $f(u, v)$, which can be positive, zero, or negative, is called the flow from vertex $u$ to vertex $v$. In the maximum flow problem, given a flow network $G$ with source $s$ and sink $t$ we wish to find a flow of maximum value.

For simplicity, we use an implicit summation notation in which every argument, or both, may be a set of vertices. For example, if $X$ and $Y$ are sets of vertices, then

$$f(X, Y) = \sum_{x \in X} \sum_{y \in Y} f(x, y)$$

Thus, the flow-conservation constraint can be expressed as the condition that $f(u, V) = 0$ for all $u \in V - \{s, t\}$. Also, for convenience, we shall typically omit set braces when they would otherwise be used in the implicit summation notation. For example, in the equation $f(s, V - s) = f(s, V)$, the term $V - s$ means the set $V - \{s\}$. For capacity, we also give the similar expression:

$$c(X, Y) = \sum_{x \in X} \sum_{y \in Y} c(x, y)$$

A cut $(S, T)$ of flow network $G = (V, E)$ is a partition of $V$ into $S$ and $T = V - S$ such that $s \in S$ and $t \in T$. If $f$ is a flow, then the network flow across the cut $(S, T)$ is defined to be $f(S, T)$. The capacity of the cut $(S, T)$ is $c(S, T)$. Notice that, if $T \notin V - S$, $c(S, T)$ isn't a cut of flow network $G = (V, E)$. A minimum cut of a flow network is a cut whose capacity is minimum over all cuts of the network. By the max-flow min-cut theorem, the value of the maximum flow in a flow network $G$ equals to the capacity of the minimum cut of $G$.

**2.2. Review of max-flow/min-cut algorithms.** Finding a maximum flow in a directed graph with edge capacities arises in many applications, and efficient algorithms for the problem have received a great deal of attention. There are polynomial

algorithms for max-flow problems on directed weighted graphs with two terminals. Most of these algorithms belong to one of the two groups: Ford-Fulkerson style "augmenting paths" algorithms [13] and Goldberg-Tarjan style "push-relabel" algorithms [15].

Standard augmenting paths-based algorithm works by pushing flow along non-saturated paths, named augmenting paths, in residual network $G_f$, from the source to the sink until no augmenting path in the residual network $G_f$ can be found. The residual network has the identical topology to $G$ but the capacity of an edge in $G_f$ reflects the residual capacity of the same edge in $G$ given the amount of flow already in the edge. The max-flow min-cut theorem shows that: if the residual network $G_f$ contains no augmenting paths, the flow $f$ in $G$ is a maximum flow and the maximum flow equals to the minimum cut of $G$. The basic augmenting paths-based algorithm begin with an initialization there is no flow from the source to sink and edge capacities in the residual network $G_f$ are equal to the original capacities in $G$. Then in each iteration, we find some augmenting path $p$ and increase the flow $f$ on each edge of $p$ by the residual capacity $c_f(p)$, which contains the minimum capacity of all the edges in path $p$. The iteration stops and the maximum flow are obtained when no augmenting path can be found.

The Dinic algorithm [12] is one of the fastest implementation among the augmenting paths-based algorithms. It uses breadth-first search to find the shortest paths from $s$ to $t$ on the residual graph $G_f$. After all shortest paths of a fixed length $k$ are saturated, the algorithm starts the breadth-first search paths of length $k+1$. The time complexity of the Dinic algorithm is $O(MN^2)$, where $M$ is the number of edges and $N$ is the number of nodes in $G$. Note that the use of the shortest paths is an important factor that improves theoretical running time complexities for algorithms based on augmenting paths.

Boykov and Kolmogorov [5] propose an augmenting path algorithm whose worst case time complexity $O(MN^2\,|C|)$ is not polynomial, where $|C|$ is the minimum cut or maximum flow found by the algorithm. The complexity is worse than Dinic's algorithm, but in practice [5] significantly outperforms it and other polynomial algorithms , and can work well in the context of relatively sparse grids widely used on computer vision. But, as [5] notes, empirical performance of the algorithm deteriorates on denser (larger neighborhood) grids and when moving from 2D to 3D applications. In fact, 3D grids are widely used in vision and larger neighborhoods are known to reduce geometric artifacts [4]. In [19] Juan and Boykov use capacity scaling approach to accelerate the algorithm in [5] suited to 3D applications with denser neighborhoods. In [25], Liu and Sun propose a novel adaptive bottom-up approach to parallelize the algorithm in [5].

Push-relabel algorithms [14] work in a more localized manner than the Ford-Fulkerson methods. Push-relabel algorithms work on one vertex at a time, looking only at the vertex's neighbors in the residual network, rather than examine the entire residual network $G$ to find an augmenting path. Furthermore, unlike the Ford-Fulkerson method, push-relabel algorithms do not maintain the flow-conservation property throughout their execution. They maintain a preflow, which satisfies skew symmetry, capacity constraints, and the relaxation of flow conservation such that there is an excess flow at each vertex other than the source is nonnegative. The algorithms give every vertex a height. The excess flow is only pushed from a higher vertex to a lower vertex. When there aren't vertices which are lower than vertex $u$ with some excess flows, we must increase its height more than the height of the lowest of

its neighbors to which it has an unsaturated edge. The operation is called relabeling vertex $u$. Eventually, all the flow that can possibly get through to the sink has arrived there, and the residual excess flows are sent back to the source by continuing to relabel vertices to above the fixed height $n$ of the source. Thus, the preflow becomes a legal flow, and is also a maximum flow. The generic push-relabel algorithm has a simple implementation that runs in $O(MN^2)$ time, and the relabel-to-front algorithm [15, 8] refines to obtain another push-relabel algorithm that run in $O(N^3)$ time.

**2.3. The Motivation and Method Overview.** Besides developing $st$-mincut algorithms with lower time complexity, many useful strategies have been presented to increase the efficiency of the $st$-mincut algorithms in vision applications.

***Dynamically reusing flow*** Boykov and Jolly [7] use a partially dynamic $st$-mincut algorithm in a vision application by proposing a technique with which they could update capacities of $t$-link [7] of certain graph edges and recomputed the $st$-mincut dynamically. They used this method for performing interactive image segmentation, where the user could improve segmentation results by giving additional segmentation cues (seeds) in an online fashion. Specially, they described a method for updating the cost of $t$-link in the graph. Comparably, Kohli and Torr [20] presented a fully dynamic algorithm for the $st$-mincut problem, which allows for arbitrary changes in the graph. Specifically, given the solution of the max-flow problem on a graph, the dynamic algorithm efficiently computes the maximum flow in a modified version of the graph, and the time taken by it is roughly proportional to the total amount of change in the edge weighs of the graph.

***Dynamically reusing cut*** In [18, 17], Juan et al. proposed an active graph cuts algorithm in which they reused the $s$t-mincut solution corresponding to the previous instance to generate an initialization. Specially, in video segmentation, the active cuts algorithm can use the $st$-mincut solution of previous frames for accelerating to compute the $st$-mincut of the current frame, and the running time is proportional to the amount of motion between two consecutive frames.

Active cuts initialized the algorithm with any arbitrary cut $C_0$ and use the trick in [20] to saturate every edges going from source part to the sink part. The notion of pseudoflow is introduced to develop a symmetric Push-Pull-Relabel framework and the new $t$-links produced by the saturated edges of the initialized cut can be transformed into excess and deficit nodes. Then the original graph $G$ was split in two sub-graphs: the source graph $G_1$ which lie the deficit nodes and the sink graph $G_2$ which lie the excess nodes. The push-relabel algorithm was applied to $G_2$ to process the excess to get the min-cut $C^+$ of $G_2$ and the pull-relabel was applied to $G_1$ to process the deficit to get the min-cut $C^-$ of $G_1$, as shown in Fig.1. Juan [17] illustrated the min-cut $C_{\min}$ of $G$ lies in the gray zone in Fig. 2.1 (row 1). Therefore, the remaining work will focus only on the gray part $G_3$, and the next step of the active cuts algorithm consists into alternatively converting back every excesses or every deficits into $t$-links and relaunching the algorithm until convergence (see Fig. 2.1 (row 2)). Finally, the min-cut $C_{\min}$ of $G$ can be obtained, as shown in Fig. 2.1 (row 3). More detail can be found in [17].

It should be noticed that, although [17] proposes a good idea to decompose the original graph $G$ into two subgraphs $G_1$ and $G_2$ and respectively compute their min-cuts, and illustrates the min-cut $C_{\min}$ of $G$ lies in the gray zone in Fig. 2.1 (row 1), they don't provide one strict proof and just give some descriptive notes. That is to say, in [17] Juan doesn't strictly prove that the source part of $G_1$ (the red part shown in Fig. 2.1 (row 1)) belongs to the source part of $G$, and the sink part of $G_2$ (the

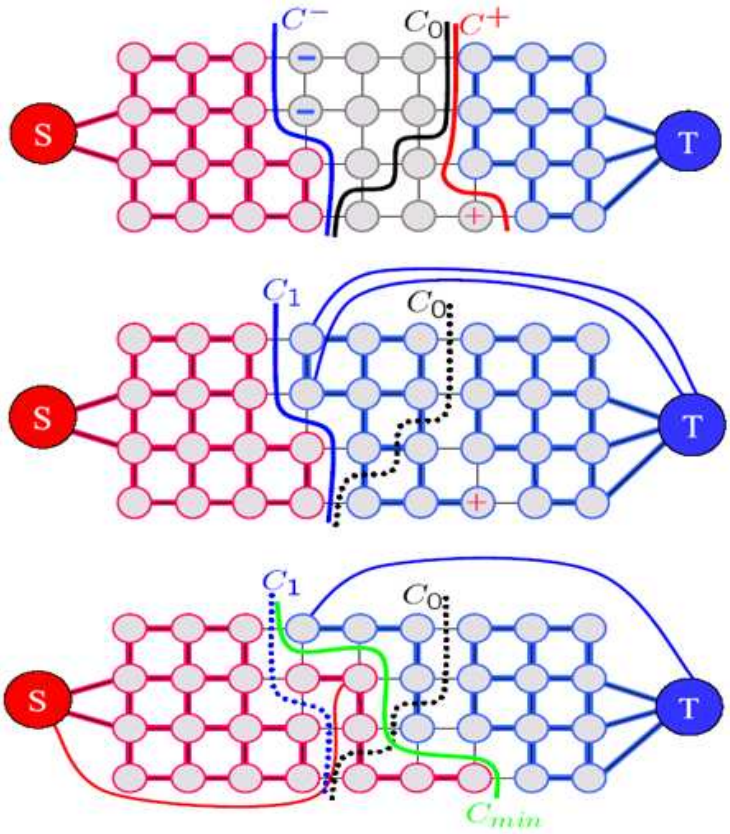


FIG. 2.1. *Active cuts method illustrations comes from [17].*

blue part shown in Fig. 2.1 (row 1)) belongs to the sink part of G from the point of view of graph theory. And converting back all the remaining excesses and deficits into $t$-links to compute the last min-cut of $G$ also hasn't theoretic support. Since the source code in [17] is unavailable, we have implemented this algorithm by ourselves and conducted a number experiment to test it. We found in some cases the solution by the algorithm in [17] is just a good approximation of the original push-relabel algorithm despite that the approximate error is very small, just several nodes, from that we infer that the construction of the subgraphs $G_1$, $G_2$ and $G_3$ in [18, 17] isn't equivalent to the original graph. We will develop an improved subgraph construction method to solve the problem in [17] and provide strict proof that the min-cuts of the decomposed subgraphs are equivalent to the min-cuts of the original graph.

