Abstract

Recently, unsupervised image segmentation has become increasingly popular. Starting from a superpixel segmentation, an edge-weighted region adjacency graph is constructed. Amongst all segmentations of the graph, the one which best conforms to the given image evidence, as measured by the sum of cut edge weights, is chosen.

Since this problem is NP-hard, we propose a new approximate solver based on the move-making paradigm: first, the graph is recursively partitioned into small regions (cut phase). Then, for any two adjacent regions, we consider alternative cuts of these two regions defining possible moves (glue & cut phase). For planar problems, the optimal move can be found, whereas for non-planar problems, efficient approximations exist.

We evaluate our algorithm on published and new benchmark datasets, which we make available here. The proposed algorithm finds segmentations that, as measured by a loss function, are as close to the ground-truth as the global optimum found by exact solvers. It does so significantly faster then existing approximate methods, which is important for large-scale problems.

1. Introduction

Segmentation is an important problem in computer vision as a first step towards understanding an image. Many algorithms start with an over-segmentation into superpixels, which are then clustered into “perceptually meaningful” regions. Usually, the number of these regions is not known beforehand.

Recently, the multicut formulation [9] (sometimes called correlation clustering, [8]) has become increasingly popular for unsupervised image segmentation. Given an edge-weighted region adjacency graph, the problem is to find the segmentation which minimizes the cost of the cut edges. Such an approach has been shown to yield state-of-the-art results on the Berkeley Segmentation Database [4, 23, 2].

Un fortunately, solving the multicut problem is in general NP-hard. Any exact solver will therefore be plagued by scalability issues. However, segmentation of images with ever finer superpixel partitionings and of large volume images in computational neuroscience [5] demand solutions to large scale multicut problems. In this regime of large-scale problems, existing solvers either fail to find any solution after a reasonable time, rely on suitable edge weights (such that the problem decomposes naturally into independent subproblems) or yield an approximate solution possibly far away from the optimum.

Contribution. In this work we consider (i) a new perspective on solving the multicut problem by local move making methods together with (ii) a new approximate multicut solver called Cut, Glue & Cut (CGC). Furthermore, (iii) our method avoids re-solving the same moves by tracking their “dirtyness”, which decreases the runtime. (iv) An extensive evaluation on existing and new benchmark datasets shows that CGC provides results close to optimality and equal application performance significantly faster than all its competitors, both exact and approximative methods, and gives new insights concerning the applicability of competing methods. (v) Our C++ implementation of CGC is part of the open-source OpenGM inference library [3].

Organization. In Sec. 2 we review two formulations of the multicut problem. After discussing related work (Sec. 3), we review the max-cut problem (Sec. 4). Based on a general formulation of local partition moves on segmentations in Sec. 5, we describe our new Cut, Glue & Cut algorithm in Sec. 6. Extensive experiments on benchmark datasets are presented in Sec. 7. Finally, we discuss our findings.

2. Problem Formulation

Let $G = (V, E, w)$ be a weighted region adjacency graph of nodes $V$, representing superpixels, and edges $E$. The function $w : E \rightarrow \mathbb{R}$ assigns a weight to each edge. A positive weight expresses the desire that two adjacent nodes should be merged, whereas a negative weight indicates that these nodes should be separated into two different regions.

A subgraph $G_A = \{A, E_A, w\}$ consists of nodes $A \subseteq V$ and edges $E_A \subseteq E$.
V and edges $E_A := E \cap (A \times A)$. We call a connected component of nodes V a region $R \subseteq V$. The edges between $A$ and $B$ form the set $E_{A-B} := E \cap (A \times B)$. We use the shorthand $\overline{A} := V \setminus A$ for the complement of $A$.

Each node $i$ is assigned a label $L_i \in \{0, \ldots, |V| - 1\}$. Using the indicator function $\delta(\cdot)$, the multicut problem can be written as a node labeling problem [7]:

$$\arg\min_{L} \left\{ \sum_{e=(i,j) \in E} w(e) \cdot \delta(L_i \neq L_j) \right\}, \quad (1)$$

where $\delta(a) = 1$ if $a$ is true and 0 else. While (1) looks like a common Potts energy without unary terms, there are some crucial differences:

First, any weight $w(e \in E)$ can be positive or negative. Second, the lack of any unary data terms renders the problem much harder and introduces ambiguity. Third, the label space is large. In general we have to set $|L| = |V|$ to allow for the solution where every supervoxel becomes its own region without having to solve the graph coloring problem implicitly. For planar graphs, it would be sufficient that $L_i \in \{0, \ldots, 3\}$ because any planar-graph is 4-colorable [6].

An alternative formulation of problem (1) is in terms of binary edge indicator variables $y \in \{0, 1\}^{|E|}$:

$$\arg\min_{y} \left\{ \sum_{e=(i,j) \in E} w(e) \cdot y_{e} \right\} \text{ s.t. } y \in \text{MC}_G. \quad (2)$$

A segmentation $y$ has an associated energy which is denoted by $\text{CUT}_G(y)$. $\text{MC}_G$ is the set of all valid multicus of $G$; its convex hull forms the so called multicut polytope [9]. Restricting $y$ to $\text{MC}_G$ ensures that the labeling is consistent: if $y_{ij} = 1$ then nodes $i$ and $j$ should be in separate regions even after connected component labeling. Intuitively, dangling line segments are forbidden in two dimensional images, and punctured walls or faces are ruled out in three dimensional images.

