Int J Comput Vis (2010) 89: 40–55
DOI 10.1007/s11263-010-0321-2

Dinkelbach NCUT: An Efficient Framework for Solving Normalized Cuts Problems with Priors and Convex Constraints

Bernard Ghanem · Narendra Ahuja

Received: 16 February 2009 / Accepted: 1 February 2010 / Published online: 25 February 2010

Abstract In this paper, we propose a novel framework, called Dinkelbach NCUT (DNCUT), which efficiently solves the normalized graph cut (NCUT) problem under general, convex constraints, as well as, under given priors on the nodes of the graph. Current NCUT methods use generalized eigen-decomposition, which poses computational issues especially for large graphs, and can only handle linear equality constraints. By using an augmented graph and the iterative Dinkelbach method for fractional programming (FP), we formulate the DNCUT framework to efficiently solve the NCUT problem under general convex constraints and given data priors. In this framework, the initial problem is converted into a sequence of simpler sub-problems (i.e. convex, quadratic programs (QP's) subject to convex constraints). The complexity of finding a global solution for each sub-problem depends on the complexity of the constraints, the convexity of the cost function, and the chosen initialization. However, we derive an initialization, which guarantees that each sub-problem is a convex QP that can be solved by available convex programming techniques. We apply this framework to the special case of linear constraints, where the solution is obtained by solving a sequence of sparse linear systems using the conjugate gradient method. We validate DNCUT by performing binary segmentation on real images both with and without linear/nonlinear constraints, as well as, multi-class segmentation. When possible, we compare DNCUT to other NCUT methods, in terms of segmentation performance and computational efficiency. Even though the new formulation is applied to the problem of spectral graph-based, low-level image segmentation, it can be directly applied to other applications (e.g. clustering).

Keywords Normalized graph cuts · Graph theory · Graph cuts · Interactive segmentation · Graph-based image segmentation · Dinkelbach method for fractional programming · Quadratic programming · Conjugate gradient method · Eigen-decomposition · Multi-way segmentation

B. Ghanem (✉) · N. Ahuja
Beckman Institute for Advanced Science and Technology,
Department of Electrical and Computer Engineering,
University of Illinois at Urbana-Champaign, Urbana, IL 61801,
USA
e-mail: bghanem2@vision.ai.uiuc.edu

N. Ahuja
e-mail: ahuja@vision.ai.uiuc.edu

1 Introduction

This paper is about extending normalized graph cuts so the result satisfies general, convex constraints under given priors on the graph. The new formulation is applied to and presented in terms of low level image segmentation using spectral graph theory, a problem that has received extensive attention recently. However, this framework can be applied to general data clustering problems, where the aforementioned segmentation problem is a special case. In this problem, an image is represented by an undirected graph, wherein nodes correspond to image pixels or regions, and edges connect pairs of nodes. An edge has a weight proportional to the similarity of the properties of the connected nodes (e.g. pixel intensities). Given such a graph representation, image segmentation becomes equivalent to partitioning the nodes of the graph into disjoint sets, or segments, which optimize a given cost function. Such a partition is denoted as a graph cut. Traditionally, the cost of a cut between two segments is the sum of weights corresponding to the graph edges that need to be severed to produce this segmentation. However, other graph cut formulations exist, so graph cut methods are categorized by the global cost function they optimize. If the objective is to minimize the cost of a cut between two or more segments, then the problem can be formulated as a min-cut and efficiently solved as a max-flow problem (Greig et al. 1989; Boykov and Funka-Lea 2006; Paragios et al. 2006; Boykov et al. 2001). This cost can include node priors by introducing *artificial* nodes corresponding to the number of desired segments. The edge weights from these nodes to the rest of the graph embed the node priors. In what follows, we will use the label *unnormalized graph cuts* to denote this graph cut problem.

Despite the merits of the unnormalized graph cut formulation and its proposed algorithms, it is biased towards producing cuts that contain a small number of nodes. Consequently, a normalized version of this cost function has been proposed, in which the cost of the normalized cut (NCUT) incorporates the association cost of each segment (i.e. the sum of edge weights connecting nodes of this segment to all other nodes in the graph), thus, suppressing the appearance of small cuts (Shi and Malik 2000). For a detailed comparison of these two formulations, refer to Weiss (1999). Minimizing the resulting NCUT objective function is an NP hard discrete optimization problem, so it is relaxed to take on real values. The popularity of this method can be partially attributed to its closed form approximation of the optimal solution, using generalized eigen-decomposition. Unconstrained NCUT has been used extensively for image segmentation and data clustering. In Yu and Shi (2004), the authors constrain the NCUT problem by incorporating additional grouping constraints, in the form of linear homogeneous equalities. This is extended to the case of non-homogeneous equalities in Eriksson et al. (2007).

In all the above NCUT methods (Shi and Malik 2000; Yu and Shi 2004; Eriksson et al. 2007), the NCUT solution is obtained from either a single generalized eigen-decomposition or a sequence of such decompositions. Even though these methods offer certain advantages, their following shortcomings need to be addressed. (1) These methods only allow for linear equality constraints. This is due to their underlying use of eigen decomposition to minimize the Rayleigh quotient. For example, Cour and Shi (2007) showed that it is NP-hard, in general, to solve for eigenvectors under linear inequalities. This suggests that if a unifying framework for such constrained NCUT problems is desired, it should avoid using this quotient formulation. (2) Unlike unnormalized graph cut techniques, these methods do not embed any unary terms (i.e. priors on individual graph nodes) in the NCUT cost function, which is equivalent to assuming a zero prior for all nodes. (3) Since these methods compute generalized eigenvectors of large matrices and form null spaces of highly rank deficient matrices, their computational complexity remains a practical issue, despite the special measures that were considered for complexity reduction.

Contributions In this paper, we present a first step in the direction of formulating a unifying framework for convexly constrained NCUT problems, which addresses the aforementioned limitations. The contributions of this framework are threefold. (i) It allows the efficient solution of the NCUT problem under general convex constraints. It uses the Dinkelbach method to transform the initial fractional problem into a sequence of convexly constrained, quadratic programming (QP) problems, whose convexity is ensured by a suitable initialization that we construct. More importantly, the global solutions of these problems converge superlinearly to the required solution of the fractional problem. Since the Dinkelbach method is central to our framework, we denote our proposed method as Dinkelbach NCUT (DNCUT). In fact, the Dinkelbach method was recently used in the context of parametric max-flow algorithms in Kolmogorov et al. (2007). However, the energy function to be minimized was unconstrained and hence no effort was made to construct a valid initialization that allows efficient estimation of the global solution for each Dinkelbach iteration. (ii) Our framework can incorporate prior knowledge of nodes belonging to different segments, which is equivalent to the unary term present in the energy function minimized by unnormalized graph cut algorithms. This case arises in various supervised problems including interactive segmentation and supervised clustering. These problems can not be solved using the current NCUT algorithms. (iii) Guaranteeing the convexity of each Dinkelbach sub-problem allows this framework to handle general convex constraints, due to the availability of efficient convex optimization techniques. Therefore, our proposed algorithm refrains from eigen-decomposition and, in the case of linear constraints, it degenerates into solving a set of sparse linear systems using conjugate gradients.

Mathematical Notation Before proceeding with the details of the paper, we summarize the mathematical notations used in the remainder of the paper. In what follows, a matrix is denoted by a bold, uppercase letter (e.g. $\mathbf{W}$). A vector is denoted by a bold, lowercase letter with an arrow above it (e.g. $\vec{\mathbf{x}}$). A scalar is denoted by a normal lowercase letter. Elements of vectors and matrices are indexed using parentheses. For example, $\mathbf{x}(i)$ represents the ith element of $\vec{\mathbf{x}}$, while $\mathbf{W}(i, j)$ represents the element of $\mathbf{W}$ at the ith row and the jth column. Furthermore, a parenthesized superscript is added to a variable to denote its value at a given iteration. For example, $\vec{\mathbf{x}}^{(k)}$ represents the value of $\vec{\mathbf{x}}$ at the kth iteration. Also, every optimization problem has a solution appended with a superscript $*$ (e.g. $\vec{\mathbf{x}}^{*}$) and constraints (if any) listed underneath the cost function prefixed by s.t., the abbreviation of such that.

We separate the discussion of applying hard and other general convex constraints to the NCUT problem. This is done because hard constraints have direct impact on the cost function itself, while the other constraints do not. In Sect. 2, we consider the problem of applying hard constraints (e.g. placed by the user via interactive placement of constraints, which we call here interactive NCUT). In Sect. 3, we describe the augmented graph structure used in formulating and solving the NCUT problem. In Sect. 4, we use the Dinkelbach approach (Dinkelbach 1967) to formulate a unifying framework for solving the convexly constrained NCUT problem. We also describe an algorithm to solve this problem with linear constraints. Finally, we test the algorithm by applying it to low-level segmentation of real images in Sect. 5.

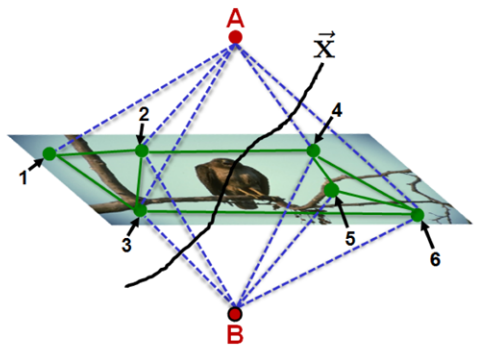

Fig. 1 (Color online) Example of a graph cut ($\vec{\mathbf{x}}$), which separates nodes $1, 2, 3$ (labeled as A) from $4, 5, 6$ (labeled as B). The cost of $\vec{\mathbf{x}}$ is the sum of the edge weights that have to be cut to produce the final labeling. Note that the regional costs are represented by the perforated *blue lines*, while *the solid green lines* represent the boundary costs

2 Hard Constraints

In this section, we consider the NCUT problem under the first type of convex constraints, namely hard constraints. To put this problem into context, we visit the unnormalized graph cut problem, which will also help us introduce the interactive version of the NCUT problem. Due to the nature of the two different cost functions optimized in the unnormalized graph cut problem and its corresponding NCUT problem, the analysis/solution of these two problems is quite different. However, we still make use of two general concepts used in the unnormalized graph cut context: the augmented graph structure and the notion that hard constraints directly affect the graph's weight matrix. We will elaborate on these two points in what follows. Next, we introduce the terminology to be used and give a brief description of the unnormalized graph cut and interactive unnormalized graph cut problems.

2.1 Unnormalized Graph Cuts and Hard Constraints

Let $\mathcal{G} = (\mathcal{P}, \mathcal{E})$ be an undirected graph, where $\mathcal{P}$ denotes the set of nodes contained in $\mathcal{G}$, and $\mathcal{E}$ the set of corresponding edges. Let $\mathbf{W}$ denote the edge weight matrix of $\mathcal{G}$, where $\mathbf{W}(i, j)$ is a non-negative weight associated with the edge connecting nodes i and j in $\mathcal{P}$. This non-negativity is the only restriction on $\mathbf{W}$, whereby the computation of pairwise edge weight values is application-dependent. When applied to low-level spectral graph image segmentation, an edge weight between two pixel nodes is commonly based on the similarity in color and/or image gradient between these two pixels. A graph cut $\vec{\mathbf{x}}$ is a binary segmentation of the nodes in $\mathcal{P}$, where $\forall k = 1, \dots, |\mathcal{P}|$, $\mathbf{x}(k) = 1$ if the kth node belongs to segment A and $\mathbf{x}(k) = 0$ if it belongs to segment B (refer to Fig. 1). The aim of a graph cuts algorithm is to find $\vec{\mathbf{x}}$ that minimizes a given cost function. In Greig et al. (1989), the cost function was defined as $E_{\vec{\mathbf{x}}}(A, B) = \lambda R(\vec{\mathbf{x}}) + S(\vec{\mathbf{x}})$. The regional cost term, $R(\vec{\mathbf{x}}) = \sum_{i \in \mathcal{P}} R_i(\mathbf{x}(i))$, is the sum of costs incurred by assigning each node i to its label $\mathbf{x}(i)$. The boundary (cut) term, $S(\vec{\mathbf{x}}) = cut(A, B) = \sum_{i \in A, j \in B} \mathbf{W}(i, j)$, is the cost of the cut resulting in segments A and B. Numerous efficient algorithms have been developed to find the global minimizer of $E_{\vec{\mathbf{x}}}(A, B)$ (Boykov et al. 2001).