***Method Overview*** In this paper, we will propose a general recursive minimum cuts (RMC) framework. Any min-cut algorithms can be used for the subgraph problems. Especially, the solution by the recursive framework is exactly the same as the original minimum cuts algorithm, not an approximation. If the time complexity of the integrated min-cut algorithms is $O(N^p)$, we can prove that the total time complexity of the proposed RMC algorithm is $O(N^{\frac{\log 3}{\log 2}})$. This is implemented based on the proposed graph decomposition method and minimum cuts composition theorem. We use Fig. 2.2 to illustrate the basic ideas of our general framework.

Give a graph $G$ and an arbitrary initial cut $c_0(S_0, T_0)$, shown in row 1 of Fig. 2.2, we can construct two subgraphs $G_1$ and $G_2$, shown in row 2 of Fig. 2.2. The nodes in $G_1$ are $S_0 + t$, where t is the sink node of $G_1$ and the nodes in $G_2$ are as $T_0 + s$, where $s$ is the source node of $G_2$. We can use one min-cut algorithm to respectively compute the minimum cuts $c_1(S_1, T_1)$ and $c_2(S_2, T_2)$ of $G_1$ and $G_2$. We then construct $G_3$ according to $c_1(S_1, T_1)$ and $c_2(S_2, T_2)$. The nodes in $G_3$ are $T_1 + S_2$. The selected min-cut algorithm is used to compute the minimum cut $c_3(S_3, T_3)$ of $G_3$. Based on our constructing method of the subgraphs $G_1$, $G_2$ and $G_3$, we can prove that the minimum cut $c(S, T)$ of $G$ is $c(S_1 + S_3 - s, T_2 + T_3 - t)$. Using this observation, we can compute the minimum cut $c(S, T)$ of G by respectively computing the minimum cuts $c_1(S_1, T_1)$, $c_2(S_2, T_2)$ and $c_3(S_3, T_3)$ of $G_1$, $G_2$ and $G_3$ given an initial cut $c_0(S_0, T_0)$ of $G$. Assume that we use a min-cut algorithms with time complexity $O(N^3)$, then the mean computational time using this graph decomposition method is decreased to 3/8 of the computation time of original algorithm. The graph decomposition procedure can be recursively continued until the subgraphs can't be decomposed any more and the time complexity of this recursive minimum cuts algorithm is $O(N^{\frac{\log 3}{\log 2}})$. A detailed analysis will be given in section 5. Based on the graph decomposition and minimum cuts composition framework, we can also construct a parallel algorithm with the time complexity on the order of $O(N)$ time using $N$ processors.

**3. Graph Decomposition.** Given a flow network $G = (V, E)$, let cut $(S, T)$ be the minimum cut of $G$, named $c(S, T)$. Given $(S_0, T_0)$ be one arbitrary cut of $G$, we call it initial cut $c(S_0, T_0)$.

We construct flow network $G_1 = (V_1, E_1)$ by $G$ and $c(S_0, T_0)$ as follows:

$$V_1 = S_0 + t, , \quad E_1 = E_{1a} + E_{1b}. \tag{3.1}$$

where

$$\begin{aligned} E_{1a} &= \{(u, v) | (u, v) \in E,\ u, v \in S_0\}, \\ E_{1b} &= \{(u, t) | (u, v) \in E, u \in S_0, v \in T_0\}. \end{aligned} \tag{3.2}$$

From equations (3.2), the following relation holds

$$c_1(u, v) = c(u, v) \quad u, v \in S_0, \quad c_1(u, t) = \sum_{v \in T_0} c(u, v) = c(u, T_0) \quad u \in S_0. \tag{3.3}$$

Therefore,

$$c_1(S_0, t) = c(S_0, T_0). \tag{3.4}$$

Let cut $(S_1, T_1)$ be the minimum cut of $G_1$, named $c_1(S_1, T_1)$. We also construct flow network $G_2 = (V_2, E_2)$ from $G$ and $c(S_0, T_0)$, where

$$V_2 = T_0 + s, \quad E_2 = E_{2a} + E_{2b}. \tag{3.5}$$

and

$$\begin{aligned} E_{2a} &= \{(u, v) | (u, v) \in E, u, v \in T_0\}, \\ E_{2b} &= \{(s, v) | (u, v) \in E, u \in S_0, v \in T_0\}. \end{aligned} \tag{3.6}$$

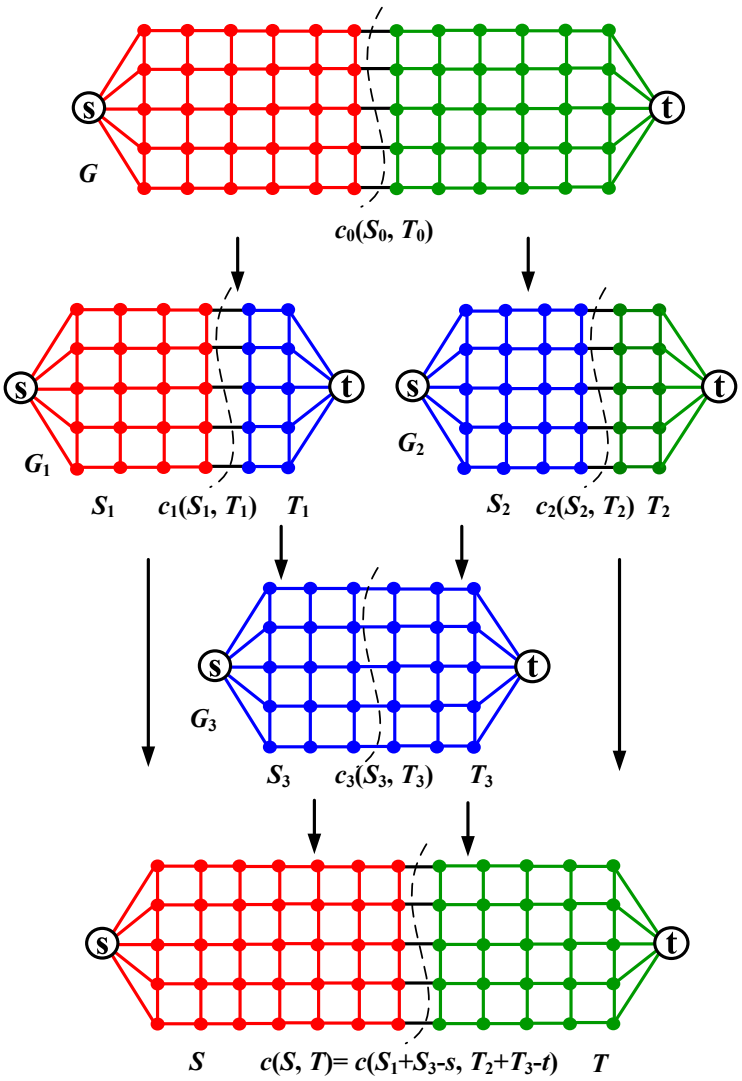


FIG. 2.2. *Graph decomposition and minimum cuts composition illustration.*

From equations (3.6), we see that

$$\begin{aligned} c_2(u,v) &= c(u,v) \quad u,v \in T_0, \\ c_2(s,v) &= \sum_{u\in S_0} c(u,v) = c(S_0,v) \quad v \in T_0. \end{aligned} \tag{3.7}$$

Therefore,

$$c_2(s,T_0) = c(S_0,T_0). \tag{3.8}$$

Let cut $(S_2,T_2)$ be the minimum cut of $G_2$, named $c_2(S_2,T_2)$. We further construct $G_3=(V_3,E_3)$ from $G_1$ and $G_2$ as

$$V_3 = T_1 + S_2, \quad E_3 = E_{3a} + E_{3b} + E_{3c}. \tag{3.9}$$

with

$$\begin{aligned} E_{3a} &= \{(u,v)|(u,v)\in E, u,v\in V_3 - s - t\},\\ E_{3b} &= \{(s,v)|(u,v)\in E, u\in S_1, v\in V_3 - s - t\},\\ E_{3c} &= \{(u,t)|(u,v)\in E, u\in V_3 - s - t, v\in T_2\}. \end{aligned} \tag{3.10}$$

Obviously, the following relation holds using (3.10),

$$\begin{aligned} c_3(u,v) &= c(u,v) \quad u,v\in V_3 - s - t,\\ c_3(s,v) &= \sum_{u\in S_1} c(u,v) = c(S_1,v) \quad v\in V_3 - s - t,\\ c_3(u,t) &= \sum_{v\in T_2} c(u,v) = c(u,T_2) \quad u\in V_3 - s - t,\\ c_3(s,t) &= 0. \end{aligned} \tag{3.11}$$

We denote $(S_3, T_3)$ the minimum cut of $G_3$, named $c_3(S_3, T_3)$.

**4. Minmum Cuts Composition .** LEMMA 4.1. *Give a flow network $G = (V, E)$ and let $c(S, T)$ be the minimum cut of $G$. Let $c(S_0, T_0)$ be one arbitrary initial cut of $G$. Assume that the flow network $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are constructed according to (3.1), (3.2), (3.5) and (3.6) respectively, and their minimum cuts are respectively $c_1(S_1, T_1)$ and $c_2(S_2, T_2)$. The following inequalities hold.*

$$\begin{aligned} &c_1(S_1,T_1)\le c(S_0,T_0), c_2(S_2,T_2)\le c(S_0,T_0),\\ &c(S,T)\le c_1(S_1,T_1), c(S,T)\le c_2(S_2,T_2). \end{aligned}$$

*Proof.* Based on the construction of $G_1$ and (3.4), we have $c_1(S_1, T_1) \le c_1(V_1 - t, t) = c(S_0, T_0)$. Here we have also used the fact that $c_1(V_1 - t, t)$ is one cut of $G_1$ and $c_1(S_1, T_1)$ that is the minimum cut of $G_1$. Similarly, according to (3.8), we have $c_2(S_2, T_2) \le c_2(s, V_2 - s) = c(S_0, T_0)$ thanks to the fact that $c_2(s, V_2 - s)$ is one cut of $G_2$ and $c_2(S_2, T_2)$ that is the minimum cut of $G_2$.