The interactive unnormalized graph cuts problem is an extension of the aforementioned problem with the addition that some hard constraints are applied to the nodes of $\mathcal{P}$ (i.e. labels of some nodes are known beforehand). These constraints can originate from direct user interaction or domain specific knowledge. Many recent works have addressed this problem under different forms of user interaction (Boykov and Jolly 2001; Rother et al. 2004; Li et al. 2004). We denote S_A and S_B to be the sets of nodes satisfying the hard constraints (i.e. whose labels are known beforehand to be 1 or 0, respectively). Under these constraints, the minimization problem becomes more difficult to solve. However, Boykov and Jolly (2001) proved that there is no need to explicitly solve this new problem. They show that it is equivalent to an unconstrained unnormalized graph cut problem on $\mathcal{G}$ with the edge weights appropriately modified to implicitly reflect the hard constraints. While such an equivalence exists for the interactive unnormalized graph cut problem, it does not transfer to the corresponding NCUT problem, which we will appropriately call the *interactive NCUT problem*. Next, we show how these hard constraints are *explicitly* applied to this NCUT problem. We study this case independently, since it will have direct influence on the cost function to be minimized.

2.2 NCUT and Hard Constraints

A normalized cut of $\mathcal{G}$ into segments A and B has the following cost: $\mathrm{NCUT}_{\vec{\mathbf{x}}}(A, B) = \frac{cut(A,B)}{\sum_{i \in A, j \in \mathcal{P}} \mathbf{W}(i,j)} +$

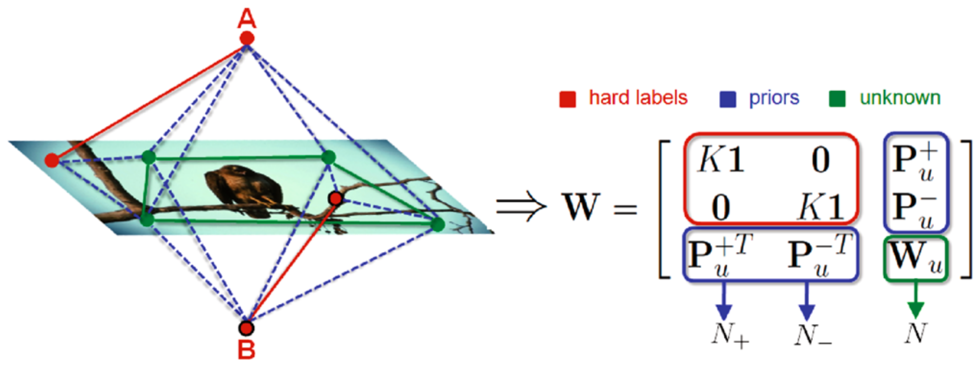
A
B
🟥 hard labels
🟦 priors
🟩 unknown
$\Rightarrow \mathbf{W} = \begin{bmatrix} K\mathbf{1} & \mathbf{0} & \mathbf{P}_u^+ \\ \mathbf{0} & K\mathbf{1} & \mathbf{P}_u^- \\ \mathbf{P}_u^{+T} & \mathbf{P}_u^{-T} & \mathbf{W}_u \end{bmatrix}$
N_+
N_-
N

Fig. 2 (Color online) Example of a graph with hard constraints colored in *red*. Here, $N_+ = N_- = 2$, where two *pixel* nodes are already labeled as A and B. The perforated *blue* edges are designated weights that embed prior knowledge of the unknown nodes. The bold *red* edges have weights of K. Bold *green* edges represent the similarity between the unknown nodes they connect

$\frac{cut(A,B)}{\sum_{i\in B, j\in\mathcal{P}} \mathbf{W}(i,j)}$, where the cut cost is normalized by the association cost of each segment. The NCUT formulation defined in Shi and Malik (2000) is:

$$\vec{\mathbf{y}}_{\text{NCUT}} = \arg\min \frac{\vec{\mathbf{y}}^T(\mathbf{D}-\mathbf{W})\vec{\mathbf{y}}}{\vec{\mathbf{y}}^T\mathbf{D}\vec{\mathbf{y}}}$$
$$\text{s.t.} \quad \begin{cases} \mathbf{y}(i) \in \{1, -b\}, & \forall i = 1, \ldots, |\mathcal{P}|, \\ \vec{\mathbf{y}}^T\mathbf{D}\vec{\mathbf{1}} = 0 \end{cases} \qquad (1)$$

where $\mathbf{D}$ is the degree matrix of $\mathcal{G}$ (i.e. $\mathbf{D} = \text{diag}(\mathbf{W}\vec{\mathbf{1}})$) and $b = \frac{\sum_{\mathbf{x}(i)>0}\mathbf{D}(i,i)}{\sum_{\mathbf{x}(i)<0}\mathbf{D}(i,i)}$. Here, we note that b quantifies how connected (similar) each segment is to the rest of the graph. A large value of b (>1) means that the nodes of segment A have stronger connections to the rest of the graph than those of segment B. Since the value of b is dependent on the final labeling, solving this problem exactly becomes infeasible. In general, the optimization problem in (1) is NP hard, so it is relaxed to render a real solution. This vector is discretized as a post processing step, which does not incorporate the value of b. In Shi and Malik (2000), the authors show that the solution to the relaxed problem can be obtained in closed form by solving a generalized eigenvalue problem (i.e. eig$(\mathbf{D}-\mathbf{W}, \mathbf{D})$). This is a direct conclusion from the fact that this relaxed problem is in the form of a Rayleigh quotient. In fact, $\vec{\mathbf{y}}_{\text{NCUT}}$ is computed as the generalized eigenvector corresponding to the second smallest eigenvalue. In Shi and Malik (2000), the NCUT framework was applied to low-level image segmentation and the edge weight between each pair of nodes/pixels was computed using intervening contours (Malik et al. 2001).

Now, let us extend (1) to include some hard constraints, rendering the interactive NCUT problem. To the best of our knowledge, the interactive NCUT problem has not been addressed previously in the literature. To solve this problem, we cannot use the same graph used in the traditional NCUT formulation. This graph is composed solely of *pixel* nodes originating from the image itself. To this graph, we add two *artificial* nodes A and B to represent the two segments, thus, differentiating them from the *pixel* nodes. This augmented graph will allow us to incorporate hard constraints and include prior knowledge. We decompose the labeling vector $\vec{\mathbf{y}}$ into three disjoint parts: (1) $\vec{\mathbf{y}}_+$ (of size $N_+ = |S_A|$), which corresponds to the pre-labeled nodes of S_A, (2) $\vec{\mathbf{y}}_-$ (of size $N_- = |S_B|$), which corresponds to the pre-labeled nodes of S_B, and (3) $\vec{\mathbf{y}}_u$ (of size N_u), which designates the unknown labels of the rest of the nodes in the graph. We use the augmented graph in Fig. 2 to illustrate an example. We update $\mathbf{W}$ and $\mathbf{D}$ to include the hard labels, as shown below. The nodes corresponding to $\vec{\mathbf{y}}_+$ are listed first, followed by those of $\vec{\mathbf{y}}_-$, and then $\vec{\mathbf{y}}_u$. Similar to the interactive unnormalized graph cut problem seen before, we embed the hard constraints into the edge weights by setting $\mathbf{W}(i,j) = \mathbf{W}(i',j') = K$, $\mathbf{W}(i,i') = \mathbf{W}(j,j') = 0$ for every node pair $(i,j) \in S_A \times S_A$ and $(i',j') \in S_B \times S_B$. This means that nodes belonging to the same segment are maximally similar, where the similarity value is a constant denoted by K. For now, K and b are left unknown. In Sect. 4.2.2, we show how they are computed. Similarly, the nodes belonging to different segments are minimally similar. Furthermore, we set $\mathbf{W}(i,j) = \mathbf{p}_u^+(i)$ and $\mathbf{W}(i,j') = \mathbf{p}_u^-(i)$ for every $i \notin S_A \cup S_B$, $j \in S_A$, and $j' \in S_B$. Here, prior knowledge can be incorporated. The edge weight between an unknown node and the hard constraints (i.e. the pre-labeled nodes) can be viewed as the likelihood of that unknown node belonging to either segment. So, if a likelihood node model (e.g. a Gaussian Mixture Model (GMM)) is available, it is used to evaluate $\vec{\mathbf{p}}_u^+$ and $\vec{\mathbf{p}}_u^-$. In our experiments, given hard constraints, we assume the Nearest Neighbor model, so we set $\mathbf{p}_u^+(i)$ and $\mathbf{p}_u^-(i)$ to the maximum edge weight between node i and nodes of S_A and S_B respectively. In the absence of prior knowledge, these edge weights are set to the same constant. In our experiments and in the absence of hard constraints, we set $\mathbf{p}_u^+(i) = \mathbf{p}_u^-(i) = \frac{1}{2}$, since all edge weights (based on intervening contours Malik et al. 2001) take values in $[0, 1]$ for the case of low-level image segmentation. Consequently, the updated $\mathbf{W}$ and $\mathbf{D}$ matrices are:

$$\mathbf{W} = \begin{bmatrix} K\mathbf{1} & \mathbf{0} & \mathbf{P}_u^+ \\ \mathbf{0} & K\mathbf{1} & \mathbf{P}_u^- \\ \mathbf{P}_u^{+T} & \mathbf{P}_u^{-T} & \mathbf{W}_u \end{bmatrix};$$

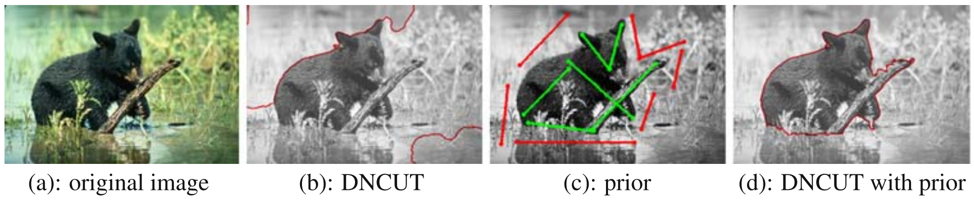

Fig. 3 (Color online) For the image in (**a**), the unconstrained binary DNCUT segmentation is provided in (**b**). This segmentation is obtained when all unknown nodes have equally likely prior probabilities of belonging to S_A and S_B (i.e. $\mathbf{p}_u^+(i) = \mathbf{p}_u^-(i) = \frac{1}{2}$). However, (**d**) represents the DNCUT solution when nonuniform prior probabilities are used. These probabilities are based on how similar the unknown nodes are to the nodes delineated by the *green* (S_A) and *red* (S_B) strokes. We used a simple GMM (two Gaussians) likelihood model. Note how the incorporation of prior knowledge rendered the binary DNCUT segmentation more meaningful

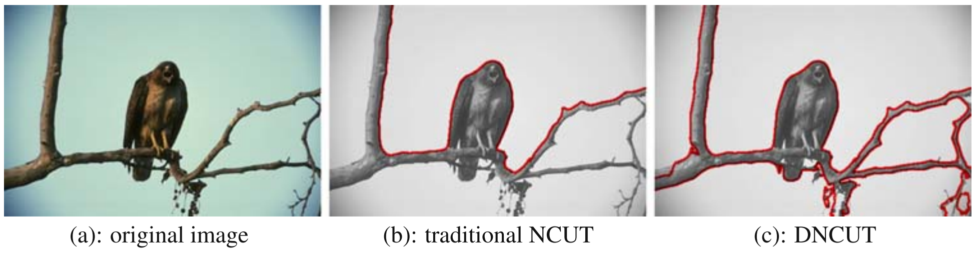

Fig. 4 For the image in (**a**), the traditional NCUT solution to the unconstrained NCUT problem is illustrated in (**b**) and the DNCUT solution in (**c**). Note that similar *pixel* nodes (i.e. sky) are assigned to different segments in (**b**) and to the same segment in (**c**)

$$\mathbf{D} = \begin{bmatrix} (KN_+ + \vec{\mathbf{1}}^T\overrightarrow{\mathbf{p}_u^+})\mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & (KN_- + \vec{\mathbf{1}}^T\overrightarrow{\mathbf{p}_u^-})\mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{D}_u + \mathbf{P}_D \end{bmatrix}$$

where $\mathbf{P}_D = \mathrm{diag}(N_+\overrightarrow{\mathbf{p}_u^+} + N_-\overrightarrow{\mathbf{p}_u^-})$, $\mathbf{P}_u^+ = \vec{\mathbf{1}}(\overrightarrow{\mathbf{p}_u^+})^T$, and $\mathbf{P}_u^- = \vec{\mathbf{1}}(\overrightarrow{\mathbf{p}_u^-})^T$ (refer to Fig. 2).

3 DNCUT Graph Structure

As opposed to the traditional NCUT formulation that only uses *pixel* nodes to construct the graph, we adopt the augmented graph structure used in the unnormalized graph cut problem. As illustrated before in Fig. 2, the augmented graph adds *artificial* nodes to the original graph. These added nodes correspond to the "sink" and "source" terminals in Paragios et al. (2006). This augmented graph structure allows for two major benefits.