Using (3.1) and (3.3), we get

$$\begin{aligned} c_1(S_1,T_1) &= c_1(S_1,T_1 - t) + c_1(S_1,t) = c(S_1,T_1 - t) + c(S_1,T_0)\\ &= c(S_1,T_1 + T_0 - t) = c(S_1,S_0 + T_0 - S_1) = c(S_1,V - S_1). \end{aligned}$$

Because $c(S_1, V - S_1)$ is a cut of flow network $G$ and $c(S, T)$ is the minimum cut of $G$, we have

$$c_1(S_1,T_1) = c(S_1,V - S_1)\ge c(S,T).$$

By Formulas (3.5) and (3.7), we have

$$\begin{aligned} c_2(S_2,T_2) &= c_2(S_2 - s,T_2) + c_2(s,T_2) = c(S_2 - s,T_2) + c(S_0,T_2)\\ &= c(S_2 + S_0 - s,T_2) = c(T_0 + S_0 - T_2,T_2) = c(V - T_2,T_2). \end{aligned}$$

Because $c(V - T_2, T_2)$ is a cut of flow network $G$ and $c(S, T)$ is the minimum cut of $G$, we have

$$c_2(S_2,T_2) = c(V - T_2,T_2)\ge c(S,T).$$

This proves the result. □

COROLLARY 4.2. *For a given flow network $G = (V, E)$, assume $c(S, T)$ is the minimum cut of $G$. Let $c(S_0, T_0)$ be one arbitrary initial cut of $G$. The flow network $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are constructed according to Equations (3.1), (3.2), (3.5) and (3.6) respectively. Let their minimum cuts be $c_1(S_1, T_1)$ and $c_2(S_2, T_2)$ respectively. If $S \subseteq S0$, then we have $S = S_1$. If $T \subseteq T_0$, then we have $T = T_2$.*

*Proof.* Since $S \subseteq S_0$, let $S_0 = S + W$, then $T_0 = T - W$. From (3.3), we get

$$\begin{aligned} c(S, T) &= c(S, T_0 + W) = c(S, T_0) + c(S, W) \\ &= c_1(S, t) + c_1(S, W) = c_1(S, W + t). \end{aligned}$$

From Lemma 4.1, we have $c_1(S, W + t) = c(S, T) \leq c_1(S_1, T_1)$. Using (3.1), we get that $V_1 = S_0 + t = S + W + t$, thus $c_1(S, W + t)$ is a cut of $G_1$. Since $c_1(S_1, T_1)$ is the minimum cut of $G_1$, then we have

$$c_1(S, W + t) \geq c_1(S_1, T_1).$$

Therefore, it is true that $c_1(S, W + t) = c_1(S_1, T_1)$ , thus $S = S_1$ holds.

Similarly, we can prove that if $T \subseteq T_0$, then $T = T_2$ holds.

□

THEOREM 4.3. *For a given flow network $G = (V, E)$, assume that $c(S, T)$ is the minimum cut of $G$. Let $c(S_0, T_0)$ be one arbitrary initial cut of $G$. The flow network $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are constructed according to Equations (3.1), (3.2), (3.5) and (3.6) respectively. Let their minimum cuts be $c_1(S_1, T_1)$ and $c_2(S_2, T_2)$ respectively. Then, we have $S_1 \subseteq S$ and $T_2 \subseteq T$.*

*Proof.* Let $W_1 = \{(u, v) | (u, v) \in S_1 \text{ and } (u, v) \in S\}$, $W_2 = \{(u, v) | (u, v) \in S_1 \text{ and } (u, v) \notin S\}$, $W_3 = \{(u, v) | (u, v) \notin S_1 \text{ and } (u, v) \in S\}$. Then we have $s \in W_1$. In addition, $W_1$, $W_2$ and $W_3$ aren't intersectant. Therefore we have

$$S = W_1 + W_3, S_1 = W_1 + W_2, T = V - W_1 - W_3, T_1 = V_1 - W_1 - W_2,$$

$$c_1(S_1, T_1) = c_1(W_1 + W_2, V_1 - W_1 - W_2) = c_1(W_1, V_1 - W_1) - c_1(W_1, W_2) + c_1(W_2, T_1).$$

As $c_1(W_1, V_1 - W_1)$ is one cut of $G_1$ and $c_1(S_1, T_1)$ is the minimum cut of $G_1$, we see that

$$c_1(W_1, V_1 - W_1) - c_1(S_1, T_1) = c_1(W_1, W_2) - c_1(W_2, T_1) \geq 0.$$

Using (3.3), we get

$$c_1(W_1, W_2) = c(W_1, W_2),$$

$$\begin{aligned} c_1(W_2, T_1) &= c_1(W_2, T_1 - t) + c_1(W_2, t) \\ &= c(W_2, T_1 - t) + c(W_2, T_0) = c(W_2, V - S_1). \end{aligned}$$

Therefore. we can deduce that

$$c(W_1, W_2) - c(W_2, V - S_1) \geq 0. \tag{4.1}$$

We know that $c(W_1+W_2+W_3, V-W_1-W_2-W_3)$ is a cut of $G=(V,E)$, then

$$c(W_1+W_2+W_3, V-W_1-W_2-W_3) \geq c(S,T) = c(W_1+W_3, V-W_1-W_3). \quad (4.2)$$

Observe that

$$\begin{aligned}
&c(W_1+W_2+W_3, V-W_1-W_2-W_3)\\
&= c(W_1+W_3, V-W_1-W_3) - c(W_1+W_3, W_2) + c(W_2, V-W_1-W_2-W_3)\\
&= c(S,T) - c(W_1+W_3, W_2) + c(W_2, V-S_1-W_3)\\
&= c(S,T) - c(W_1,W_2) - c(W_3,W_2) + c(W_2, V-S_1) - c(W_2,W_3).
\end{aligned}$$

By Equation (4.2), it is true that

$$c(W_1,W_2) - c(W_2, V-S_1) + c(W_3,W_2) + c(W_2,W_3) \leq 0.$$

Using (4.1), we get

$$c(W_1,W_2) - c(W_2, V-S_1) + c(W_3,W_2) + c(W_2,W_3) \geq 0.$$

Thus, we have

$$c(W_1,W_2) - c(W_2, V-S_1) + c(W_3,W_2) + c(W_2,W_3) = 0.$$

Accordingly,

$$c(W_1+W_2+W_3, V-W_1-W_2-W_3) = c(S,T) = c(W_1+W_3, V-W_1-W_3).$$

Therefore we have $W_2 = \Phi$. Thus $S_1 \subseteq S$ holds.

Similarly, we can prove $T_2 \subseteq T$ as follows. Let $W_1 = \{(u,v)|(u,v) \in T_2 \; and \; (u,v) \in T\}$, $W2 = \{(u,v)|(u,v) \in T_2 \; and \; (u,v) \notin T\}$, $W_3 = \{(u,v)|(u,v) \notin T_2 \; and \; (u,v) \in T\}$. Then we have $t \in W_1$. In addition, $W_1$, $W_2$ and $W_3$ aren't intersectant. Therefore we have

$$T_2 = W_1 + W_2, T = W_1 + W_3, S_2 = V_2 - W_1 - W_2, S = V - W_1 - W_3,$$

$$c_2(S_2,T_2) = c_2(V_2 - W_1 - W_2, W_1 + W_2) = c_2(V_2 - W_1, W_1) - c_2(W_2, W_1) + c_2(S_2, W_2).$$

Since $c_2(V_2 - W_1, W_1)$ is one cut of $G_2$ and $c_2(S_2,T_2)$ is the minimum cut of $G_2$, then

$$c_2(V_2 - W_1, W_1) - c_2(S_2,T_2) = c_2(W_2,W_1) - c_2(S_2,W_2) \geq 0.$$

By Equation (3.7), we have

$$c_2(W_2,W_1) = c(W_2,W_1),$$

$$c_2(S_2,W_2) = c_2(S_2 - s, W_2) + c_2(s,W_2) = c(S_2 - s, W_2) + c(S_0, W_2) = c(V - T_2, W_2).$$

Then we have

$$c(W_2,W_1) - c(V - T_2, W_2) \geq 0. \quad (4.3)$$

We know that $c(V - W_1 - W_2 - W_3, W_1 + W_2 + W_3)$ is a cut of $G=(V,E)$, then

$$c(V - W_1 - W_2 - W_3, W_1 + W_2 + W_3) \geq c(S,T) = c(V - W_1 - W_3, W_1 + W_3). \quad (4.4)$$

We also have

$$\begin{aligned}
&c(V - W_1 - W_2 - W_3, W_1 + W_2 + W_3) \\
&= c(V - W_1 - W_3, W_1 + W_3) - c(W_2, W_1 + W_3) + c(V - W_1 - W_2 - W_3, W_2) \\
&= c(S, T) - c(W_2, W_1 + W_3) + c(V - T_2 - W_3, W_2) \\
&= c(S, T) - c(W_2, W_1) - c(W_2, W_3) + c(V - T_2, W_2) - c(W_3, W_2).
\end{aligned}$$

By Equation (4.4),

$$c(W_2, W_1) - c(V - T_2, W_2) + c(W_2, W_3) + c(W_3, W_2) \leq 0.$$

By Equation (4.3),

$$c(W_2, W_1) - c(V - T_2, W_2) + c(W_2, W_3) + c(W_3, W_2) \geq 0.$$

Then we have

$$c(W_2, W_1) - c(V - T_2, W_2) + c(W_2, W_3) + c(W_3, W_2) = 0,$$

$$c(V - W_1 - W_2 - W_3, W_1 + W_2 + W_3) = c(S, T) = c(V - W_1 - W_3, W_1 + W_3).$$

Therefore we have $W_2 = \Phi$. Thus $T_2 \subseteq T$ holds.
□

THEOREM 4.4. *For a give flow network $G = (V, E)$ assume that $c(S, T)$ is the minimum cut of $G$. Let $c(S_0, T_0)$ be one arbitrary initial cut of $G$. The flow network $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$ and $G_3 = (V_3, E_3)$ are constructed according to Equations (3.1), (3.2), (3.5), (3.6), (3.9) and (3.10) respectively. Let minimum cuts be $c_1(S_1, T_1)$, $c_2(S_2, T_2)$ and $c_3(S_3, T_3)$ respectively. Then, we have $S = S_1 + S_3 - s$ and $T = T_2 + T_3 - t$.*

*Proof.* Let $c_3(S_3^{'}, T_3^{'})$ is an arbitrary cut of $G_3 = (V_3, E_3)$, then

$$c_3(S_3^{'}, T_3^{'}) = c_3(S_3^{'} - s, T_3^{'} - t) + c_3(S_3^{'} - s, t) + c_3(s, T_3^{'} - t) + c_3(s, t).$$

By Equation (3.11),

$$c_3(S_3^{'} - s, T_3^{'} - t) = c(S_3^{'} - s, T_3^{'} - t),$$

$$c_3(S_3^{'} - s, t) + c_3(s, T_3^{'} - t) = c(S_3^{'} - s, T_2) + c(S_1, T_3^{'} - t),$$

$$c_3(s, t) = 0.$$

Then, we have

$$\begin{aligned}
&c_3(S_3^{'}, T_3^{'}) = c(S_3^{'} - s, T_3^{'} - t) + c(S_3^{'} - s, T_2) + c(S_1, T_3^{'} - t) \\
&= c(S_1 + S_3^{'} - s, T_2 + T_3^{'} - t) - c(S_1, T_2).
\end{aligned}$$

Accordingly, we have

$$c(S_1 + S_3^{'} - s, T_2 + T_3^{'} - t) = c_3(S_3^{'}, T_3^{'}) + c(S_1, T_2). \tag{4.5}$$

By Lemma 3.3, $S_1 \subseteq S$ and $T_2 \subseteq T$, we let $S = S_1 + S_3^{''} - s$ and $T = T_2 + T_3^{''} - t$. Then we have $S_3^{''} + T_3^{''} = S + T - S_1 - T_2 + s + t = S_0 + T_0 - S_1 - T_2 + s + t = V_1 - S_1 + V_2 - T_2 = T_1 + S_2 = V_3$. Therefore $c_3(S_3^{''}, T_3^{''})$ is a cut of $G_3 = (V_3, E_3)$. By Equation (4.5), we have the following two equalities:

$$c(S_1 + S_3^{''} - s, T_2 + T_3^{''} - t) = c_3(S_3^{''}, T_3^{''}) + c(S_1, T_2),$$

$$c(S_1 + S_3 - s, T_2 + T_3 - t) = c_3(S_3, T_3) + c(S_1, T_2).$$

Since $c_3(S_3, T_3)$ is the minimum cut of $G_3$, we have

$$\begin{aligned} c(S,T) &= c(S_1 + S_3^{''} - s, T_2 + T_3^{''} - t) = c_3(S_3^{''}, T_3^{''}) + c(S_1, T_2) \\ &\geq c_3(S_3, T_3) + c(S_1, T_2) = c(S_1 + S_3 - s, T_2 + T_3 - t). \end{aligned}$$

But $c(S,T)$ is the minimum cut of $G = (V, E)$ and $c(S_1 + S_3 - s, T_2 + T_3 - t)$ is one cut of $G$ by Equation (3.1), (3.5) and (3.9), thus we have

$$c(S,T) \leq c(S_1 + S_3 - s, T_2 + T_3 - t).$$

Therefore, $c(S,T) = c(S_1 + S_3 - s, T_2 + T_3 - t)$. This gives us the conclusion that $S = S_1 + S_3 - s, T = T_2 + T_3 - t$. □

**5. Recursive Minimum Cuts(RMC) Algorithm.** Now we shall design our recursive minimum cuts algorithm by the graph construction method in Section 2 and the graph cuts composition theorems in Section 3. By theorem 3.4, we know that $c(S,T)$ of $G = (V, E)$ can be computed from $c_1(S_1, T_1)$ of $G_1 = (V_1, E_1)$, $c_2(S_2, T_2)$ of $G_2 = (V_2, E_2)$, and $c_3(S_3, T_3)$ of $G_3 = (V_3, E_3)$ given an arbitrary initial cut $c(S_0, T_0)$ of $G$, where $G_1$, $G_2$ and $G_3$ are constructed according to Equations (3.1), (3.2), (3.5), (3.6), (3.9) and (3.10) respectively. Since most of the present min-cuts algorithms have approximate $O(N^3)$ time complexity, the graph decomposition and minimum cuts composition method can accelerate the min-cuts algorithm. It should be noticed when $G$ is decomposed to $G_1$ and $G_2$ with the same number of nodes, the method is possible to obtain the "best" acceleration.

The graphs $G_1$, $G_2$ and $G_3$ can be recursively decomposed and we can use Theorem 3.4 to obtain the minimum cuts for the subgrapghs. Before we go to the details of the *Recursive Minimum Cuts* (RMC), let us first supply an algorithms for the construction of the three subgraphs.

Assumes that the input graph $G = (V, E)$ and the output graphs $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$ and $G_3 = (V_3, E_3)$ are represented using adjacency lists. Let $N$, $N_1$, $N_2$, $N_3$ be respectively the number of nodes of $G$, $G_1$, $G_2$ and $G_3$ in addition to the source and sink nodes. Let $c(S,T)$, $c_1(S_1, T_1)$, $c_2(S_2, T_2)$ and $c_3(S_3, T_3)$ be respectively the minimum cuts of $G$, $G_1$, $G_2$ and $G_3$. The algorithm need to maintain several data structures with each vertex in the graph. The color of each vertex $u \in \{V - s - t\}$ is stored in the variable $color[u]$, which denotes the attribute of vertex $u$, i.e., which subgraph of $G_1$, $G_2$ and $G_3$ vertex $u$ belongs to. Assume that $color[u] = RED$ means $u \in V_1$, $color[u] = GREEN$ means $u \in V_2$, $color[u] = BLUE$ means $u \in V_3$. Let $map[u]$ be the corresponding node number in $G1$, $G_2$ and $G_3$ of the node $u$ in $G$. Then $color[u]$ and $map[u]$ can decide the attribute of the node $u$ belonging to $G_1$, $G_2$ and $G_3$. For example, $color[u] = RED$ and $map[u] = k$, means node $u$ in $G$ corresponds to the $k^{th}$ node in $G_1$. Assume that $map1[u]$ is the corresponding node number in $G$ of node $u$ from $G_1$, $map2[u]$ is the corresponding node number in $G$ of node $u$ from

```
ConstructSubgraph12(G, G1, G2)
////////////////////////////////////////////////////////////////////////////////
1    get one arbitrary vertex r∈ Adj[s]
2    for each vertex u∈ V-s-t -r
3         do color[u]←WHITE
4    color[r]←GRAY
5    ENQUEUE (Q, r)
6    i=1
7    while Q ≠ Φ
8         do u ← DEQUEUE (Q)
9              for each v ∈ Adj[u]
10                  do   if color[v]=WHITE
11                       then color[v]← GRAY
12                            ENQUEUE (Q, v)
13             color[u] ← RED
14             i ← i+1
15             map[u] ← i
16             map1[i] ← u
17             if i>N/2
18                  then break while
19   N1 ← i+2, V1={u| color[ u]=RED}+s+t
20   i=0
21   for each vertex u ∈ V-s-t
22        do if color[u] ≠ RED
23                  then color[u]← GREEN
24                       i ← i+1
25                       map[u] ← i
26                       map2[i]← u
27   N2 ← i+2, V2={u| color[u]=GREEN}+s+t
////////////////////////////////////////////////////////////////////////////////
28   color[s] ← RED color[t] ← GREEN
29   for each vertex u ∈ V1-t
30        c1(u,t)=0
31   for each vertex u ∈ V2-s
32        c2(s,u)=0
33   for each vertex u ∈ V
34        for each v ∈ Adj[u]
35             do   if color[u]= color[v]=RED
36                       then c1(map[u],map[v])= c(u,v)
37                  if color[u]= color[v]=GREEN
38                       then c2(map[u],map[v])= c(u,v)
39                  if color[u]= RED and color[v]=GREEN
40                       then c1(map[u],t)= c1(map[u],t)+c(u,v)
41                            c2(s,map[v])= c2(s, map[v])+c(u,v)
```

FIG. 5.1. *The construction algorithm of subgraphs* $G_1 = (V_1, E_1)$ *and* $G_2 = (V_2, E_2)$ *according to Equations (3.1), (3.2), (3.5) and (3.6) from graph* $G = (V, E)$.

$G_2$ and $map3[u]$ is the corresponding node number in $G$ of node $u$ from $G_3$. With this notation, $map1[u_1] = k_1$ indicates that node $u_1$ from $G_1$ corresponds to the $k_1^{th}$ node in $G$, $map2[u_2] = k_2$ indicates that node $u_2$ from $G_2$ corresponds to the $k_2^{th}$ node in $G$ and $map3[u_3] = k_3$ indicates that node $u_3$ from $G_3$ corresponds to the $k_3^{th}$ node in $G$.

The label of each vertex $u \in \{V - s - t\}$ is stored in the variable $label[u]$, which denotes which part of $G$, source part or sink part, vertex $u$ belongs to. Assume

```
ConstructSubgraph3(G, G3, S1, T1, S2, T2);
////////////////////////////////////////////////////////////////////////////////
1    for each vertex u ∈ S1
2        do color[u] ⟵ RED
3    for each vertex u ∈ T1-t
4        do color[u] ⟵ BLUE
5    for each vertex u ∈ S2-s
6        do color[u] ⟵ BLUE
7    for each vertex u ∈ T2
8        do color[u] ⟵ GREEN
9    i=0
10   for each vertex u ∈ V-s-t
11       do if color[u] = BLUE
12              then i ⟵ i+1
13                   map[u] ⟵ i
14                   map3[i] ⟵ u
15   N3 ⟵ i+2, V3={u| color [u]= BLUE }+s+t
16   for each vertex u ∈ V3-s-t
17       c1(u,t)= c1(s,u)=0
18   for each vertex u ∈ V
19       for each v ∈ Adj[u]
20           do  if color[u]= color[v]= BLUE
21                   then c3(map[u],map[v])= c(u,v)
22               if color[u]=RED and color[v]= BLUE
23                   then c3(s,map[v])= c3(s,map[v])+c(u,v)
24               if color[u]= BLUE and color[v]=GREEN
25                   then c3(map[u],t)= c3(map[u],t)+c(u,v)
```

FIG. 5.2. *The construction algorithm of subgraph* $G_3 = (V_3, E_3)$ *based on the min-cuts* $c_1(S_1, T_1)$ *and* $c_2(S_2, T_2)$ *of* $G_1$ *and* $G_2$ *according to Equations (3.9)and (3.10)from graph* $G = (V, E)$.

that $label[u] = SOURCE$ means $u \in S$ and $label[u] = SINK$ means $u \in T$. The algorithm for the construction of $G_1$, $G_2$ is similar to $breadth - first - search(BFS)$. The number of nodes in $G_1$ is stored in the variable $i$. Let $r \in Adj[s]$. The algorithm also uses a first-in, first-out queue $Q$ to manage the set of gray vertices.

The construction procedure of $G_1$ and $G_2$ is shown in Fig. 5.1. In Line 1 an arbitrary adjacent node $r$ of source node $s$ is chosen. Line 2-18 is a $BFS$ procedure whose root is node $r$ in $G$. But the $BFS$ process terminates when the number of nodes in the produced $BFT$ is beyond $N/2$. The nodes in the $BFT$ belong to $G_1$ and the other nodes in $G$ belong to $G_2$. Line 2-18 and Line 19-27 determine the nodes in $G_1$ and $G_2$. Line 28-41 determines the edges in $G_1$ and $G_2$ and their weights by the construction method in Section 2.

Note that the construction of $G_3$ depends on the minimum cuts $c_1(S_1, T_1)$ and $c_2(S_2, T_2)$ of $G_1$ and $G_2$. The construction procedure of $G_3$ is shown in Fig. 5.2. Line 1-15 determines the nodes in $G_3$ and labels the three different color regions in $G$, where the nodes in $RED$ region belong to $S$ part, the nodes in $GREEN$ region belong to $T$ part and the nodes in $BLUE$ region belong to $G_3$, the undecided region. Line 16-25 determines the edges in $G_3$ and their weights by the construction method in Section 2.

The RMC procedure is shown in Fig. 5.3, where $BasicMinCut(G)$ is the selected basic max-flow/min-cut algorithm, which can be "augmenting paths" style algorithms or "push-relabel" style algorithms. Parameter $i$ is the recursive time. In line 1, the

```
RMC (G, N, S, T)
////////////////////////////////////////////////////////////////////////////////
1     i=[logN/log2]
2     if   i=0
3          do   BasicMinCut(G, S, T)
4     else
5          do   i=i-1
6               ConstructSubgraph12(G, G1, G2)
7               RMC(G1, N1, S1, T1)
8               RMC (G2, N2, S2, T2)
9               ConstructSubgraph3(G, G3, S1, T1, S2, T2)
10              RMC (G3, N3, S3, T3)
11              c(S, T)=c(S1+ S3-s, T2 +T3-t)
```

FIG. 5.3. *The RMC algorithm to compute the min-cut* $c(S,T)$ *of* $G = (V, E)$*, where parameter* $i$ *is determined by the number of nodes in graph.*

```
RMC (G, N, i, S, T)
////////////////////////////////////////////////////////////////////////////////
1     if   i>[logN/log2]
2          do   i=[logN/log2]
3     if   i=0
4          do   BasicMinCut(G)
5     else
6          do   ConstructSubgraph12(G, G1, G2)
7               RMC (G1, N1, i-1, S1, T1)
8               RMC (G2, N2, i-1, S2, T2)
9               ConstructSubgraph3(G, G3, S1, T1, S2, T2)
10              RMC (G3, N3, i-1, S3, T3)
11              c(S, T)=c(S1+ S3-s, T2 +T3-t)
```

FIG. 5.4. *The RMC algorithm to compute the min-cut* $c(S,T)$ *of* $G = (V, E)$*, where parameter* $i$ *is set in advance.*

recursive time is set to $[logN/log2]$, where $x - 1 < [x] \leq x$, $x$ is a positive real value and $[x]$ is a positive integer. If $i = 0$, the basic min-cut algorithm is executed to $G$. If $i > 0$, in line 6 the graph $G$ should be split into $G_1$ and $G_2$ and the recursion is applied to $G_1$ and $G_2$ in line 7 and 8. After get $c_1(S_1, T_1)$ and $c_2(S_2, T_2)$, $G_3$ is constructed by $c_1(S_1, T_1)$ and $c_2(S_2, T_2)$ and the recursion is applied to $G_3$ in line 10. In line 11 the min-cut $c(S, T)$ of $G$ is computed.