1. It inherently incorporates prior knowledge. We designate $\overrightarrow{\mathbf{p}_u^+}$ and $\overrightarrow{\mathbf{p}_u^-}$ to be the vectors of edge weights connecting the unknown nodes to the nodes in segments S_A and S_B respectively. In fact, they correspond to the unary, regional cost terms included in the unnormalized graph cut formulation. They can also be viewed as the likelihoods that the unknown nodes belong to each of the two segments. As such, the traditional NCUT problem becomes a special case of the DNCUT framework, when $\overrightarrow{\mathbf{p}_u^+} = \overrightarrow{\mathbf{p}_u^-} \to \vec{\mathbf{0}}$ (i.e. zero priors) and $K \to 0$. Figure 3 shows an example of how prior knowledge on the nodes of the augmented graph can improve segmentation/labelling results. Figure 3(b) is the binary DNCUT segmentation, if uniform prior knowledge is assumed on the nodes of this graph. The prior knowledge in Fig. 3(c) depicts how nodes belonging to class S_A and S_B should look like in the image. Note the significant difference in segmentation when comparing Fig. 3(b) to Fig. 3(d). Traditional NCUT methods cannot incorporate prior knowledge on the nodes of the graph.
2. It supports an indirect connection between every pair of *pixel* nodes in the graph, while preserving the sparsity of $\mathbf{W}$. In the case of image segmentation and due to memory restrictions, $\mathbf{W}$ is made sparse by setting the edge weights between far pixels to 0. For the traditional NCUT problem, this can yield pairs of nodes that have high similarity but are neither directly nor indirectly connected in the graph. This biases the segmentation to assign such nodes to different segments. This usually occurs due to occlusions. On the other hand, the DNCUT framework ensures an indirect connection between these nodes, via the nodes in S_A and S_B, thus, alleviating the previous segmentation bias. In Fig. 4, we show an example of the traditional NCUT and the DNCUT solutions to the unconstrained NCUT problem described in Sect. 2.2 (Shi and Malik 2000). Note that these solutions are binary solutions obtained by discretizing the real solutions to the unconstrained NCUT problem.

4 DNCUT Framework Under Hard & Convex Constraints

In this section, we highlight the details of our proposed DNCUT framework. We consider the convexly constrained NCUT problem, where hard and/or other convex constraints are applied. Given sets S_A and S_B and the updated weight matrix $\mathbf{W}$, we formally define the problem as a fractional quadratic program (FQP) with convex constraints, as shown in (2). Note that the following setup is the same even if no hard constraints exist. For this special case, S_A and S_B only contain a single node each (i.e. $N_+ = N_- = 1$).

$$\vec{\mathbf{y}}_u^* = \arg\min \frac{\vec{\mathbf{y}}_u^T \mathbf{Q} \vec{\mathbf{y}}_u + \vec{\mathbf{m}}^T \vec{\mathbf{y}}_u + (a-c)}{\vec{\mathbf{y}}_u^T \mathbf{R} \vec{\mathbf{y}}_u + a}$$
$$\text{s.t.} \begin{cases} \mathbf{y}_u(i) \in \{1, -b\}, & \forall i = 1, \dots, N_u, \\ \vec{\mathbf{y}}_u^T \mathbf{R}\vec{\mathbf{1}} + q = 0, \\ \Phi_i(\vec{\mathbf{y}}_u) \le 0, & \forall i = 1, \dots, |\mathrm{I}|, \\ \Psi_j(\vec{\mathbf{y}}_u) = 0, & \forall j = 1, \dots, |\mathrm{E}|, \end{cases} \tag{2}$$

where

$$\begin{cases} \mathbf{Q} = (\mathbf{D}_u - \mathbf{W}_u) + \mathbf{P}_D = \mathbf{L}_u + \mathbf{P}_D, \quad \mathbf{Q} \in \mathbb{S}^+_{N_u}, \\ \vec{\mathbf{m}} = 2(bN_- \vec{\mathbf{p}_u^-} - N_+ \vec{\mathbf{p}_u^+}), \\ a = N_+(KN_+ + \vec{\mathbf{1}}^T \vec{\mathbf{p}_u^+}) + b^2 N_-(KN_- + \vec{\mathbf{1}}^T \vec{\mathbf{p}_u^-}), \\ c = K[(N_+)^2 + (bN_-)^2], \\ \mathbf{R} = \mathbf{D}_u + \mathbf{P}_D, \quad \mathbf{R} \in \mathbb{S}^+_{N_u}, \\ q = N_+(KN_+ + \vec{1}^T \vec{\mathbf{p}_u^+}) - bN_-(KN_- + \vec{\mathbf{1}}^T \vec{\mathbf{p}_u^-}), \\ \Phi_i(\vec{\mathbf{x}}) \text{ and } \Psi_j(\vec{\mathbf{x}}) \text{ are convex} \\ \quad \forall i = 1, \dots, |\mathrm{I}|, j = 1, \dots, |\mathrm{E}|. \end{cases} \tag{3}$$

We define the Laplacian matrix corresponding to the unknown nodes as $\mathbf{L}_u$, which is known to belong to $\mathbb{S}^+_{N_u} = \{\mathbf{X} \in \mathbb{R}^{N_u \times N_u} : \mathbf{X} = \mathbf{X}^T, \ \mathbf{X} \succeq \mathbf{0}\}$ (i.e. the set of symmetric positive semi-definite matrices of size $N_u \times N_u$) (Pothen et al. 1990). For image segmentation, $\mathbf{W}_u$ and $\mathbf{Q}$ are sparse, in general. $\Phi_i(\vec{\mathbf{y}}_u)$ and $\Psi_j(\vec{\mathbf{y}}_u)$ are general convex constraints that can be linear (e.g. partial groupings Yu and Shi 2004) or non-linear (e.g. upper bounds on $\|\vec{\mathbf{y}}_u\|^2$).

We keep b and K as scalar variables. Consequently, $\vec{\mathbf{m}}$, a, c, and q are variables too. Note that the value of b depends on $\vec{\mathbf{y}}_u^*$ itself and that the problem in (2) is still an NP hard discrete optimization problem, so we propose two forms of relaxation: (1) we relax $\vec{\mathbf{y}}_u$ to take on real values and (2) we assume b takes on a constant value b_0. In Sect. 4.2.2, we show how K and b_0 are computed in the DNCUT framework. The relaxed problem becomes the FQP defined in (4).

$$\vec{\mathbf{y}}_u^* = \arg\min \left[H(\vec{\mathbf{y}}_u) = \frac{F(\vec{\mathbf{y}}_u)}{G(\vec{\mathbf{y}}_u)} \right]$$
$$= \arg\min \frac{\vec{\mathbf{y}}_u^T \mathbf{Q} \vec{\mathbf{y}}_u + \vec{\mathbf{m}}^T \vec{\mathbf{y}}_u + (a-c)}{\vec{\mathbf{y}}_u^T \mathbf{R} \vec{\mathbf{y}}_u + a}$$
$$\text{s.t.} \begin{cases} \vec{\mathbf{y}}_u^T \mathbf{R}\vec{\mathbf{1}} + q = 0, \\ \Phi_i(\vec{\mathbf{y}}_u) \le 0, & \forall i = 1, \dots, |\mathrm{I}|, \\ \Psi_j(\vec{\mathbf{y}}_u) = 0, & \forall j = 1, \dots, |\mathrm{E}|. \end{cases} \tag{4}$$

Equation (4) is no longer a Rayleigh quotient, solvable by general eigen-decomposition, as compared to the traditional unconstrained NCUT formulation. As such, there is no closed form solution for this problem. In particular, Cour and Shi (2007) showed that it is NP-hard, in general, to solve for general eigenvectors under linear inequalities. So, we propose an iterative algorithm to find the global minimum of this optimization problem using Dinkelbach's method for fractional programming (FP) (Dinkelbach 1967; Rodenas et al. 1999). To make the paper self-contained, we give a brief description of this algorithm next.

4.1 Dinkelbach Algorithm for Fractional Programming

Given two continuous functions $f : \mathbb{R}^n \to \mathbb{R}$ and $g : \mathbb{R}^n \to \mathbb{R}$ defined on a compact set $\mathcal{S} \subseteq \mathbb{R}^n$ such that $g(\vec{\mathbf{x}}) > 0 \ \forall \vec{\mathbf{x}} \in \mathcal{S}$, the fractional programming problem is to find the global minimizer, $\vec{\mathbf{x}}^*$, of $h(\vec{\mathbf{x}}) = \frac{f(\vec{\mathbf{x}})}{g(\vec{\mathbf{x}})}$. According to the parametric approach of Dinkelbach (1967), $\vec{\mathbf{x}}^*$ minimizes this problem if and only if $\mathcal{F}(\vec{\mathbf{x}}^*, \lambda^*) = \min_{\vec{\mathbf{x}} \in \mathcal{S}} [f(\vec{\mathbf{x}}) - \lambda^* g(\vec{\mathbf{x}})]$, where $\lambda^* = h(\vec{\mathbf{x}}^*)$. Dinkelbach proved that $\mathcal{F}(\vec{\mathbf{x}}, \lambda)$ is monotonically decreasing in λ. This equivalence was extended to prove that λ^* can be reached iteratively. In fact, he proposed an algorithm that produces a sequence of monotonically decreasing values of $\lambda^{(i)} = h(\vec{\mathbf{x}}^{(i)})$, which converges superlinearly to λ^*.

The Dinkelbach algorithm was extended by Rodenas et al. (1999) to provide a general framework for FP's, summarized below as Algorithm 1. λ^* is the global minimum value of the objective function. Here, we emphasize that the superlinear convergence property is *only* regarding the iterations needed to achieve λ^* and not the convergence of each iteration. Note that each iteration involves solving a different optimization problem, which might be an NP hard problem in its own right! Consequently, choosing the initial guess $\vec{\mathbf{x}}^{(0)}$ is not a trivial task, since setting it to an arbitrary value might lead to a sequence of NP hard problems. However, we choose $\vec{\mathbf{x}}^{(0)}$ to reduce the computational complexity of each iteration and the total number of iterations required.

4.2 Applying the Dinkelbach Algorithm to (4)

Equation (4) fits the form required to apply the Dinkelbach algorithm, where $\mathcal{S} = \{\vec{\mathbf{y}}_u : \vec{\mathbf{y}}_u^T \mathbf{R}\vec{\mathbf{1}} + q = 0, \Phi_i(\vec{\mathbf{y}}_u) \le 0 \ \forall i =$

Algorithm 1: *Dinkelbach*

```
Input  : $\mathcal{S}$, $f$, $g$, $\vec{\mathbf{x}}^{(0)} \in \mathcal{S}$, $\epsilon$
Output: $N_\epsilon$, $\{\lambda^{(i)}\}_{i=0}^{N_\epsilon}$, $\vec{\mathbf{x}}^*$, and $\lambda^*$
1  begin
2      Initialization: $\lambda^{(0)} = h(\vec{\mathbf{x}}^{(0)}) = \frac{f(\vec{\mathbf{x}}^{(0)})}{g(\vec{\mathbf{x}}^{(0)})}$; $i = 0$;
       $\delta(i) \leftarrow \infty$; $N_\epsilon = 1$; $\lambda^* = \lambda^{(0)}$
3      while $\delta(i) > \epsilon$ do
4          Solve $\vec{\mathbf{x}}^* = \arg\min_{\vec{\mathbf{x}} \in \mathcal{S}}[f(\vec{\mathbf{x}}) - \lambda^{(i)} g(\vec{\mathbf{x}})]$
5          $\delta(i+1) = |f(\vec{\mathbf{x}}^*) - \lambda^{(i)} g(\vec{\mathbf{x}}^*)|$
6          $N_\epsilon \leftarrow N_\epsilon + 1$, $i \leftarrow i + 1$
7          If $\delta(i+1) \leq \epsilon$: $\lambda^* = h(\vec{\mathbf{x}}^*)$, break
8          else $\lambda^{(i+1)} = h(\vec{\mathbf{x}}^*)$
9          end
10     end
11 end
```

$1, \ldots, |\mathrm{I}|, \Psi_j(\vec{\mathbf{y}}_u) = 0, \forall j = 1, \ldots, |\mathrm{E}|\}$, $f = F$, $g = G$, and $\vec{\mathbf{y}}_u^{(0)} = \vec{\mathbf{x}}^{(0)}$. The resulting optimization problem to be solved in each iteration of *Dinkelbach*$(\mathcal{S}, F, G, \vec{\mathbf{y}}_u^{(0)}, \epsilon)$ is a QP subject to convex constraints. Equation (5) shows the problem to be solved at iteration i.

$$\vec{\mathbf{y}}_u^* = \arg\min\left[\vec{\mathbf{y}}_u^T\left(\mathbf{Q} - \lambda^{(i)}\mathbf{R}\right)\vec{\mathbf{y}}_u + \vec{\mathbf{m}}^T\vec{\mathbf{y}}_u + \left(1 - \lambda^{(i)}\right)a - c\right]$$
$$\text{s.t.} \quad \vec{\mathbf{y}}_u \in \mathcal{S}. \tag{5}$$

The computational cost of each iteration is highly dependent on the nature of the matrix $\mathbf{M}^{(i)} = \mathbf{Q} - \lambda^{(i)}\mathbf{R}$. If $\mathbf{M}^{(i)} \succeq \mathbf{0}$, the problem becomes a convex QP, whose global minimum can be found efficiently depending on the nature of $\mathcal{S}$. However, if $\mathbf{M}^{(i)}$ has at least one negative eigenvalue, then finding the global minimum of (5) becomes NP hard (Pardalos and Vavasis 2004). Therefore, it is essential that we study the existence/uniqueness of an initial guess $\vec{\mathbf{y}}_u^{(0)}$ that guarantees the convexity of each *Dinkelbach* iteration or equivalently the positive semi-definiteness of each $\mathbf{M}^{(i)}$.