In Fig. 5.3, parameter $i$ is determined by the number of nodes in the graph being processed by RMC, which can assure that the processed graph can achieve the maximum recursive time. It means in the last recursion of RMC, the produced subgraphs will include only one or two nodes in addition to the source and sink nodes. In practical applications, if the graph isn't very large just several recursions can lead to high computational efficiency. Therefore, we can input the parameter $i$ in advance.

The procedure is shown in Fig. 5.4. The number of recursion of $G$ is $i$ and the number of recursions of their subgraphs decreases to $i-1$. When the current recursion number decreases to 0, the basic min-cut algorithm will be applied to the current graphs.

**6. Complexity Analysis of RMC Algorithm.** Now we analyze the time complexity of the proposed recursive minimum cuts framework. The time complexity of constructing subgraphs $G_1$, $G_2$ and $G_3$ is linear according to the algorithms presented in Fig. 5.1 and Fig. 5.2 because BFS algorithm is linear. Therefore, the time complexity of our recursive framework algorithm mainly depends on the selected basic min-cut algorithm and our recursive scheme. We first give the following definition.

DEFINITION 6.1. *If $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$ and $G_3 = (V_3, E_3)$ are constructed according to Equations (3.1), (3.2), (3.5), (3.6), (3.9) and (3.10) from $G = (V, E)$ given an initial cut $c(S_0, T_0)$ of $G$. $G$ is called the* parent *graph of $G_1$, $G_2$ and $G_3$, and $G_1$, $G_2$ and $G_3$ are the* child *graphs of $G$. Moreover, $G_1$, $G_2$ are the* direct child *graphs of $G$ and $G3$ is the* indirect child *graph of $G$.*

We select the typical min-cut/max-flow algorithm with $O(n^3)$ time complexity as an example to analyze the performance of the recursive framework. Obviously, the construction of the *indirect child* graph depends on the minimum cuts of the two *direct child* graph of its *parent* graph. We first simply discuss the complexity of the recursive minimum cuts method when this recursive time is one. Let the time of directly computing the minimum cuts $c(S, T)$ of $G = (V, E)$ be $t(|V|)$, the time of computing by the recursive framework is $t_1(|V|)$. The initial cut splits the graph into two parts with the same number of nodes. Then $|S_0| = |T_0| = |V|/2$, we have $t(|V_1|) = t(|V_2|) = t(|V/2|) = t(|V|)/8$. Concerning the time of computing $c_3(S_3, T_3)$ of $G_3 = (V_3, E_3)$, where $V_3 = T_1 + S_2$, $|V_3|$ changes from 0 to $|V|$, then $t(|V_3|)$ changes from 0 to $t(|V|)$. Since the $G = (V, E)$ has arbitrary topology, it is reasonable to presume that $V_3$ obey uniform distribution in $[0, |V|]$, whose mean is $|V|/2$. Thus, the mean time of $t_1(|V|)$ is $t(|V|) \times 3/8$. If we extend to the basic min-cut/max-flow algorithm with $O(n^p)$ time complexity, we can get $t_1(|V|) = t(|V|) \times 3/2^p$. This means only when $p > log3/log2$, we have $t_1(|V|) < t(|V|)$ and the RMC framework can accelerate.

Next, we discuss the complexity of the proposed recursive framework versus the recursion number. As *direct child* graphs can be directly constructed from its *parent* graph and their number of nodes is a half of the number of nodes of their *parent* graph. The *indirect child* graph relies on the *direct child* graphs of its *parent* graph, and the number of nodes of *indirect child* graph changes from 0 to the number of nodes of its *parent* graph, we can't precisely estimate the computational time of *indirect child* graph within the recursive framework. As we previously said, it is reasonable to presume that the number of nodes of *indirect child* graph obey uniform distribution from 0 to the number of nodes of its *parent* graph. Therefore, in the next part, the number of nodes of *indirect child* graph is presumed same as the number of nodes of *direct child* graph, i.e., half of the number of nodes of their *parent* graph. Therefore, the time complexity that we discuss in this paper is the mean complexity of the proposed recursive framework.

Let $N = |V| = 2^n$ be the number of the nodes of $G = (V, E)$. If $n$ is positive integer, the complexity analysis is easier than that $n$ isn't positive integer. We first discuss the case when $n$ is positive integer, based on which we further discuss the case where $n$ is positive real number.

**6.1. $N = |V| = 2^n$, where $n$ is integer.** Let $N = |V| = 2^n$ be the number of the nodes of $G = (V, E)$, where $n$ is integer. The recursive minimum cuts framework is applied to compute the minimum cuts $c(S, T)$ of $G$ and the selected basic min-cut/max-flow algorithm has $O(N^p)$ time complexity. Let the total recursive time be $k$, where $k$ is integer, and the recursive time variable be $i$, where $i$ is integer.

Let the node number of *direct child* graphs and *indirect child* graphs be the half of the node number of its *parent* graph. Therefore, in the $i^{th}$ recursion, the number of the nodes in *direct child* graphs and *indirect child* graphs is $N_i = \frac{N}{2^i}$ and the number of the *child* graphs is $3^i$. Let $t_i(N)$ be the computational time of the recursive framework with recursive time $i$, and especially let $t_0(N)$ be the computational time using the selected basic min-cut algorithm.

LEMMA 6.2. *Let $k$ the total number of recursions. Then we have the following estimate of the mean computational times:*

$$\frac{t_{i+1}(N)}{t_i(N)} = \frac{3}{2^p}, i = 1, 2, \cdots k.$$

*The mean ratio between the computational time of the recursive framework and the time of directly computing the minmun cut by the selected basic min-cut algorithm is:*

$$r_k(N) = \frac{t_k(N)}{t_0(N)} = (\frac{3}{2^p})^k.$$

*Proof.* Since the number of nodes of *child* graphs in the $i^{th}$ recursion is $\frac{N}{2^i}$ and the number of *child* graphs is $3^i$, thus the number of nodes of *child* graphs in the $(i+1)^{th}$ recursion is $\frac{N}{2^{i+1}}$ and the number of *child* graphs is $3^{i+1}$. Moreover, the time complexity $t_0(N)$ is $O(N^p)$. Accordingly, we have

$$\frac{t_{i+1}(N)}{t_i(N)} = \frac{t_0(\frac{N}{2^{i+1}}) \times 3^{i+1}}{t_0(\frac{N}{2^i}) \times 3^i} = \frac{t_0(\frac{N}{2^{i+1}}) \times 3^{i+1}}{t_0(2 \times \frac{N}{2^{i+1}}) \times 3^i} = \frac{3}{2^p}.$$

Therefore,

$$r_k(N) = \frac{t_k(N)}{t_0(N)} = \prod_{i=0}^{k-1} \frac{t_{i+1}(N)}{t_i(N)} = (\frac{3}{2^p})^k.$$

□

From Lemma 6.2, we can get the following theorem.

THEOREM 6.3. *The mean time complexity of the RMC algorithm is $O(N^{\frac{\log 3}{\log 2}})$.*

*Proof.* Since $N = 2^n$, the maximum recursive time is $n$, where every *child* graph includes only one node in addition to the source and sink nodes. When $k = n$, the recursive framework get the best time complexity. Let $k = n$, we have

$$t_n(N) = r_n(N)t(N) = \frac{3^n}{2^{pn}}t(N) = 3^n \times \frac{t(N)}{N^p} = N^{\frac{\log 3}{\log 2}} \times \frac{t(N)}{N^p}.$$

Since the time complexity of $t(N)$ is $O(N^p)$, then the time complexity of $t_n(N)$ is $O(N^{\frac{\log 3}{\log 2}})$. □

Note that the above analysis is approximate because we ignore the influence of the source and sink nodes. By the construction method of the subgraphs in Section 2, if the number of nodes of *parent* graph is $N$ which is even, the numbers of nodes of the corresponding two direct *child* graphs are not $N/2$, but $N/2+1$ because every produced *child* graph must be added a source node or sink node in addition to the inherited nodes from its *parent* graph.

Assume that the graph $G$ has $2^n$ nodes in addition to the source and sink nodes, where $n$ is an integer, then $N = 2^n + 2$. If we decompose the *parent* graph without considering the source and sink nodes, in the $i^{th}$ recursion the produced *child* graphs have $2^{n-i} + 2$ nodes. Then we have

$$\frac{t_{i+1}(N)}{t_i(N)} = \frac{t_0(2^{n-i-1}+2)\times 3}{t_0(2^{n-i}+2)} = 3(\frac{2^{n-i-1}+2}{2^{n-i}+2})^p,$$

$$r_n(N) = \frac{t_n(N)}{t_0(N)} = \prod_{i=0}^{n-1}\frac{t_{i+1}(N)}{t_i(N)} = 3^n(\frac{2^{n-n}+2}{N})^p = 3^p \times \frac{3^n}{N^p}.$$

Thus, the time complexity of $t_n(N)$ is $O(3^p N^{\frac{\log 3}{\log 2}})$. Since $3^p$ is a constant that isn't related to the Graph $G$, the time complexity of the RMC algorithm can still be approximately considered as $O(N^{\frac{\log 3}{\log 2}})$. As ignoring the source and sink nodes has no evident effect on the time complexity of the RMC framework, for the simplicity, we will continue to adopt the approximation in the following analysis.

**6.2. $N = |V| = 2^n$, where $n$ is positive real value.** It should be noticed that, for the above complexity analysis, the number of nodes of $G = (V, E)$ is $N = |V| = 2^n$, where $n$ is integer. But in most applications, if $N$ is denoted as $2^n$, $n$ may not be an integer. We will extend the result to this case now.

If $N = 2^n$, where $n$ is positive real number, then the "number" of nodes of each *direct child* graph $\frac{N}{2^i}$ in the $i^{th}$ recursion may not be positive integer ($i$ is an integer), which has no practical sense. Assume that $\frac{N}{2^j}$ is a positive integer and $\frac{N}{2^{j+1}}$ isn't a positive integer, where $0 \le j < j+1 \le n$ and $j$ is an integer. Therefore, Lemma 5.2 is still valid when the total recursive time $k \le j$. We analyze the case when $j < k \le n$ in the following.

Let $M$ be the number of nodes of $G^m$. If $M$ is even, then we let $M = 2m$ and $m$ is a positive integer. Then the corresponding node number of two *direct child* graphs is both the half of the node number of its *parent* graph, i.e., $m$, and the corresponding node number of one *indirect child* graph is set to its mean $m$. If $M$ is odd, then we let $M = 2m + 1$ for a given positive integer $m$. The corresponding node number of two *direct child* graphs is respectively $m + 1$ and $m$. The corresponding mean node number of *indirect child* graphs will be $m + 1$ or $m$. The following Lemma 5.4 tells us, if $N = 2^n$ for a positive real $n$, then the difference between the maximum node number and the minimum node number of the child graphs in the $i^th$ recursion is one.

LEMMA 6.4. *Assume $N = 2^n$ with a positive real $n$. Let the* child *graphs in the $i^{th}$ recursion be $G_i^m, (1 \le m \le 3^i, 0 \le i \le n)$, where $m$, $i$ are integers. Let the number of nodes in $G_i^m$ is $N_i^m$. Assume that $\frac{N}{2^j}$ is an integer and $\frac{N}{2^{j+1}}$ isn't an integer, where $0 \le j < j+1 \le n$, $j$ is an integer. Let $N_i^{\max}$ and $N_i^{\min}$ be respectively the maximum*

*and the minimum of* $N_i^m, (1 \le m \le 3^i, 0 \le i \le n)$. *Then, we have*

$$N_i^{\max} = N_i^{\min} = \frac{N}{2^i}, \quad 0 \le i \le j.$$

$$N_i^{\max} = [\frac{N}{2^i}] + 1, \quad dj < i \le n$$

$$N_i^{\min} = [\frac{N}{2^i}],$$

*where* $[\frac{N}{2^i}]$ *is the largest integer that is no more than* $\frac{N}{2^i}$.

*Proof.* When $0 \le i \le j$, by the above analysis, $N_i^{\max} = N_i^{\min} = \frac{N}{2^i}$ holds.We shall prove when $j < i \le n$, $N_i^{\max} = [\frac{N}{2^i}] + 1$ and $N_i^{\min} = [\frac{N}{2^i}]$ hold.

The proof is by induction. When $i = j+1$, because $\frac{N}{2^j}$ is an integer and $\frac{N}{2^{j+1}}$ isn't an integer, $\frac{N}{2^j}$ is odd. Let $\frac{N}{2^j} = 2M + 1$, where $M$ is an integer, then $\frac{N}{2^{j+1}} = M + \frac{1}{2}$. Therefore,we have $N_1^{\min} = M = [\frac{N}{2^{j+1}}]$ and $N_{j+1}^{\max} = M + 1 = [\frac{N}{2^{j+1}}] + 1$ hold.

In the induction step, the induction hypothesis is that when $i = j+k$ and $j+1 < i \le n-1$, $N_{j+k}^{\max} = [\frac{N}{2^{j+k}}]+1$ and $N_{j+k}^{\min} = [\frac{N}{2^{j+k}}]$ hold. We must prove when $i = j+k+1$ and $j+1 < i \le n$, $N_{j+k+1}^{\max} = [\frac{N}{2^{j+k+1}}] + 1$ and $N_{j+k+1}^{\min} = [\frac{N}{2^{j+k+1}}]$ hold.

Let $N_{j+k}^{\max} = [\frac{N}{2^{j+k}}] + 1 = M + 1$ and $N_{j+k}^{\min} = [\frac{N}{2^{j+k}}] = M$. Then every $N_{j+k}^m = M$ *or* $M+1$ $(1 \le m \le 3^{j+k})$. Since it is impossible that $M$ and $M+1$ are even or odd at the same time, we assume $M = 2K$ and $M + 1 = 2K + 1$. For every $G_{j+k}^m$ with $N_{j+k}^m = 2K$ nodes, we can produce three corresponding *child* graphs $G_{j+k+1}^{ml}(1 \le l \le 3)$ with $N_{j+k+1}^{ml} = K(1 \le l \le 3)$ nodes. For every $G_{j+k}^m$ with $N_{j+k}^m = 2K + 1$ nodes, we can produce three corresponding *child* graphs $G_{j+k+1}^{ml}(1 \le l \le 3)$ with $N_{j+k+1}^{ml} = K$ *or* $K + 1$ $(1 \le l \le 3)$, nodes. Thus, we have $N_{j+k+1}^{ml} = K$ *or* $K + 1$ $(1 \le l \le 3, 1 \le m \le 3^{j+k+1})$. Then, $N_{j+k+1}^{\max} = K + 1$ and $N_{j+k+1}^{\min} = K$. Thus, $N_{j+k+1}^{\max} = K + 1 = \frac{M}{2} + 1 = \frac{[\frac{N}{2^{j+k}}]}{2} + 1 = [\frac{N}{2^{j+k+1}}] + 1$ and $N_{j+k+1}^{\min} = K = \frac{M}{2} = \frac{[\frac{N}{2^{j+k}}]}{2} = [\frac{N}{2^{j+k+1}}]$. The induction step holds.

□

Corollary 6.5. *Assume that* $N = 2^n$, *where* $n$ *is positive real number. The mean time complexity using the recursive minimum cuts framework is still* $O(N^{\frac{\log 3}{\log 2}})$.

*Proof.* Let $k - 1 < n \le k$, then $2^{k-1} < 2^n \le 2^k$. When $n = k$, by Corollary 5.3, The mean time complexity is $O(N^{\frac{\log 3}{\log 2}})$. When $k - 1 < n < k$, $2^{k-1} < N < 2^k$, then we have $t_0(2^{k-1}) < t_0(N) < t_0(2^k)$, $\frac{t_0(N)}{t_0(2^{k-1})} = \frac{N^p}{2^{(k-1)p}}$, and $\frac{t_0(N)}{t_0(2^k)} = \frac{N^p}{2^{kp}}$ hold.

Let $G_{k-1}$, $G_k$ and $G_n$ be graphs that respectively include $2^{k-1}$ nodes, $2^k$ nodes and $N$ nodes. For $G_n$, the maximum recursive time is $k - 1$. When the recursive time is $k - 1$, the produced *child* graphs of $G_{k-1}$, $G_k$ and $G_n$ will be the same, $3^{k-1}$, every *child* graph of $G_{k-1}$ includes one node in addition to the source and sink nodes, every *child* graph of $G_k$ includes two nodes in addition to the source and sink nodes, but for $G_n$, every *child* graph include one or two nodes in addition to the source and sink nodes by Lemma 5.4. Therefore, we have $t_{k-1}(2^{k-1}) < t_{k-1}(N) < t_{k-1}(2^k)$, $t_{k-1}(2^{k-1}) = \frac{3^{k-1}}{2^{(k-1)p}} t_0(2^{k-1})$, and $t_{k-1}(2^k) = \frac{3^{k-1}}{2^{(k-1)p}} t_0(2^k)$. Then by Theorem 5.4 we have

$$t_{k-1}(N) < t_{k-1}(2^k) = \frac{3^{k-1}}{2^{(k-1)p}} t_0(2^k) = 2^p \times 3^{k-1} \times \frac{t_0(2^k)}{2^{kp}} = 2^p \times 3^{k-1} \times \frac{t_0(N)}{N^p}.$$

$$t_{k-1}(N) > t_{k-1}(2^{k-1}) = \frac{3^{k-1}}{2^{(k-1)p}} t(2^{k-1}) = 3^{k-1} \times \frac{t(2^{k-1})}{2^{(k-1)p}} = 3^{k-1} \times \frac{t_0(N)}{N^p}.$$

As $k-1 < n < k$ we have $\frac{\log N}{\log 2} - 1 < k-1 < \frac{\log N}{\log 2}$, thus

$$\frac{1}{3} N^{\frac{\log 3}{\log 2}} \frac{t_0(N)}{N^p} < t_{k-1}(N) < 2^p N^{\frac{\log 3}{\log 2}} \frac{t_0(N)}{N^p}.$$

Since the time complexity of $t_0(N)$ is $O(N^p)$, then the time complexity of $t_{k-1}(N)$ is $O(N^{\frac{\log 3}{\log 2}})$. □

**7. Parallel Implementation and Analysis.** Based on our graph decomposition method in section 3 and minimum cut composition theorems in section 4, we can design a parallel algorithm to solve the min-cut problem.

We first give the definition of *triple tree*. The *triple tree* is recursively defined similar to binary tree. A *triple tree* $T$ is a structure defined on a finite set of nodes that either 1) contains no nodes, or 2)is composed of four disjoint sets of nodes: a root node, a left subtree, a right subtree, and a middle subtree. Similar to the *complete binary tree*, we can define the *complete triple tree*. A *complete triple tree* is a *triple tree* in which all leaves have the same depth and all the internal nodes have the same degree 3. Thus, a *triple tree* is a *k-ary tree* with $k = 3$ and a *complete triple tree* is a *complete k-ary tree* with $k = 3$ [27].

Considering the original graph as *root* node and all the *child* graphs as *child* nodes, and connecting the *parent* graph with its *child* graphs using edges, we can get a *complete triple tree* $T$. Notice that every node in $T$ corresponds to a graph. Fig. 7.1 shows a *complete triple tree* $T$ which indicates the original graph is recursively decomposed to 3 layers.

LEMMA 7.1. *If the original graph is recursively decomposed to k layers* ($k > 0$)*, the depth of every leaf in the corresponding complete triple tree T is k and the number of the leaves and the paths from the root to the leaves are both* $3^k$.

Every node except for the *root* in the *complete triple tree* $T$ has an *attribute* value to indicate that the node corresponds to a *direct child* graph or *indirect child* graph. We define the *attribute* of node is 0 if the node corresponds to a *direct child* graph and 1 if the node corresponds to an *indirect child* graph. Especially, the attribute of *root* is 0.

In the *complete triple tree* $T$ we can find a *path* from the *root* node to any node $v$. The *path* to node $v$ is also called the *path* of node $v$.

We define the *attribute* of *path* is the binary sequence of the *attributes* of the nodes that are included in the path, where the higher bit is the *attribute* of *root*. Let $x_q \in \{0, 1\}, q = 0, \cdots, k$ be a binary variable. Then the *attribute* of the *path* of node $v$ can be written as $x_0, x_1 \cdots x_k$ , where $x_0$ is the *attribute* of *root* and $x_k$ is the attribute of node $v$. Since the *attribute* of *root* is always 0 we can ignore the *attribute* of *root* in the *attribute* of *path* and the *attribute* of the *path* of node $v$ can be written as $x_1, x_2 \cdots x_k$. We define the *label* of node $v$ as the *attribute* of its *path*.