4.2.1 *Dinkelbach Initialization for (4): $\lambda^{(0)}$*

In what follows, we will determine an α bound on $\lambda^{(0)}$ that ensures $\mathbf{M}^{(i)} \succeq \mathbf{0}$ $\forall i = 0, \ldots, N_\epsilon$, thus, making (5) convex for every iteration (refer to Theorem 1). Furthermore, we show how to construct a valid $\vec{\mathbf{y}}_u^{(0)}$ that satisfies this α bound.

Theorem 1 (α Bound on $\lambda^{(0)}$) *If $\lambda^{(0)} \leq \alpha$, $\mathbf{M}^{(i)} \succeq \mathbf{0}$ $\forall i = 0, \ldots, N_\epsilon$. Here, $\alpha = \frac{\min(N_+\overrightarrow{\mathbf{p}_u^+} + N_-\overrightarrow{\mathbf{p}_u^-})}{\max(\mathbf{R})}$ is a non-trivial, upper bound computed without eigen-decomposition.*

Proof We propose to select $\lambda^{(0)}$ in order to guarantee $\mathbf{M}^{(i)} \succeq \mathbf{0}$ $\forall i = 0, \ldots, N_\epsilon$. Actually, this is equivalent to ensuring $\mathbf{M}^{(0)} \succeq 0$, since there exists a recursive relationship between $\mathbf{M}^{(i)}$ and $\mathbf{M}^{(0)}$: $\mathbf{M}^{(i)} = \mathbf{M}^{(0)} + (\lambda^{(0)} - \lambda^{(i)})\mathbf{R}$. Notice that $\mathbf{R} \succeq \mathbf{0}$ and $\lambda^{(0)} > \lambda^{(i)}$ $\forall i = 1, \ldots, N_\epsilon$ (refer to Sect. 4.1). To do this, we study the relationship between the eigenvalues of $\mathbf{M}^{(0)}$, $\mathbf{P}_D$, $\mathbf{R}$, and $\mathbf{Q}$, which are matrices in $\mathbb{R}^{N_u \times N_u}$. Here, we use the eigenvalue notation $\rho_k(\mathbf{B})$ to denote the kth largest eigenvalue of matrix $\mathbf{B}$. Using the results of Bhatia (1997), Thompson and Freede (1970), we bound the eigenvalues of $\mathbf{M}^{(0)}$ as follows:

$$\rho_{N_u}(\mathbf{Q}) \leq \rho_j(\mathbf{M}^{(0)}) + \lambda^{(0)}\rho_j(\mathbf{R}) \leq \rho_1(\mathbf{Q}) \quad \forall j = 1, \ldots, N_u.$$

Since $\mathbf{Q}, \mathbf{P}_D \in \mathbb{S}_{N_u}^+$, we can bound the eigenvalues of $\mathbf{Q}$ in a similar manner to give: $\rho_{N_u}(\mathbf{Q}) \geq \rho_{N_u}(\mathbf{P}_D) + \rho_{N_u}(\mathbf{L}_u)$. Here, we use the fact that the smallest eigenvalue of a Laplacian matrix is zero (Shi and Malik 2000) (i.e. $\rho_{N_u}(\mathbf{L}_u) = 0$). This step is performed to avoid calculating $\rho_{N_u}(\mathbf{Q})$, which is computationally expensive. But, it also loosens the bound on $\lambda^{(0)}$, which may lead to slower convergence. However, even with this step, our experiments show that merely 1–3 *Dinkelbach* iterations are needed for convergence.

Combining the above results, we find that $\rho_j(\mathbf{M}^{(0)}) \geq \min(\mathrm{diag}(\mathbf{P}_D)) - \lambda^{(0)}\mathbf{R}(j, j)$. This simplification is possible because $\mathbf{P}_D$ is a diagonal matrix and its eigenvalues are its diagonal elements themselves. For $\mathbf{M}^{(0)} \succeq \mathbf{0}$, we require that $\rho_j(\mathbf{M}^{(0)}) \geq 0$ $\forall j = 1, \ldots, N_u$. This can be achieved when: $\lambda^{(0)} \leq \alpha$, where $\alpha = \frac{\min(N_+\overrightarrow{\mathbf{p}_u^+} + N_-\overrightarrow{\mathbf{p}_u^-})}{\max(\mathbf{R})}$. □

4.2.2 *Dinkelbach Initialization for (4): $\vec{\mathbf{y}}_u^{(0)}$*

Theorem 1 proved the existence of a non-trivial upper bound for $\lambda^{(0)}$; however, it did not show how to find a particular value of $\vec{\mathbf{y}}_u^{(0)}$ that satisfies this bound (if one exists). Here, we construct such a $\vec{\mathbf{y}}_u^{(0)}$, by setting b_0 and K to appropriate values. From $\lambda^{(0)} = H(\vec{\mathbf{y}}_u^{(0)}) \leq \alpha$, we obtain a quadratic feasibility problem with convex constraints, as shown in (6), with $(\mathbf{Q} - \alpha\mathbf{R}) \succeq \mathbf{0}$.

$$\begin{cases} \text{(I):} & \vec{\mathbf{y}}_u^{(0)T}(\mathbf{Q} - \alpha\mathbf{R})\vec{\mathbf{y}}_u^{(0)} + \vec{\mathbf{m}}^T\vec{\mathbf{y}}_u^{(0)} + (1 - \alpha)a - c \leq 0, \\ \text{(II):} & \vec{\mathbf{d}}^T\vec{\mathbf{y}}_u^{(0)} + q = 0 \quad (\vec{\mathbf{d}} = \mathbf{R}\vec{\mathbf{1}}), \\ \text{(III):} & \Phi_i(\vec{\mathbf{y}}_u^{(0)}) \leq 0, \qquad \Psi_j(\vec{\mathbf{y}}_u^{(0)}) = 0, \\ & \quad \forall\, i = 1, \ldots, |\mathrm{I}|,\ \forall j = 1, \ldots, |\mathrm{E}|. \end{cases} \tag{6}$$

We define

$$\begin{cases} \beta_1 = -\alpha[(N_+)^2 + (bN_-)^2], \\ \beta_2 = (1 - \alpha)[N_+(\vec{1}^T\overrightarrow{\mathbf{p}_u^+}) + b^2N_-(\vec{\mathbf{1}}^T\overrightarrow{\mathbf{p}_u^-})], \\ \beta_3 = [(N_+)^2 - b(N_-)^2], \\ \beta_4 = [N_+(\vec{1}^T\overrightarrow{\mathbf{p}_u^+}) - bN_-(\vec{\mathbf{1}}^T\overrightarrow{\mathbf{p}_u^-})], \\ (1 - \alpha)a - c = \beta_1 K + \beta_2, \\ q = \beta_3 K + \beta_4. \end{cases}$$

Replacing (II) (i.e. $K = -\frac{\vec{\mathbf{d}}^T\vec{\mathbf{y}}_u^{(0)}+\beta_4}{\beta_3}$) in (I) and enforcing that $K \geq 0$, we obtain an equivalent feasibility problem, where $r(b) = \frac{\beta_1\beta_4}{\beta_3} - \beta_2$, as shown in (7). Now, all three constraints are dependent on the value of b alone.

$$\begin{cases} \text{(I):} & \vec{\mathbf{y}}_u^{(0)T}(\mathbf{Q}-\alpha\mathbf{R})\vec{\mathbf{y}}_u^{(0)} + (\vec{\mathbf{m}} - \frac{\beta_1}{\beta_3}\vec{\mathbf{d}})^T\vec{\mathbf{y}}_u^{(0)} \leq r(b), \\ \text{(II):} & \frac{\vec{\mathbf{d}}^T\vec{\mathbf{y}}_u^{(0)}+\beta_4}{\beta_3} \leq 0, \\ \text{(III):} & \Phi_i(\vec{\mathbf{y}}_u^{(0)}) \leq 0, \qquad \Psi_j(\vec{\mathbf{y}}_u^{(0)}) = 0, \\ & \qquad \forall i = 1,\dots,|\mathrm{I}|,\ \forall j = 1,\dots,|\mathrm{E}|. \end{cases} \tag{7}$$

By setting b to $b_0 = \arg\max_{b\geq 0} r(b)$, we maximize the upper bound on (I). In fact, it can be shown (refer to Appendix A) that this bound can be made arbitrarily large, if $b_0 = \kappa(\frac{N_+}{N_-})^2$, where κ is chosen in the following manner.

$$\begin{cases} \kappa = 1-\varepsilon, & \text{if } \tau \leq 1, \\ \kappa = 1+\varepsilon, & \text{if } \tau > 1, \end{cases} \quad \text{where} \quad \begin{cases} \varepsilon \geq 0, \quad \varepsilon \to 0, \\ \tau = \frac{N_-(\vec{\mathbf{1}}^T\overrightarrow{\mathbf{p}_u^+})}{N_+(\vec{\mathbf{1}}^T\overrightarrow{\mathbf{p}_u^-})}. \end{cases}$$

As stated in Sect. 2.2, b quantifies how connected each segment is to the rest of the graph. Ideally, the value of b is dependent on the final solution $\vec{\mathbf{y}}_u^*$. However, subject to the hard constraints (if any) and prior to computing $\vec{\mathbf{y}}_u^*$, b_0 is a reasonable estimate of b. In the absence of hard constraints (i.e. $N_+ = N_- = 1$), $b_0 \to 1$. In the traditional NCUT formulation (Shi and Malik 2000), no attempts were made to approximate b.

Therefore, by setting $b = b_0$, we can construct a feasible solution to (7) by solving the convex QP shown in (8). Due to the convexity of the cost function and the constraints in (8), we are guaranteed that a solution exists. This solution can be obtained by using a suitable QP solver (e.g. methods based on trust regions, active sets, etc.) to find the global minimum. Moreover, the sparsity of $(\mathbf{Q}-\alpha\mathbf{R})$ should be exploited to reduce the overall computational complexity.

$$\vec{\mathbf{y}}_u^{(0)} = \arg\min_{\vec{\mathbf{x}}}\left[\vec{\mathbf{x}}^T(\mathbf{Q}-\alpha\mathbf{R})\vec{\mathbf{x}} + \left(\vec{\mathbf{m}} - \frac{\beta_1}{\beta_3}\vec{\mathbf{d}}\right)^T\vec{\mathbf{x}}\right]$$
$$\text{s.t.} \quad \begin{cases} \frac{\vec{\mathbf{d}}^T\vec{\mathbf{x}}+\beta_4}{\beta_3} \leq 0, \\ \Phi_i(\vec{\mathbf{x}}) \leq 0, \qquad \Psi_j(\vec{\mathbf{x}}) = 0. \end{cases} \tag{8}$$

4.3 Proposed DNCUT Algorithm

In Algorithm 2, we show the steps required to solve a convexly constrained NCUT problem, under the DNCUT framework. This algorithm reiterates how b, K, and $\vec{\mathbf{y}}_u^{(0)}$ are computed to ensure that $\mathbf{M}^{(i)} \succeq 0$ in every iteration of the *Dinkelbach* algorithm. Notice that the global minima of (5) and (8) are computed using the same convex QP solver. We

Algorithm 2: DNCUT

```
Input  : $\mathcal{S}$, $\mathbf{W}_u$, $\overrightarrow{\mathbf{p}_u^+}$, $\overrightarrow{\mathbf{p}_u^-}$, $N_+$, $N_-$, $\epsilon$, and $\varepsilon$
Output: $\vec{\mathbf{y}}_u^*$ and $\lambda^*$
1 begin
2     Use $\varepsilon$ to compute $b = b_0 = \kappa(\frac{N_+}{N_-})^2$
3     Store $\mathbf{P}_D$, $\mathbf{Q}$, $\mathbf{R}$, $\vec{\mathbf{m}}$, and $\vec{\mathbf{d}}$
4     Compute $\alpha$, $\beta_1$, $\beta_2$, $\beta_3$, and $\beta_4$
5     Solve (8) $\Rightarrow \vec{\mathbf{y}}_u^{(0)}$ [QP Solver]
6     Compute $K = -\frac{\vec{\mathbf{d}}^T\vec{\mathbf{y}}_u^{(0)}+\beta_4}{\beta_3}$, $a$, $c$, $q$, and
      $\lambda^{(0)} = H(\vec{\mathbf{y}}_u^{(0)})$
7     $(N_\epsilon,\ \{\lambda^{(i)}\}_{i=0}^{N_\epsilon},\ \vec{\mathbf{y}}_u^*,\ \lambda^*) = Dinkelbach(\mathcal{S},\ F(\vec{\mathbf{y}}_u),$
      $G(\vec{\mathbf{y}}_u), \vec{\mathbf{y}}_u^{(0)}, \epsilon)$ [QP Solver]
8     Discretise: $\vec{\mathbf{y}}^* = [\vec{\mathbf{1}}^T\ -b\vec{\mathbf{1}}^T\ \vec{\mathbf{y}}_u^{*T}]^T$
9 end
```

gain significant speed up by initializing the solution of each *Dinkelbach* iteration with the solution of the previous one. In the discretization step, we cluster the values of the relaxed solution $\vec{\mathbf{y}}_u^*$ to obtain the binary solution. Any clustering algorithm can be used here (e.g. k-means clustering with $k = 2$).