We decompose the original graph into many subgraphs so that the subgraphs can be computed in parallel. Fig. 7.1 shows the case where the original graph is recursively decomposed for three layers and the *complete triple tree* $T$ is used to describe the decomposition, where the red nodes with *attribute* 0 correspond to the *direct child* graphs and the blue nodes with *attribute* 1 correspond to the *indirect child*

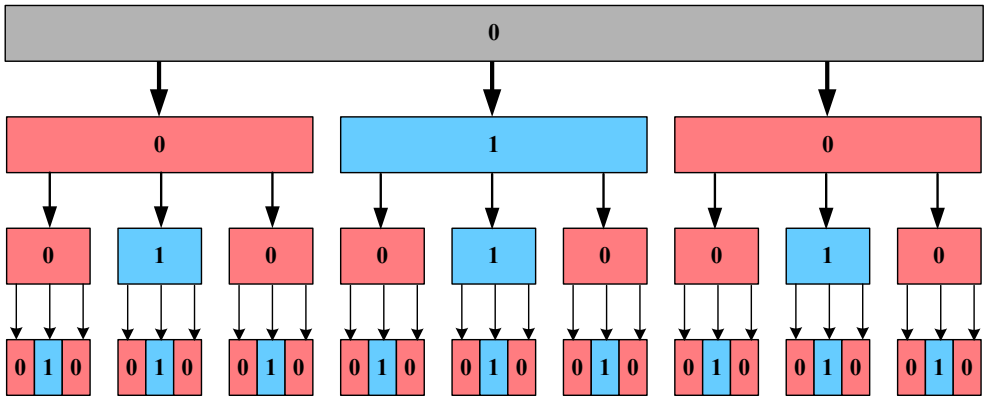


FIG. 7.1. *the parallel algorithm illustration.*

graphs. For example, the *attribute* of the *path* from the *root* to the $5^{th}$ *leaf* from left is $x_1x_2x_3 = 011$.

All the leaves in $T$ correspond to the subgraphs whose minimum cuts will be computed to compose the minimum cuts of the original graph. However, it should be noticed that not all the corresponding subgraphs of the leaves can be computed in parallel. Some subgraphs should be constructed depending on the other subgraphs because the construction of the *indirect child* graph depends on the minimum cuts of the *direct child* graphs of its *parent* graph. Therefore, in our parallel algorithm, we must decide the subgraphs that can be computed in parallel and the sequence of the computation.

THEOREM 7.2. *If the original graph is recursively decomposed to k layers* $(k > 0)$*, in the corresponding complete triple tree T all the leaves can be classified into* $2^k$ *categories according to their labels and each category includes* $2^i$ *leaves with the same label, where* $i = \sum_{q=1}^{k} (1 - x_q)$ *and* $x_1x_2 \cdots x_k$ *is the attributes of the paths.*

*Proof.* The proof is by induction method. We first prove the lemma holds when $k = 1$, which is the induction basis. From Fig. 7.1, we can see that when $k = 1$ the number of the leaves is 3 and are classified into 2 categories. The number of the leaves with label 0 is 2 and the number of the leaf with label 1 is 1. Then the induction basis holds.

In the induction step, by the induction hypothesis, when $k = j$, the *labels* of the *leaves*, i.e., $x_1x_2 \cdots x_j$ , has $2^j$ values, and for a given label $x_1x_2 \cdots x_j$, we can find $2^i$ leaves, where $i = \sum_{q=1}^{k} (1 - x_q)$.

When $k = j+1$, the *labels* of the leaves is $x_1x_2 \cdots x_jx_{j+1}$. Every *child* graph in the $j^{th}$ layer will be decomposed to two *direct child* graphs and one *indirect child* graph in the $(j+1)^{th}$ layer. The attribute $x_{j+1}$ of the corresponding nodes of the *direct child* graphs is 0 and the *attribute* $x_{j+1}$ of the corresponding node of the *indirect child* graph is 1. Therefore, $x_1x_2 \cdots x_jx_{j+1}$ has $2^{j+1}$ values, and for a given label $x_1x_2 \cdots x_jx_{j+1}$, we can find $2^i$ leaves, where $i = \sum_{q=1}^{k} (1 - x_q) + 1 - x_{k+1} = \sum_{q=1}^{k+1} (1 - x_q)$ hold. The Theorem is proved.

□

THEOREM 7.3. *If the original graph is recursively decomposed to k layers* $(k > 0)$*, in the corresponding complete triple tree T, the corresponding subgraphs of any two leaves with the same label can be constructed independently and their minimum cuts can be computed in parallel.*

*Proof.* The proof is by induction method. When $k = 1$, the induction basis holds.

In the induction step, the induction hypothesis is that when $k = j$, the corresponding subgraphs of any two *leaves* with the same *label* can be constructed independently.

When $k = j + 1$, if the corresponding subgraphs of two *leaves* with the same *label* have the same *parent* graph, the two subgraphs must be the two *direct child* graphs, and then they just depend on their *parent* graph and can be constructed independently and their minimum cuts can be computed in parallel. If the corresponding subgraphs of two *leaves* with the same *label* have the different *parent* graphs, they can be constructed independently and their minimum cuts can be computed in parallel since their *parent* graphs are constructed independently by induction hypothesis. The induction step holds.

□

THEOREM 7.4. *If the original graph is recursively decomposed to k layers* $(k > 0)$*, in the corresponding* complete triple tree T*, the corresponding subgraphs of any two* leaves *with the different* label *can not be constructed independently and the construction of the one subgraph must depend on the minimum cut of the other subgraph. Then their minimum cuts must be computed sequentially.*

*Proof.* The proof is by induction method. When $k$=1, the induction basis holds.

In the induction step, the induction hypothesis is that when $k = j$, the corresponding subgraphs of any two *leaves* with the different *label* can not be constructed independently and the construction of the one subgraph must depend on the minimum cut of the other subgraph.

When $k = j + 1$, if the *parent* of two *leaves* with the different *label* have different label, the corresponding subgraphs of any two *leaves* can not be constructed independently by induction hypothesis. If their *parents* have same label, obviously the *attributes* of the two leaves are different and one is 1 and the other is 0. The leaf with *attribute* 1 is constructed depending on its *parent* but the other isn't. Therefore, the two leaves can't be constructed independently although their *parents* can be constructed independently. The induction step holds.

By Theorem 7.2,7.3 and 7.4, when the original graph is recursively decomposed for $k$ layers $(k > 0)$, the subgraphs that can be computed in parallel in one time can be decided. Since in all the $3^k$ subgraphs, only the ones with the same *labels* can be computed in parallel in one time and there are $2^k$ different labels for all the subgraphs. Therefore we need to do $2^k$ parallel computation and the number of the needed processor in one parallel is $2^i$ , where $i = \sum_{q=1}^{k} (1 - x_q)$ and $x_1 x_2 \cdots x_k$ is the *label* of the currently processed leaves.

□

THEOREM 7.5. *If the minimum cuts of the corresponding subgraphs of the leaves in T with the same label are computed in parallel and the corresponding subgraphs of the leaves in T with the different label are computed sequentially, then the sequence of the computation will be from the leaves with small label value to the leaves with large*

*label value.*

*Proof.* The theorem is equivalent to that given any two *leaves*, the corresponding subgraph of the *leaf* with smaller *label* must be computed before the subgraph of the *leaf* with larger *label*.

The proof is by induction method. When $k = 1$, the induction basis holds.

In the induction step, the induction hypothesis is that when $k = j$, the sequence of the computation is from the *leaves* with small *label* to the *leaves* with large *label*.

When $k = j + 1$, if the corresponding subgraphs of two *leaves* with different *label* $x_1x_2\cdots x_jx_{j+1}$ have the same *parent* graph with *label* $x_1x_2\cdots x_j = a_1a_2\cdots a_j$, the subgraphs are respectively one *direct child* graphs and one *indirect child* graph of this *parent* graph. Obviously the *direct child* graphs with *label* $x_1x_2\cdots x_jx_{j+1} = a_1\cdots a_k0$ is computed before the *indirect child* graph with *label* $x_1x_2\cdots x_jx_{j+1} = a_1\cdots a_k1$ is computed.

If the corresponding subgraphs of two *leaves* with different *label* $x_1x_2\cdots x_jx_{j+1}$ have the different *parent* graphs. Let the *labels* $x_1x_2\cdots x_j$ of their *parent* graphs are respectively $a_1a_2\cdots a_j$ and $b_1b_2\cdots b_j$, and the *labels* $x_1x_2\cdots x_jx_{j+1}$ of the two leaves are respectively $a_1a_2\cdots a_jc$ and $b_1b_2\cdots b_jd$. Let us presume $a_1a_2\cdots a_j < b_1b_2\cdots b_j$, then $a_1a_2\cdots a_jc < b_1b_2\cdots b_jd$. By the induction hypothesis the *parent* graph with *label* $a_1a_2\cdots a_j$ is computed before the *parent* graph with *label* $b_1b_2\cdots b_j$. Therefore, the subgraph with *label* $a_1a_2\cdots a_jc$ is computed before the subgraph with *label* $b_1b_2\cdots b_jd$, which is consistent with $a_1a_2\cdots a_jc < b_1b_2\cdots b_jd$. Thus the induction step holds. □

Theorem 6.5 gives the sequence of the parallel computation for the corresponding subgraphs of all the leaves in $T$. For example, in Fig. 7.1, the sequence of the parallel computation for the corresponding subgraphs of all the *leaves* in $T$ is as: $000 \to 001 \to 010 \to 011 \to 100 \to 101 \to 110 \to 111$.

By this theorem we can design our parallel algorithm using at most $N$ processor, where $N$ is the number of the nodes in the original graph.

We now analyze the time complexity of the proposed parallel algorithm. For simplicity, we just analyze the complexity when $N = 2^n$, where n is positive integer. We can get the approximate conclusion when $n$ is positive real number like section 5 does.

LEMMA 7.6. *The original graph is recursively decomposed to k layers* ($k > 0$) *and all the corresponding subgraphs of the* leaves *with the same* label *are computed in parallel. If the minimum cuts of every subgraph is computed by the selected basic min-cuts algorithm with* $O(N^p)$ *time, the mean ratio between the computational time of the parallel algorithm and directly computing the original graph by the selected basic min-cut algorithm is* $s_k(N) = (\frac{1}{2^{p-1}})^k$.

*Proof.* The computational time of every subgraph that the leaf in $T$ corresponds to is $(\frac{1}{2^p})^k t_0(N)$, and the total times is $2^k$ (Theorem 6.2). Therefore, $s_k(N) = (\frac{1}{2^p})^k 2^k = (\frac{1}{2^{p-1}})^k$.
□

COROLLARY 7.7. *If the minimum cuts of every subgraph is computed by the RMC algorithm, the mean ratio between the computational time of the parallel algorithm and directly computing the original graph by the selected basic min-cut algorithm with* $O(N^p)$ *is* $s_k(N) = (\frac{2}{3})^k(\frac{3}{2^p})^n$.

FIG. 8.1. *Left: Cameraman image; Right: Lena image*

*Proof.* If the minimum cut of every subgraph is computed by the RMC algorithm, we have $s_k(N) = (\frac{1}{2^{p-1}})^k(\frac{3}{2^p})^{n-k} = (\frac{2}{3})^k(\frac{3}{2^p})^n$.
□

THEOREM 7.8. *The time complexity of the proposed parallel algorithm is* $O(N)$.

*Proof.* Since $s_k(N) = (\frac{2}{3})^k(\frac{3}{2^p})^n$ monotonously decreases as parameter $k$, $s_k(N)$ get the smallest value when $k = n$. Therefore, $s_n(N) = (\frac{2}{3})^n(\frac{3}{2^p})^n = \frac{1}{2^{n(p-1)}} = \frac{1}{N^{p-1}}$. Then the computational time of the parallel algorithm is $t_n(N) = s_n(N)t_0(N) = \frac{t_0(N)}{N^{p-1}}$ and the time complexity is $O(N)$ because $t(N)$ has $O(N^p)$ time complexity. □

**8. Experiment.** Our main contribution of this work is to: design a graph decomposition method, prove min-cut composition theorems, and analyze the computational complexity of the constructed algorithms. Thorough experiments validation for the RMC framework and its parallelization is a difficult task since our proposed framework and algorithm is suitable to graphs with any topological construction and any minimum cuts algorithms, including "augmenting paths" style algorithms [13] and "push-relabel" style algorithms [15]. Therefore, we should prepare and design all kinds of graph data sets of different topological construction to test the proposed methods, which is extremely difficult since there aren't available graph data sets for open usage.