Beyond Binary Segmentation: DNCUT can be extended to multi-class ($C \geq 3$) segmentation in a recursive fashion, similar to what was done in Shi and Malik (2000), Malik et al. (2001). We first construct an over-segmentation of the image by clustering the values of $\vec{\mathbf{y}}_u^*$ into $k \gg C$ clusters. No particular choice of clustering algorithm is required here. In our experiments, we used both k-means clustering and mean shift clustering (Georgescu et al. 2003). Then, the k segments are greedily merged until only C segments remain. At each step, two clusters are merged, if the NCUT cost between them is the largest among all other pairs of clusters. This guarantees that the merged segments are the "most similar". Each resulting segment does not need to be spatially connected (i.e. spatially fragmented labels might occur). In our experiments, this simple merging method produced meaningful results; however, more elaborate merging schemes can be employed.

4.4 Special Case: DNCUT Under Linear Constraints

In this section, we describe how to efficiently apply the DNCUT framework under linear in/equality constraints. The reason we give this type of constraint special attention is twofold. (1) Linear constraints encode important first-order relationships between graph nodes, such as partial groupings (Yu and Shi 2004) or area constraints (Eriksson et al. 2007). However, current methods are restricted to linear equality constraints. In what follows, we show that DNCUT readily incorporates linear inequalities. (2) There exist efficient iterative methods that solve the underlying convex QP (e.g. interior point or active set methods).

As emphasized earlier, we need a single QP solver to find the global minima of the convex QPs in (8) and (5). We use a basic active set solver for the QP in (9).

$$\begin{aligned} \min \quad & \vec{\mathbf{x}}^T \mathbf{A}\vec{\mathbf{x}} + \vec{\mathbf{b}}^T \vec{\mathbf{x}} \\ \text{s.t.} \quad & \begin{cases} \mathbf{C}\vec{\mathbf{x}} \le \vec{\mathbf{d}}, \\ \mathbf{E}\vec{\mathbf{x}} = \vec{\mathbf{f}}, \end{cases} \end{aligned} \tag{9}$$

where $\mathbf{A} \in \mathbb{S}^{+}_{N_u}$, $\mathbf{C} \in \mathbb{R}^{|\mathrm{I}| \times N_u}$, $\mathbf{E} \in \mathbb{R}^{(|\mathrm{E}|+1) \times N_u}$, $\vec{\mathbf{b}} \in \mathbb{R}^{N_u}$, $\vec{\mathbf{d}} \in \mathbb{R}^{|\mathrm{I}|}$, and $\vec{\mathbf{f}} \in \mathbb{R}^{(|\mathrm{E}|+1)}$. Also, for our implementation purposes, we assume that C and E are sparse matrices (e.g. case of partial groupings). This assumption highly reduces the complexity of the active set method, whose fundamental step employs solving a large, sparse linear system with conjugate gradients. When $|\mathrm{I}| = 0$, there exists a closed form solution, which requires solving a pair of linear systems. Also, if $|\mathrm{E}| = 1$ and no hard constraints exist, the problem degenerates to the unconstrained NCUT problem. When hard constraints are applied, the problem becomes equivalent to interactive NCUT. More details of the optimization technique can be found in Appendix B.

5 Experimental Results

We conducted a set of image segmentation experiments to verify the correctness of our formulation and the proposed DNCUT algorithm. Using the NCUT criterion for image segmentation was chosen to only validate the DNCUT framework and to compare it against other NCUT algorithms. The segmentation performs only as well as the underlying NCUT formulation (i.e. our objective is only a more general NCUT framework and algorithm). The results demonstrate that our algorithm can accept hard or other linear constraints and efficiently derive a solution.

One problem that any image-as-a-graph approach must face is the need for large memory, to accommodate the large graphs created by the large number of pixels in any reasonably sized image. In particular, forming the weight matrix $\mathbf{W}_u$ is the memory bottle-neck for our DNCUT algorithm, as it is for the original NCUT implementation (Shi and Malik 2000). This problem may be partly addressed by a multiscale, coarse-to-fine implementation of DNCUT, similar to Cour et al. (2005); however, its extension to the DNCUT framework is left to future work. In our experiments, there was no need for down-sampling images, since the number of pixels in each image did not exceed the maximum allowable number of nodes. The edge weights between neighboring nodes are computed using intervening contours (Malik et al. 2001), applied to the grayscale version of the image. We use the default parameters (e.g. scale) that are available in the implementation of Malik et al. (2001). These weights take on values ranging from 0 to 1. $\mathbf{W}_u$ is made sparse by setting the weights between nodes lying farther than a certain distance (i.e. 10 pixels) from each other to zero. For an unknown node i, $\mathbf{p}_u^+(i)$ and $\mathbf{p}_u^-(i)$ are heuristically set to the maximum similarity between node i and all nodes of S_A and S_B respectively. In the absence of hard constraints (i.e. $N_+ = N_- = 1$), we set $\mathbf{p}_u^+(i) = \mathbf{p}_u^-(i) = p$. In fact, the DNCUT solution becomes equivalent to the traditional NCUT solution when $p \to 0$. For our experiments, we set $p = \frac{1}{2}$. Here, we note that if a probabilistic model exists for the graph nodes, $\mathbf{p}_u^+(i)$ and $\mathbf{p}_u^-(i)$ can be set to the likelihood that node i belongs to A and B, respectively. The tolerance values are: $\epsilon = 10^{-3}$ and $\varepsilon = 10^{-5}$.

In what follows, we give the complexity of the DNCUT algorithm (Sect. 5.1), compare the performance of DNCUT to two other NCUT algorithms when applied to the unconstrained binary NCUT problem (Sect. 5.2), and show segmentation results of DNCUT when applied to the linearly and nonlinearly constrained binary NCUT problem (Sect. 5.4) and to the unconstrained multi-class segmentation problem (Sect. 5.5).

5.1 Complexity

All our image segmentation experiments were executed using MATLAB 7.6 on a 2.4 GHz, 4 GB RAM PC. With hard and linear constraints, our algorithm required 1–3 *Dinkelbach* iterations to converge. In general, the worst case complexity of our algorithm is $\mathcal{O}(\mu_D \mu_A N_u^{3/2})$, where N_u is the total number of unknown nodes in $\mathcal{G}$ and $\mathcal{O}(N_u^{3/2})$ is the worst case complexity of solving a sparse linear system with N_u variables using conjugate gradients. μ_D is the maximum number of *Dinkelbach* iterations required for convergence and μ_A is the maximum number of active set iterations needed for one *Dinkelbach* iteration to converge. When the constraints are linear equalities, $\mu_A = 1$.

5.2 Validation

Here, we demonstrate the correctness of DNCUT by comparing it to two previous NCUT implementations (Shi and Malik 2000; Cour et al. 2005) applied to the same image. This is done quantitatively for the case of unconstrained NCUT (i.e. $|\mathrm{E}| = |\mathrm{I}| = 0$), where the global solution to the relaxed NCUT problem is known to be the second smallest generalized eigenvector (i.e. $\mathrm{eig}_2(\mathbf{D} - \mathbf{W}, \mathbf{D})$). In this case, $N_+ = N_- = 1$ (i.e. "source" and "sink" are only included). For all three algorithms, we explicitly apply the hard constraints by setting $\mathbf{y}_+ = 1$ and $\mathbf{y}_- = -b_0$.

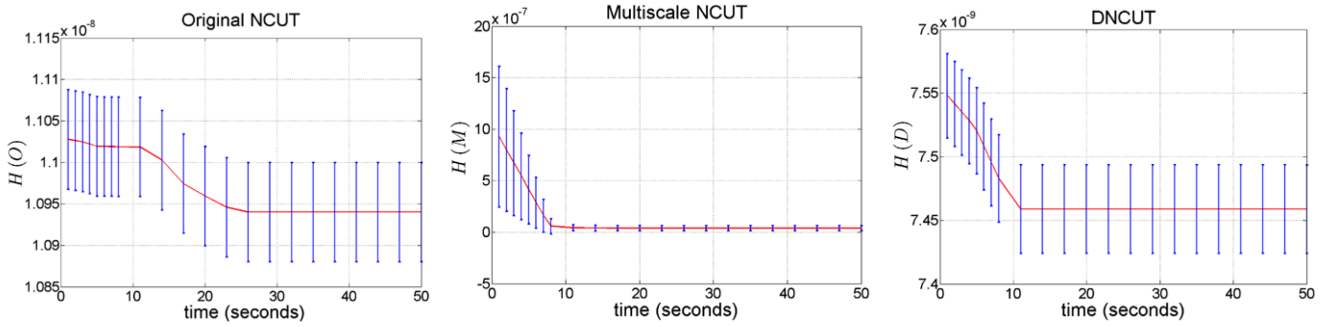

Fig. 5 (Color online) NCUT costs for the three NCUT algorithms are plotted versus run-time. The unconstrained NCUT problem is addressed here. The *red* values are averaged over all the images in the Berkeley segmentation dataset. The standard deviations are also plotted as *blue* bars

The NCUT methods we compare against are the original algorithm described in Shi and Malik (2000) and its multiscale version described in Cour et al. (2005). These algorithms use implicitly restarted Arnoldi/Lanczos methods for sparse matrices to perform eigen-decomposition in an iterative fashion (Lehoucq and Sorensen 1996). The run-time of this iterative decomposition is determined by the relative tolerance ϵ_{eig}. The iterations are terminated when the relative change in eigen solutions at the current and previous iterations is less than ϵ_{eig}. On the other hand, DNCUT solves the unconstrained problem by solving a pair of sparse linear systems. In fact, this pair of systems can be solved simultaneously, using the iterative conjugate gradient method. This is true since the conjugate directions for one of the systems can be used for the other. The run-time of DNCUT is determined by the relative tolerance ϵ_{CG}, which defines the stopping criterion for the conjugate gradient method. When the relative change in the solution to the linear system is less than ϵ_{CG}, the algorithm terminates. For all three algorithms, the run-times do not include the time needed to construct the weight matrix W_u.

In Fig. 5, we show comparative results for the three aforementioned algorithms when applied to images in the Berkeley segmentation dataset (Martin et al. 2001). Each image in this dataset has ∼155000 pixels/nodes. We aim to show how the NCUT cost (as defined in (4)) varies for each algorithm and for different run-times (i.e. for different stopping criteria). On each image in the dataset, we ran the three algorithms with varying stopping criteria and registered their corresponding NCUT costs. $H(\text{O})$, $H(\text{M})$, and $H(\text{D})$ denote the costs of the original NCUT (Shi and Malik 2000), multiscale NCUT (Cour et al. 2005), and DNCUT algorithms respectively. A total of 20 stopping criteria were used for each algorithm. Figure 5 plots the three NCUT costs at each run-time t. This cost is averaged over all images in the dataset. We also show the standard deviations of these measurements. Here, we used linear interpolation to complete the plots. It is obvious that as the run-time of an algorithm increases, its NCUT cost decreases till it reaches a stable value. All three algorithms exhibit this variation. From the plots, we conclude that the original algorithm outperforms the multiscale one by 10.9 dB, while the DNCUT algorithm outperforms the original one by 3.7 dB. This points to the obvious fact that solving (4) directly (i.e. NCUT on the augmented graph) is *not equivalent* to solving the NCUT problem on the *pixel* nodes alone. Moreover, as $p \to 0$, the original NCUT solution will approximate the DNCUT one and $H(D) \to H(O)$. Ideally, the comparison should not be a relative comparison between the three methods, but instead a comparison of each method with the global minimum of the original NCUT problem in (2). Since this problem is NP hard, obtaining its global minimum for non-trivially sized problems is infeasible.