Another reason is that it is rather difficult to test the computational complexity of one algorithm because the complexity of the algorithm is a theoretical worst bound. We choose the push-relabel algorithm with $O(N^3)$ time complexity as an example to analyze.Let the computational time of the original graph be $t$. When the number of the nodes in the graph decreases to the half of the original, the theoretical computational time will decrease to $t/8$. This is just a theoretical bound. However, in most cases the computational time of the decreased graph will change between $t$ and $t/8$, depending on the number of the edges and their capacities in the graph. Therefore, it is possible that one graph with less nodes may cost more computational time than the other graph with more nodes. This leads to the difficulty of the accurate experiment evaluation. Therefore, we plan to systematically carry out and report the experiment evaluation in another work. In this paper we will just conduct some preliminary experiment to demonstrate the performance of the proposed RMC framework with the push-relabel

algorithm. The parallel implementation isn't tested in this paper.

We construct graphs from some images to test the RMC framework described in section 4. We have proved that the RMC algorithm can obtain the same solution as the original push-relabel-type algorithms or argument path-type algorithms in solving the min-cut problem, not an approximate solution. Our experimental results validate this conclusion.

In the following, we give two flow network graphs constructed from two images "Cameraman" and "Lena" with size $800 \times 800$ which are shown in Fig. 8.1 to test the performance of the RMC algorithm. Given an image, each pixel corresponds to a node of graph. There are edges connecting the neighboring pixel nodes, called $n$-links. There are two specially designated terminal nodes source node and sink node. There are edges connecting source node and sink node with each pixel node, called $t$-links. We also set gray values to the source node and sink node respectively. Therefore, all the $n$-links and $t$-links are computed as $e^{-\beta|G(p)-G(q)|}$ according to the gray values of the neighboring two nodes, where $G(p)$ and $G(q)$ are the gray values of the neighboring two nodes $p$ and $q$, $\beta$ is a fixed constant. Minimizing the cut of the constructed flow network graph we can get a segmentation of the image. Changing the gray values of source node and sink node and the value of parameter $\beta$, we can get a new graph and new segmentation. Therefore, with the constructed graph we can test the efficiency of the RMC algorithm.

The computing time for the "Cameraman" image and "Lena" image are respectively shown in Table 8.1 and 8.2, including the original push-relabel algorithm, RMC algorithm with recursive time 1 and 2. In Table 8.1, when the recursive time is 1 we can see that the number of the nodes and edges in $G_1$ and $G_2$ are the same, the computational time of $G_1$ is much more than $G_2$. This reveals the instability of the push-relabel algorithm, extremely depending on the capacities of the graph.

In Table 8.1 and 8.2, $G_{ij}$ ($i$, $j$ =1, 2, 3) indicates the $j^{th}$ subgraph of the $i^{th}$ subgraph of the original graph. When the recursive time is 2, from Table 1 and 2, we can see that the number of the nodes of $G_{ij}$ ($i$, $j$=1, 2) is the same, 160000, but in $G_{ij}$ ($i = 3$ or $j = 3$), the number of the nodes is changed, sometime more than 160000, sometime less than 160000. From Table 8.1 and 8.2, we can see that the total time gradually decreases as the recursion number increases.

**9. Conclusion.** We have proposed one general graph decomposition and minimum cuts composition framework. One advantage of the approach is that any minimum cuts algorithm can be used for the subgraphs. We first present a graph decomposition method to split the original graph into three subgrpahs with less nodes, and prove the minimum cuts of the original graph can be composed of those of its three subgraphs. Based on the graph decomposition method and minimum cuts composition theorems, we design a recursive minimum cuts algorithm and prove the algorithm runs with a time complexity $O(N^{\frac{\log 3}{\log 2}})$. The second approach is a parallel algorithm that run with time complexity $O(N)$. We have presented preliminary experiment to verify the RMC algorithm.

REFERENCES

[1] S. ASinha, P. Mordohai, and M. Pollefeys. Multi-view stereo via graph cuts on the dual of an adaptive tetrahedral meshn. *ICCV*, 2007.

[2] E. BAE, J. YUAN, XC TAI, and Y. BOYKOV. A fast continuous max-flow approach to non-convex multilabeling problems. Technical report, Technical report CAM-10-62, UCLA, 2010.

TABLE 8.1
*The computational time of the RMC algorithm with recursive time for Camera image*

| **The recursive time is 0 (The original push-relabel algorithm)** | | | |
|---|---|---|---|
| | Comp. Time(s) | The number of the nodes | The numberof the edges |
| $G$ | 21.088944 | 640000 | 5110404 |
| **The recursive time is 1** | | | |
| | Comp. Time (s) | The number of the nodes | The number of the edges |
| $G_1$ | 13.961657 | 320000 | 2517760 |
| $G_2$ | 1.325443 | 320000 | 2516476 |
| $G_3$ | 1.410852 | 116891 | 891902 |
| *Total* | 16.697953 | | |
| **The recursive time is 2** | | | |
| | Comp. Time (s) | The number of the nodes | The number of the edges |
| $G_{11}$ | 1.963990 | 160000 | 1234152 |
| $G_{12}$ | 2.740732 | 160000 | 1197942 |
| $G_{13}$ | 6.088850 | 242016 | 1896188 |
| $G_{21}$ | 0.888230 | 160000 | 1237080 |
| $G_{22}$ | 0.343907 | 160000 | 1269472 |
| $G_{23}$ | 0.357132 | 144774 | 1125782 |
| $G_{31}$ | 0.272420 | 58446 | 407060 |
| $G_{32}$ | 0.828222 | 58445 | 403582 |
| $G_{33}$ | 1.095820 | 96881 | 733524 |
| *Total* | 14.579302 | | |

TABLE 8.2
*The computational time of the RMC algorithm with recursive time for the "Lena" image*

| **The recursive time is 0 (The original push-relabel algorithm)** | | | |
|---|---|---|---|
| | Comp. Time(s) | The number of the nodes | The numberof the edges |
| $G$ | 9.848429 | 640000 | 5110404 |
| **The recursive time is 1** | | | |
| | Comp. Time (s) | The number of the nodes | The number of the edges |
| $G_1$ | 4.107710 | 320000 | 2552804 |
| $G_2$ | 1.605400 | 320000 | 2552804 |
| $G_3$ | 2.132148 | 267098 | 2101068 |
| *Total* | 7.845259 | | |
| **The recursive time is 2** | | | |
| | Comp. Time (s) | The number of the nodes | The number of the edges |
| $G_{11}$ | 0.445473 | 160000 | 1274004 |
| $G_{12}$ | 1.424457 | 160000 | 1275676 |
| $G_{13}$ | 0.751997 | 154967 | 1216324 |
| $G_{21}$ | 0.628157 | 160000 | 1276754 |
| $G_{22}$ | 0.815949 | 160000 | 1274004 |
| $G_{23}$ | 0.403161 | 181827 | 1437616 |
| $G_{31}$ | 0.495870 | 133549 | 1007548 |
| $G_{32}$ | 1.286056 | 133549 | 1020694 |
| $G_{33}$ | 0.624665 | 79983 | 593264 |
| *Total* | 6.875785 | | |

[3] Y. Boykov and D. Huttenlocher. A new bayesian framework for object recognition. *CVPR*, 1999.
[4] Y. Boykov and V. Kolmogorov. Computing geodesics and minimal surfaces via graph cuts. *ICCV*, 2003.
[5] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(9):1124–1137, 2004.
[6] Y. Boykov, O. Veksler, and R. Zabih. Fast approaximate energy minimization via graph cuts. *ICCV*, 1999.
[7] Y.Y. Boykov and M.P. Jolly. Interactive graph cuts for optimal boundary & region segmentation of objects in nd images. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, volume 1, pages 105–112. IEEE, 2001.
[8] B. V. Cherkassky and A. V. Goldberg. On implementing the push-relabel method for the maximum flow problem. *Algorithmica*, 19(4):390–410, 1997.
[9] J. Darbon. Global optimization for first order markov random fields with submodular priors. *Discrete Applied Mathematics*, 157(16):3412–3423, 2009.
[10] J. Darbon and M. Sigelle. Image restoration with discrete constrained total variation part i: Fast and exact optimization. *Journal of Mathematical Imaging and Vision*, 26(3):261–276, 2006.
[11] J. Darbon and M. Sigelle. Image restoration with discrete constrained total variation part ii: Levelable functions, convex priors and non-convex cases. *Journal of Mathematical Imaging and Vision*, 26(3):277–291, 2006.
[12] E.A. Dinic. Algorithm for solution of a problem of maximum flow in networks with power estimation. *Soviet Math. Dokl*, 14(11):1277–1280, 1970.
[13] L. Ford and D. Fulkerson. Flows in networks. *Princeton Univ. Press*, 1962.
[14] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum flow problem. *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, 1986.
[15] A.V. Goldberg and R.E. Tarja. A new approach to the maximum-flow proble. *J. ACM*, 35(4):921–940, 1988.
[16] D. Greig, B. Porteous, and A. Seheult. Exact maximum a posteriori estimation for binary images. *Journal of the Royal Statistical Society*, 51(2):271–279, 1989.
[17] O. Juan. On some extensions of level sets and graph cuts towards their applications to image and video segmentation. *PhD thesis*, 2006.
[18] O. Juan and Y. Boykov. Active graph cuts. *CVPR*, 2006.
[19] O. Juan and Y. Boykov. Capacity scaling for graph cuts in visio. *CVPR*, 2007.
[20] P. Kohli and P.H.S. Torr. Dynamic graph cuts for efficient inference in markov random fields. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(12):2079–2088, 2007.
[21] M.P. Kumar, P.H.S. Torr, and A. Zisserman. Obj cut. *CVPR*, 2007.
[22] V. Kwatra, A. Schodl, and I. Essa. Graphcut textures: Image and video synthesis using graph cuts. *SIGGRAPH*, 2003.
[23] Y. Li, J. Sun, and H. Y. Shum. Video object cut and paste. *SIGGRAPH*, 2005.
[24] Y. Li, J. Sun, and C. K. Tang. Lazy snapping. *SIGGRAPH*, 2004.
[25] J. Liu, J. Sun, and Blake A. Parallel graph-cuts by adaptive bottom-up merging. *CVPR*, 2010.
[26] C. Rother, V. Kolmogorov, and Blake A. 'grabcut': Interactive foreground extraction using iterated graph cuts. *ACM Trans. Graphic*, 23(3):309–314, 2004.
[27] R.L. Rivest T.H. Cormen, C.E. Leiserson and C. Stein. Introduction to algorithms. *The MIT Press*, 2001.
[28] G. Vogiatzis, P.H.S. Torr, and R. Cippola. Multi-view stereo via volumetric graph-cuts. *CVPR*, 2005.
[29] J. Yuan, E. Bae, and X.C. Tai. A study on continuous max-flow and min-cut approaches. *CVPR2010, IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2217–2224, 2010.
[30] J. Yuan, E. Bae, X.C. Tai, and Y. Boykov. A continuous max-flow approach to potts model. *Computer Vision–ECCV 2010*, pages 379–392, 2010.