We also provide four qualitative examples in Fig. 6. Columns (b)–(d) show the relaxed solutions (i.e. prior to any discretization) produced by the original, the multiscale, and the DNCUT algorithms on sample images in column (a), respectively. These solutions were obtained after the stable NCUT value for each algorithm was reached. Comparing the segmentation results, we see that the DNCUT solution is more detailed, which facilitates segmentation. If we consider the *bird* image in the second row, we see that the DNCUT solution plays the role of a soft labeling, where pixels with similar values are grouped together. So, the *sky* pixels are much darker than the foreground. This is not the case for the other two algorithms, as they do not utilize the augmented graph.

5.3 Computational Analysis

Here, we focus on a computational analysis of the three NCUT methods, when applied to the problem of unconstrained NCUT for the images in the Berkeley dataset. We study the relative change of the NCUT solution with run-time. At every run-time t, we calculate the relative solution change as: $\Delta e = \frac{\|\bar{\mathbf{x}}_{t+1} - \bar{\mathbf{x}}_t\|_2}{\|\bar{\mathbf{x}}_t\|_2}$. This change is averaged over all the images in the dataset. In Fig. 7, we plot Δe, as a percentage, for each algorithm. All three

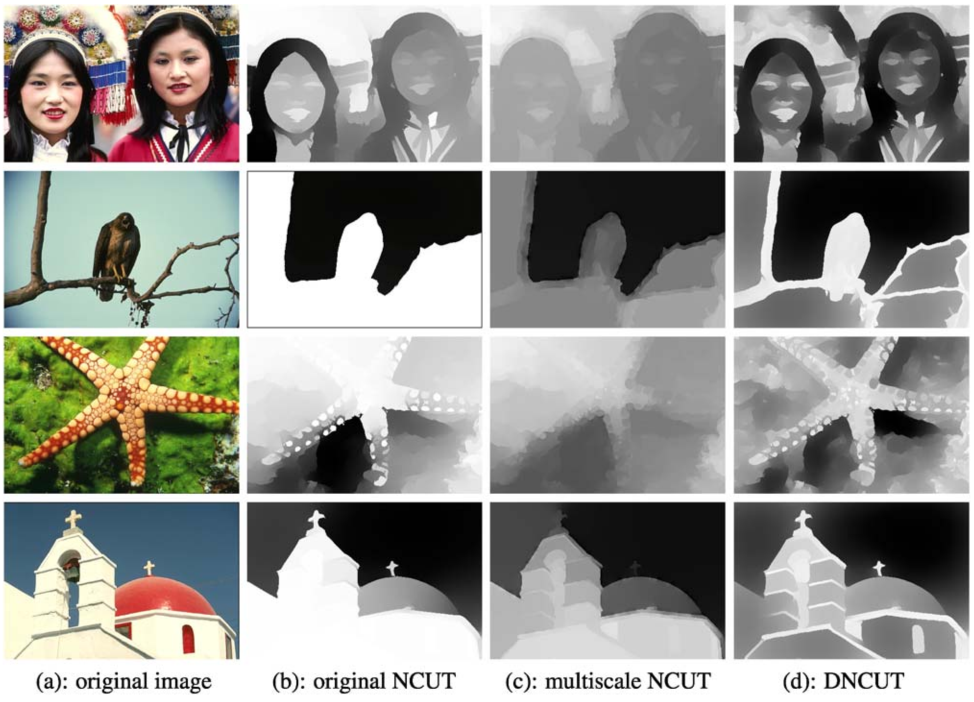

Fig. 6 Columns (**b**)–(**d**) show the stable NCUT solutions yielded by the three methods respectively, when applied to the images in column (**a**). The unconstrained NCUT problem is addressed here

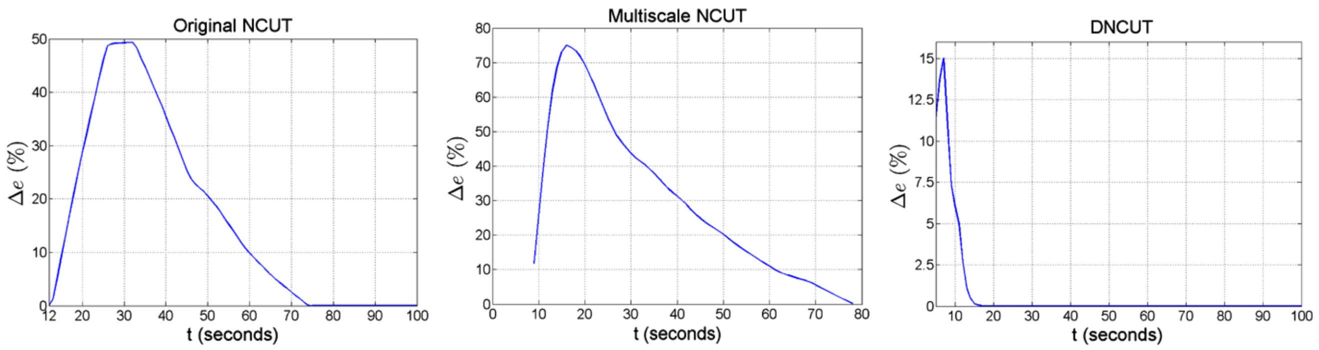

Fig. 7 Relative change of the NCUT solution with run-time

algorithms show the same type of variation. The initial monotonic increase in Δe is followed by a monotonic decrease till a stable solution is achieved. Moreover, DNCUT converges to a stable solution much quicker than the other two algorithms. For a 5% change, the original NCUT algorithm requires 65 seconds on average to stabilize, while the multiscale algorithm requires about 35 seconds. On the other hand, DNCUT requires about 11 seconds to reach a stable solution. The disparity between the three algorithms is due to the inherent computational nature of each algorithm. From these empirical results, we conclude that DNCUT (or equivalently the conjugate gradient algorithm) has significantly better convergence/stability properties than the original or multiscale NCUT algorithms (or equivalently the Arnoldi/Lanczos method for eigen-decomposition).

5.4 Interactive, Linearly Constrained, and Nonlinearly Constrained NCUT

Here, we conducted three experiments, whereby a different type of constraint on the nodes of the augmented graph is applied in each experiment. The first experiment addresses the problem of interactive NCUT. In Fig. 8, we consider the case of interactive NCUT, where a user marks the hard constraints on the displayed image in column (b) with *green* strokes defining S_A and *red* strokes defining S_B. Columns (c) and (d) show the segmentation results produced by DNCUT, with and without the hard constraints respectively. Column (e) shows the corresponding segmentations produced by the original NCUT algorithm without the hard constraints (i.e. unconstrained binary segmentation). It is evident that with additional (user-

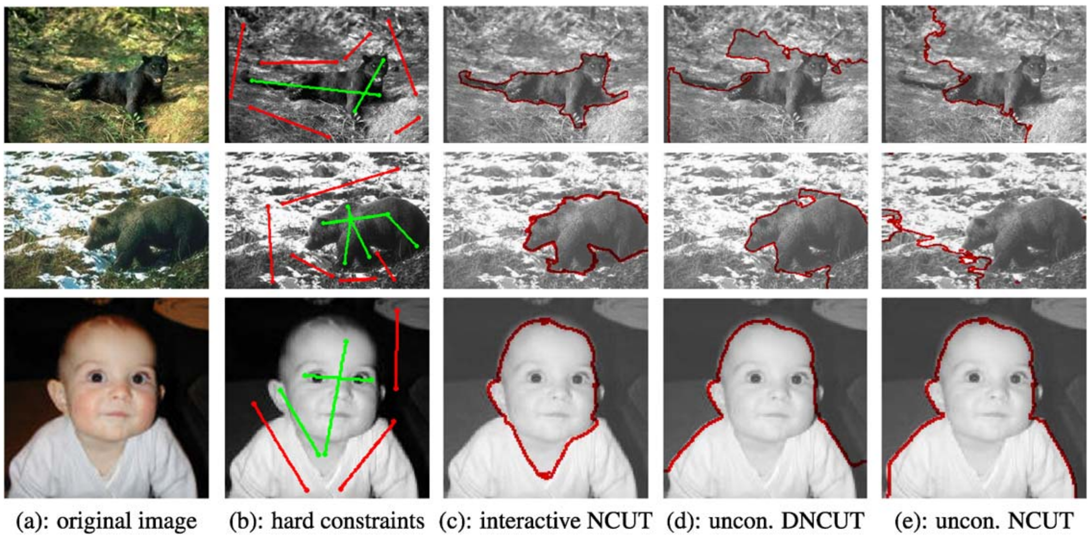

Fig. 8 (Color online) Shows examples of interactive NCUT, as compared to unconstrained NCUT and unconstrained DNCUT. The original images are shown in column (**a**). In column (**b**), we show the hard constraints of S_A (*green*) and S_B (*red*), as marked by a user. Using these hard constraints, column (**c**) shows the interactive segmentations produced by DNCUT. Columns (**d**) and (**e**) display the binary, unconstrained NCUT segmentations, produced by DNCUT and the original NCUT algorithm respectively. These last two columns are shown for comparison with the interactive DNCUT results

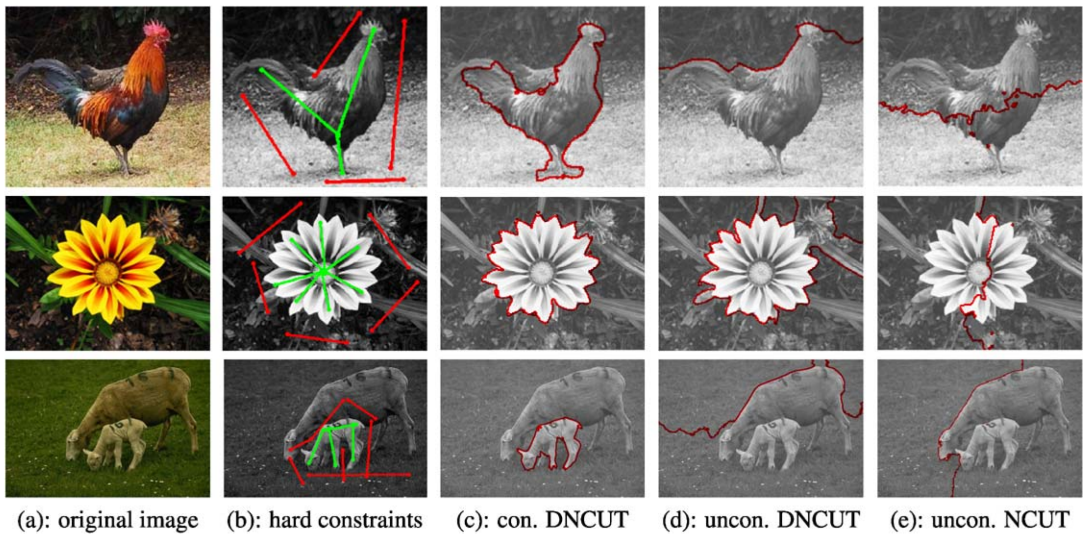

Fig. 9 (Color online) Shows examples of linearly constrained NCUT, as compared to unconstrained NCUT and unconstrained DNCUT. Two types of linear constraints are applied: partial pixel groupings and a box constraint on the values of the DNCUT solution (i.e. $-b_0\vec{\mathbf{1}} \leq \vec{\mathbf{y}}_u \leq \vec{\mathbf{1}}$). Partial groupings are marked by users in *red* and *green* strokes, as shown in column (**b**). Column (**c**) shows the binary DNCUT solutions to the linearly constrained NCUT problem. For visual comparison, columns (**d**) and (**e**) show the binary, unconstrained NCUT segmentations, produced by DNCUT and the original NCUT algorithm respectively

defined) constraints, the resulting binary (i.e foreground vs. background) segmentations become more perceptually relevant.

The second experiment addresses the problem of linearly constrained NCUT and shows some results in Fig. 9. In this experiment, we apply two types of linear constraints: (1) partial pixel groupings and (2) $-b_0\vec{\mathbf{1}} \leq \vec{\mathbf{y}}_u \leq \vec{\mathbf{1}}$ to produce binary segmentations shown in column (c). The box constraint (2) is a linear relaxation of the original discrete constraint in (2). Such linear inequalities cannot be handled by other NCUT algorithms. Unconstrained binary segmentations produced by DNCUT and the original NCUT algorithm are presented in columns (d) and (e) respectively. The partial groupings are determined by user defined strokes in column (b). The same colored pixels define nodes of the graph that should belong to the same segment.

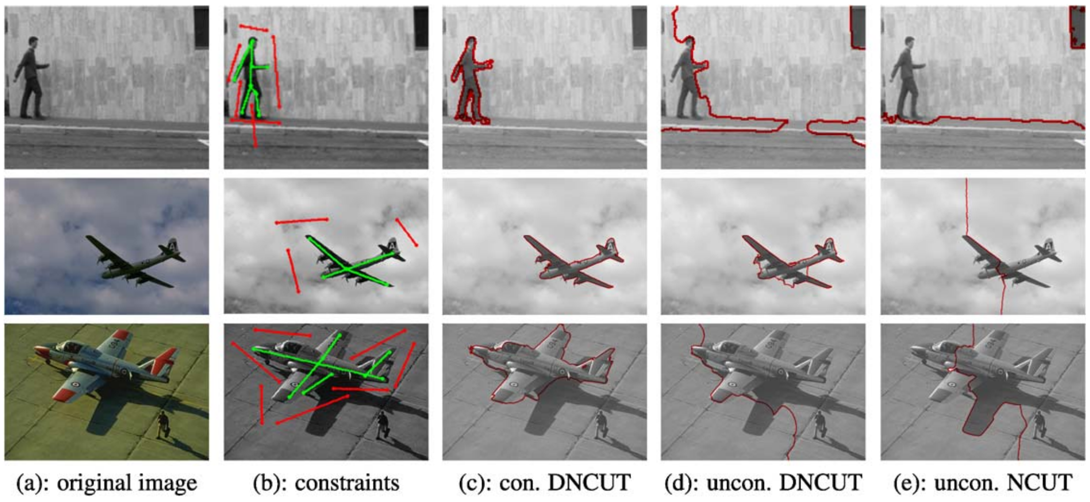

Fig. 10 (Color online) Shows examples of nonlinearly constrained NCUT, as compared to unconstrained NCUT and unconstrained DNCUT. Two types of constraints are applied: partial pixel groupings and a ball constraint on the values of the DNCUT solution (i.e. $\|\vec{\mathbf{y}}_u - (\frac{1-b_0}{2})\vec{\mathbf{1}}\|_2^2 \leq N_u(\frac{1+b_0}{2})^2$). Partial groupings are marked by users in *red* and *green* strokes, as shown in column (**b**). Column (**c**) shows the binary DNCUT solutions to the nonlinearly constrained NCUT problem. For visual comparison, columns (**d**) and (**e**) show the binary, unconstrained NCUT segmentations, produced by DNCUT and the original NCUT algorithm respectively

As in the second experiment, the third one also addresses the problem of partial groupings on the nodes of the augmented graph. However, in this version of the partial grouping problem, the box constraint on $\vec{\mathbf{y}}_u$ is replaced by a ball constraint (i.e. $\|\vec{\mathbf{y}}_u - (\frac{1-b_0}{2})\vec{\mathbf{1}}\|_2^2 \leq N_u(\frac{1+b_0}{2})^2$). This ball constraint is a relaxation on the box constraint, since it does not implicitly guarantee that $\mathbf{y}_u(i) \in [-b_0, 1]$ $\forall i = 1, \dots, N_u$. This is an example of how a nonlinear (quadratic) convex constraint can be applied to the DNCUT framework. We use an interior point (barrier) method to solve the DNCUT problem under these two constraints. Since there is a single nonlinear constraint, solving this version of the partial grouping problem is more efficient than the version in the previous experiment. Figure 10 shows some segmentation results on real images. The constrained DNCUT solutions (after discretization) are shown in column (c), while the unconstrained binary segmentations produced by DNCUT and the original NCUT algorithm are presented in columns (d) and (e) for comparison. The partial groupings are determined by user defined strokes in column (b). The same colored pixels define nodes of the graph that should belong to the same segment.

5.5 Multi-Class Segmentation

Figure 11 shows examples of unconstrained multi-class segmentation where $C = 2, 3, 4$. (b) shows the DNCUT solution to the unconstrained problem. The clustering and merging algorithm in Sect. 4.3 is used to produce the segmentations in (d), (f), and (h), where the corresponding boundaries are drawn in (c), (e) and (g). Even though this multi-class segmentation algorithm is suboptimal, it results in reasonable segmentations. However, its performance is correlated with the quality/detail of pixel groupings in the DNCUT solution. For example, in the *FLOWERS* case, the flowers are significantly delineated in the DNCUT solution; however, the lady bugs on the flower petals and the blades of grass in the out-of-focus background are not. This is primarily due to the nature of the weight matrix used. In fact, our experiments used intervening contours at a single scale and no color information was exploited. Incorporating more visual cues (e.g. color) into the edge weights and finding efficient ways to combine results at multiple scales are left for future work.

Next, we show how the low-level, multi-class segmentation results for the three algorithms relate to human segmentation results. Here, we use the Berkeley segmentation dataset, which contains multiple benchmark (human) segmentations for each image in the dataset. To each image, we applied the three multi-class algorithms, where the number of classes was set to the number of segments labeled in the benchmark segmentations corresponding to this image. Then, the resulting (binary) segment boundaries are averaged over all the benchmark segmentations. We used the evaluation kit of Martin et al. (2001) to plot the precision-recall curves shown in Fig. 12. The three algorithms yield very similar F-scores, which are higher than random performance (0.41) and significantly less than human performance (0.79). They rank low on the list of state-of-the-art segmentation algorithms. More importantly, comparing these NCUT algorithms together, we see that the multiscale one has the worst F-score (0.46), while the original one has the

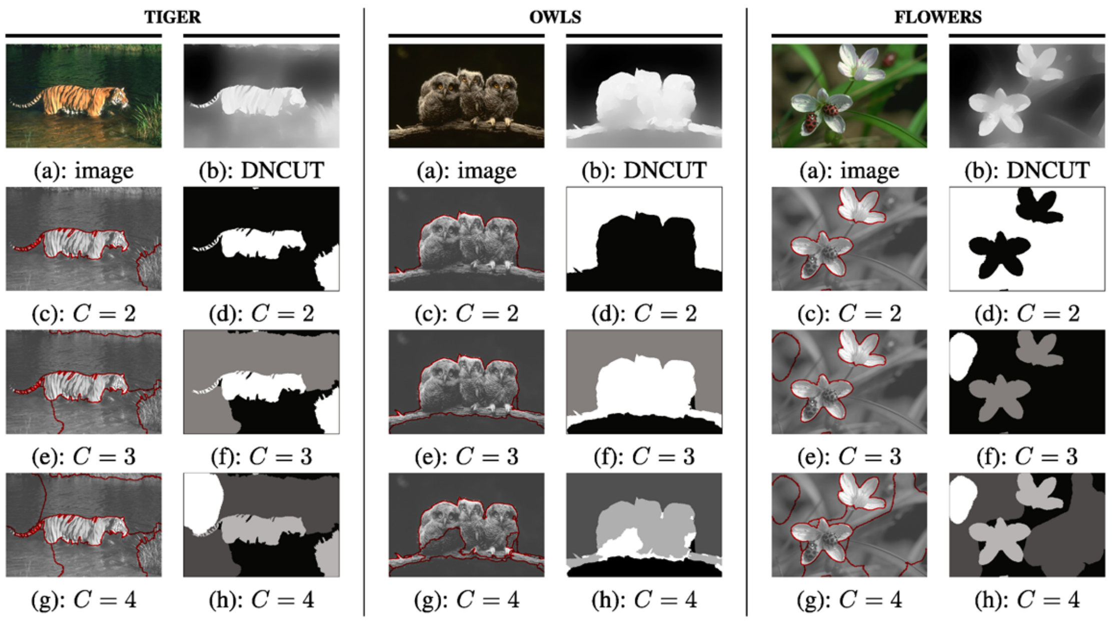

Fig. 11 Examples of unconstrained multi-class segmentation with $C = 2, 3, 4$, when applied to three images: *TIGER*, *OWLS*, and *FLOWERS*. (**b**) Shows the DNCUT solution to the unconstrained problem. (**c**), (**e**), and (**g**) Show the boundaries of the labeled segments in (**d**), (**f**), and (**h**) respectively. We refer the readers to vision.ai.uiuc.edu/~bghanem2/Shared/DNCUT/supplementary.zip for all the segmentation results

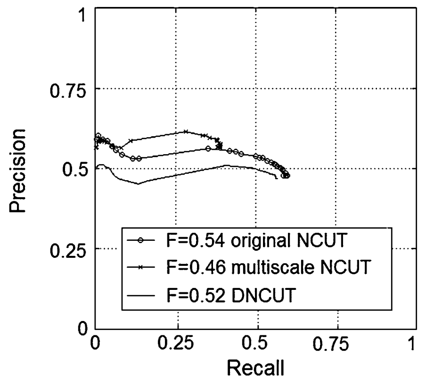

Fig. 12 Precision-Recall curves for the three NCUT algorithms, when applied to the Berkeley human segmentation dataset

best F-score (0.54). DNCUT registers an F-score (0.52) that is very similar to the original algorithm. This is the case, even though DNCUT was shown (in Sect. 5.2) to yield a smaller NCUT cost than the other algorithms. This discrepancy points to the fact that the NCUT formulation for image segmentation *does not* correlate well with human segmentation.

6 Conclusions and Future Work

We have presented a unifying DNCUT framework for solving convexly constrained NCUT problems with data priors on the augmented graph nodes. We avoid using traditional eigen-decomposition, due to its restrictions on the types of constraints it can handle and its computational complexity. In this framework, any convexly constrained NCUT problem can be converted into a sequence of convex QP's. In the case of linear constraints, we propose an algorithm to efficiently find the global solution of each QP. To validate the correctness of DNCUT, we compare it to state-of-the-art NCUT algorithms. As compared to these algorithms, DNCUT provides a better and more computationally efficient solution. We also show results of binary segmentation under hard and linear constraints, in addition to multi-class segmentation results. In the future, we plan to develop a multiscale version of this framework to handle larger graphs and to incorporate grouping information from weight matrices computed at different scales. Furthermore, we plan to improve the multi-class segmentation extension.

Acknowledgement The support of the Office of Naval Research under grant N00014-09-1-0017 and the National Science Foundation under grant IIS 08-12188 are gratefully acknowledged.

Appendix A: Dinkelbach Initialization for (4) (Sect. 4.2)

In this part, we will give a more detailed description of $b_0 = \arg\max_{b \geq 0} r(b)$. We expand $r(b)$ as in (10), where

$$\tau = \frac{N_-(\vec{\mathbf{1}}^T\overrightarrow{\mathbf{p}_u^+})}{N_+(\vec{\mathbf{1}}^T\overrightarrow{\mathbf{p}_u^-})}.$$

$$\begin{aligned} r(b) &= \frac{-\alpha[(N_+)^2+(bN_-)^2][N_+(\vec{\mathbf{1}}^T\overrightarrow{\mathbf{p}_u^+}) - bN_-(\vec{\mathbf{1}}^T\overrightarrow{\mathbf{p}_u^-})]}{[(N_+)^2 - b(N_-)^2]} \\ &\quad - (1-\alpha)\big[N_+\big(\vec{\mathbf{1}}^T\overrightarrow{\mathbf{p}_u^+}\big) + b^2N_-\big(\vec{\mathbf{1}}^T\overrightarrow{\mathbf{p}_u^-}\big)\big] \\ &= \frac{\alpha b[b+1][\frac{(\vec{\mathbf{1}}^T\overrightarrow{\mathbf{p}_u^-})N_+^2}{N_-}][1-\tau]}{(\frac{N_+}{N_-})^2 - b} \\ &\quad - \big[N_+\big(\vec{1}^T\overrightarrow{\mathbf{p}_u^+}\big) + b^2N_-\big(\vec{\mathbf{1}}^T\overrightarrow{\mathbf{p}_u^-}\big)\big]. \end{aligned} \tag{10}$$

Since $\alpha, b, \vec{\mathbf{1}}^T\overrightarrow{\mathbf{p}_u^-}, N_+, N_-$, and τ are non-negative numbers, we deduce that $r(b)$ can be arbitrarily large by setting $b = b_0 = \kappa(\frac{N_+}{N_-})^2$, where κ is chosen such that,

$$\begin{cases} \kappa = 1-\varepsilon, & \text{if } \tau \le 1, \\ \kappa = 1+\varepsilon, & \text{if } \tau > 1, \end{cases} \quad \text{where} \quad \begin{cases} \varepsilon \ge 0, \quad \varepsilon \to 0, \\ \tau = \frac{N_-(\vec{1}^T\overrightarrow{\mathbf{p}_u^+})}{N_+(\vec{\mathbf{1}}^T\overrightarrow{\mathbf{p}_u^-})}. \end{cases}$$

Appendix B: Solving a Convex QP with Linear Constraints (Sect. 4.4)

In the next three parts, we will describe how to solve a convex quadratic programming problem with general linear constraints, of the following form:

$$\begin{aligned} \min \quad & \vec{\mathbf{x}}^T\mathbf{A}\vec{\mathbf{x}} + \vec{\mathbf{b}}^T\vec{\mathbf{x}} \\ \text{s.t.} \quad & \begin{cases} \mathbf{C}\vec{\mathbf{x}} \le \vec{\mathbf{d}}, \\ \mathbf{E}\vec{\mathbf{x}} = \vec{\mathbf{f}}. \end{cases} \end{aligned} \tag{11}$$

Next, we consider the case when only a single equality constraint exists (e.g. in the case of unconstrained or interactive NCUT). Then, we consider the case of multiple equality constraints followed by general linear constraints (i.e. in/equalities).

Solving a Convex QP with a Single Linear Equality Constraint

Here, we will provide the closed form solution ($\vec{\mathbf{x}}^*$) to the following convex QP problem.

$$\begin{aligned} \min \quad & [\vec{\mathbf{x}}^T\mathbf{A}\vec{\mathbf{x}} + \vec{\mathbf{b}}^T\vec{\mathbf{x}}] \\ \text{s.t.} \quad & \vec{\mathbf{e}}^T\vec{\mathbf{x}} + f = 0. \end{aligned} \tag{12}$$

We derive the Lagrangian dual of (12), $\mathcal{L}(\vec{\mathbf{x}}, \nu)$, and determine the primal-dual solution, $\vec{\mathbf{x}}^*$ and ν^*, in closed form. Equations (14), (15) evaluate these optimal solutions. Note, that the closed form solution of the primal problem is valid, since $\mathcal{L}(\vec{\mathbf{x}}, \nu)$ is convex in $\vec{\mathbf{x}}$. Also, the form of ν^* directly follows from the concavity of $\mathcal{L}(\vec{\mathbf{x}}^*, \nu)$. The optimal primal and dual solutions can be obtained efficiently by solving a pair of linear systems: $\mathbf{A}\vec{\mathbf{x}} = \vec{\mathbf{b}}$ and $\mathbf{A}\vec{\mathbf{x}} = \vec{\mathbf{e}}$, using preconditioned conjugate gradients.

$$\mathcal{L}(\vec{\mathbf{x}}, \nu) = \vec{\mathbf{x}}^T\mathbf{A}\vec{\mathbf{x}} + (\vec{\mathbf{b}} + \nu\vec{\mathbf{e}})^T\vec{\mathbf{x}} + \nu f. \tag{13}$$

Primal Solution:

$$\vec{\mathbf{x}}^* = \arg\min_{\vec{\mathbf{x}}} \mathcal{L}(\vec{\mathbf{x}}, \nu) = -\frac{1}{2}\mathbf{A}^{-1}[\vec{\mathbf{b}} + \nu^*\vec{\mathbf{e}}]. \tag{14}$$

Dual Solution:

$$\nu^* = \arg\max_{\nu} \mathcal{L}(\vec{\mathbf{x}}^*, \nu) = \frac{2f - \vec{\mathbf{e}}^T\mathbf{A}^{-1}\vec{\mathbf{b}}}{\vec{\mathbf{e}}^T\mathbf{A}^{-1}\vec{\mathbf{e}}}. \tag{15}$$

Solving a Convex QP with Multiple Linear Equality Constraints

Here, we will provide the closed form solution ($\vec{\mathbf{x}}^*$) to the following convex QP problem.

$$\begin{aligned} \min \quad & [\vec{\mathbf{x}}^T\mathbf{A}\vec{\mathbf{x}} + \vec{\mathbf{b}}^T\vec{\mathbf{x}}] \\ \text{s.t.} \quad & \mathbf{E}\vec{\mathbf{x}} + \vec{\mathbf{f}} = 0. \end{aligned}$$

Similar to the previous part, we solve the primal dual problems by solving the linear system in (16). However, in this case, the primal and dual variables are coupled in the same linear system.

$$\begin{bmatrix} 2\mathbf{A} & \mathbf{E}^T \\ \mathbf{E} & \vec{\mathbf{0}} \end{bmatrix} \begin{bmatrix} \vec{\mathbf{x}}^* \\ \vec{\nu}^* \end{bmatrix} = -\begin{bmatrix} \vec{\mathbf{b}} \\ \vec{\mathbf{f}} \end{bmatrix}. \tag{16}$$

Solving a Convex QP with General Linear Constraints

Here, we will discuss an active set based method that iteratively solves the following convex quadratic programming problem. Let I be the set of inequality constraints.

$$\begin{aligned} \min \quad & \vec{\mathbf{x}}^T\mathbf{A}\vec{\mathbf{x}} + \vec{\mathbf{b}}^T\vec{\mathbf{x}} \\ \text{s.t.} \quad & \begin{cases} \mathbf{C}\vec{\mathbf{x}} \le \vec{\mathbf{d}}, \\ \mathbf{E}\vec{\mathbf{x}} = \vec{\mathbf{f}}. \end{cases} \end{aligned}$$

We present the main steps involved in applying the active set method.

1. Initialize a counter $k = 0$. Choose a feasible initial guess $\vec{\mathbf{x}}^{(k)}$, which satisfies a certain set $S_E^{(k)}$ of the constraints with equality, including all the equality constraints and a subset of the inequality constraints. These constraints are referred to as active constraints. They correspond to the following set of linear equalities: $\mathbf{G}^{(k)}\vec{\mathbf{x}} = \vec{\mathbf{h}}^{(k)}$.
2. Compute a step direction $\vec{\delta}^{(k)}$, by solving a convex QP with linear equality constraints corresponding to those of

$S_E^{(k)}$. This is done using the method outlined in the previous part.

$$\vec{\delta}^{(k)} = \arg\min \vec{\mathbf{z}}^T \mathbf{A}\vec{\mathbf{z}} + \left(\vec{\mathbf{b}} + 2\mathbf{A}\vec{\mathbf{x}}^{(k)}\right)^T \vec{\mathbf{z}}$$
$$\text{s.t.} \quad \mathbf{G}^{(k)}\vec{\mathbf{z}} = \vec{\mathbf{0}}.$$

3. If $\vec{\delta}^{(k)} = \vec{\mathbf{0}}$, then check the Lagrangian multipliers corresponding to the active inequality constraints. If they are all positive, then the final solution has been obtained; otherwise, remove the constraints corresponding to the negative multipliers from $S_E^{(k)}$. On the other hand, if $\vec{\delta}^{(k)} \neq \vec{\mathbf{0}}$, take a step from the previous solution along $\vec{\delta}^{(k)}$ (i.e. $\vec{\mathbf{x}}^{(k)} = \vec{\mathbf{x}}^{(k+1)} + \alpha^{(k)}\vec{\delta}^{(k)}$). The step size is computed as follows, where $\vec{\mathbf{g}}_i^{(k)}$ is the ith row of $\mathbf{G}^{(k)}$ and $\mathbf{h}(i)^{(k)}$ is the ith element of $\vec{\mathbf{h}}^{(k)}$.

$$\alpha^{(k)} = \min\left(1, \frac{\mathbf{h}(p)^{(k)} - \vec{\mathbf{g}}_p^{(k)T}\vec{\mathbf{x}}^{(k)}}{\vec{\mathbf{g}}_p^{(k)T}\vec{\delta}^{(k)}}\right)$$

$$\text{where } p = \underset{\substack{i\in S_E^{(k)}\cap I \\ \vec{\mathbf{g}}_i^{(k)T}\vec{\delta}^{(k)}>0}}{\arg\min} \left[\frac{\mathbf{h}(i)^{(k)} - \vec{\mathbf{g}}_i^{(k)T}\vec{\mathbf{x}}^{(k)}}{\vec{\mathbf{g}}_i^{(k)T}\vec{\delta}^{(k)}}\right].$$

This step value guarantees that the current solution lies on the set of active constraints. If $\alpha^{(k)} < 1$, then add the pth constraint to $S_E^{(k)}$. Increment k.

4. Iterate through steps 2–3 until convergence.

References

Bhatia, R. (1997). *Matrix analysis*. Berlin: Springer.

Boykov, Y. Y., & Jolly, M. P. (2001). Interactive graph cuts for optimal boundary & region segmentation of objects in n-d images. In *International conference on computer vision* (Vol. 1, pp. 105–112).

Boykov, Y., & Funka-Lea, G. (2006). Graph cuts and efficient n-d image segmentation. *International Journal of Computer Vision*, *70*(2), 109–131.

Boykov, Y., Veksler, O., & Zabih, R. (2001). Fast approximate energy minimization via graph cuts. *Transactions on Pattern Analysis and Machine Intelligence*, *23*(11), 1222–1239.

Cour, T., & Shi, J. (2007). Solving Markov random fields with spectral relaxation. In *International conference on artificial intelligence and statistics* (Vol. 11).

Cour, T., Benezit, F., & Shi, J. (2005). Spectral segmentation with multiscale graph decomposition. In *International conference on computer vision and pattern recognition* (pp. 1124–1131).

Dinkelbach, W. (1967). On nonlinear fractional programming. *Management Science*, *13*, 492–498.

Eriksson, A. P., Olsson, C., & Kahl, F. (2007). Normalized cuts revisited: a reformulation for segmentation with linear grouping constraints. In *International conference on computer vision*.

Georgescu, B., Shimshoni, I., & Meer, P. (2003). Mean shift based clustering in high dimensions: a texture classification example. In *International conference on computer vision* (Vol. 1, pp. 456–463).

Greig, D., Porteous, B., & Seheult, A. (1989). Exact maximum a posteriori estimation for binary images. *Journal of the Royal Statistical Society*, *51*, 271–279.

Kolmogorov, V., Boykov, Y., & Rother, C. (2007). Applications of parametric maxflow in computer vision. In *International conference on computer vision*.

Lehoucq, R. B., & Sorensen, D. C. (1996). Deflation techniques for an implicitly restarted Arnoldi iteration. *SIAM Journal on Matrix Analysis and Applications*, *17*, 789–821.

Li, Y., Sun, J., Tang, C. K., & Shum, H. Y. (2004). Lazy snapping. In *SIGGRAPH'04: ACM SIGGRAPH 2004 papers*, New York, NY, USA (pp. 303–308). New York: ACM.

Malik, J., Belongie, S., Leung, T. K., & Shi, J. (2001). Contour and texture analysis for image segmentation. *International Journal of Computer Vision*, *43*(1), 7–27.

Martin, D., Fowlkes, C., Tal, D., & Malik, J. (2001). A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *International conference on computer vision* (pp. 416–423).

Paragios, N., Chen, Y., & Faugeras, O. (2006). *The handbook of mathematical models in computer vision* (pp. 100–119). Berlin: Springer.

Pardalos, P. M., & Vavasis, S. A. (2004). Quadratic programming with one negative eigenvalue is NP-hard. *Journal of Global Optimization*, *1*(1), 15–22.

Pothen, A., Simon, H. D., & Liou, K. P. (1990). Partitioning sparse matrices with eigenvectors of graphs. *SIAM Journal on Matrix Analysis and Applications*, *11*, 430–452.

Rodenas, R., Lopez, M., & Verastegui, D. (1999). Extensions of Dinkelbach's algorithm for solving non-linear fractional programming problems. *TOP: Journal of the Spanish Society of Statistics and Operations Research*, *7*(1), 33–70.

Rother, C., Kolmogorov, V., & Blake, A. (2004). "GrabCut": interactive foreground extraction using iterated graph cuts. *ACM Transactions on Graphics*, *23*(3), 309–314.

Shi, J., & Malik, J. (2000). Normalized cuts and image segmentation. *Transactions on Pattern Analysis and Machine Intelligence*, *22*, 888–905.

Thompson, R. C., & Freede, L. J. (1970). Eigenvalues of partitioned hermitian matrices. *Bulletin of the Australian Mathematical Society*, *3*, 23–37.

Weiss, Y. (1999). Segmentation using eigenvectors: a unifying view. In *International conference on computer vision* (pp. 975–982).

Yu, S., & Shi, J. (2004). Segmentation given partial grouping constraints. *Transactions on Pattern Analysis and Machine Intelligence*, *2*(26), 173–183.