While any segmentation $\bigcup_i R_i \equiv V$ is represented by exactly one edge labeling $y \in \text{MC}_G$, many labelings $L$ represent the same segmentation. Therefore, an efficient handling of the multicut polytope is essential.

3. Related Work

Applying state-of-the-art solvers for the labeling problem (1) is challenging since any permutation of the labelings transforms an optimal solution into another optimal solution, see [15] for more details and a recent review.

In general, $y \in \text{MC}_G$ can be enforced by an exponential number of constraints [9], but in practice – for a given objective function – a small subset of those are sufficient. Therefore, a major branch of research has focused on cutting plane approaches, either by solving a relaxation of (2) by a sequence of linear programs [17, 18, 15, 11] or by solving problem (2) exactly by a sequence of integer linear programs [14, 4, 5, 15, 1, 2].

For the latter, one can avoid the cutting plane procedure by lifting to the fully connected graph at the expense of problem size [1]. For many image segmentation problems, however, the size of the complete graph is prohibitively large.

Alush and Goldberger [1, 2] suggest to use partial optimality in order to decompose the problem into its positively connected components, defined as the connected components of the graph $G' = (V, E')$ with $E' = \{e \in E | w_e > 0\}$. These problems are then separately solved to global optimality. While this recovers the true solution, it depends on the weights $w$ whether a significant reduction of the problem is possible.

Yarkony et al. [23] suggest a dual column-generating method, called PlanarCC, which solves a relaxed linear program by iteratively solving weighted two-coloring problems. Because these sub-problems require planarity to be tractable, PlanarCC is restricted to problems with planar structure.

Another branch of research has focused on specialized move-making methods [16, 24, 7] that take the degeneracy of the solution, due to label permutations in (1), into account. Move-making algorithms maintain a valid solution (segmentation defined via $L$ or, equivalently, $y \in \text{MC}$) throughout the optimization procedure. In each step, a set of possible moves transforming the current segmentation is considered, and the move which realizes the maximal energy decrease is chosen. Therefore, the energy decreases monotonically until a (local) optimum is found. The Kerninghan-Lin method [16], used in circuit-layout design, applies a sequence of greedy local moves to neighboring regions. Bagon [7] recently proposed an extension to the $\alpha$-expansion move-making algorithm for solving multicut problems, which handles the large label space and label ambiguity in (1) by using dynamic label sets and additional label fixing, respectively.

4. Max-Cut

The max-cut problem [10] (also known as the weighted 2-coloring problem) is a basic subproblem which has to be solved in the Kerninghan-Lin method [16], Expand-and-Explore [7], PlanarCC [23] and also our method.

The max-cut problem\(^1\) is the specialization of (1) for the

\(^1\)For reasons of consistency, we consider wlog. minimization instead of maximization.
This should not be confused with the min-cut problem, which refers to sub-modular “graphcut problems” with non-negative edge weights \( w \), for which a max-flow problem can be formulated [19]. Intuitively, restricting the label space to two labels (0 and 1) simplifies the problem. For planar graphs, (3) can be solved by the Blossom algorithm [22] in polynomial time. In general this problem is still NP-hard.

In our experiments, we found that using QPBO-I performs better than the greedy method used in the KL algorithm [7]. Recently, Berg and Lin have suggested a greedy method for (3) used in the KL algorithm [16]. Recently, Berg and Lin suggested to use QPBO-I [21] for (3). While this LP relaxation often gives non-integral solutions for many nodes, the improving extension (i) [21] iteratively fixes such nodes and therefore also deals with the ambiguity of the problem.

In our experiments, we found that using QPBO-I performs better than the greedy method used in the KL algorithm, but slightly worse than optimal solvers (Sec. 7).

5. Partition Moves

In this section we define a class of moves which can be used to iteratively improve segmentations. The Partition Move Theorem shows that under some technical conditions, the improvement of the local sub-problem leads to monotonous improvement of the global energy.

**Theorem: Partition Moves.** Let \( y^t \in MC_G \) represent the segmentation of \( G \) at step \( t \). Local partition moves are defined over a subset \( S \subseteq V \) for which all edges crossing its

**Case of binary node labels**

\[
\arg\min_{L \in \{0,1\}^{|\mathcal{E}|}} \left\{ \sum_{e=(i,j) \in \mathcal{E}} w(e) \cdot \delta(L_i \neq L_j) \right\} \quad (3)
\]

This should not be confused with the min-cut problem, which refers to sub-modular “graphcut problems” with non-negative edge weights \( w \), for which a max-flow problem can be formulated [19]. Intuitively, restricting the label space to two labels (0 and 1) simplifies the problem. For planar graphs, (3) can be solved by the Blossom algorithm [22] in polynomial time. In general this problem is still NP-hard.

In response, Kernighan and Lin have suggested a greedy method for (3) used in the KL algorithm [16]. Recently, Berg and Lin suggested to use QPBO-I [21] for (3). While this LP relaxation often gives non-integral solutions for many nodes, the improving extension (i) [21] iteratively fixes such nodes and therefore also deals with the ambiguity of the problem.

In our experiments, we found that using QPBO-I performs better than the greedy method used in the KL algorithm, but slightly worse than optimal solvers (Sec. 7).

5. Partition Moves

In this section we define a class of moves which can be used to iteratively improve segmentations. The Partition Move Theorem shows that under some technical conditions, the improvement of the local sub-problem leads to monotonous improvement of the global energy.

**Theorem: Partition Moves.** Let \( y^t \in MC_G \) represent the segmentation of \( G \) at step \( t \). Local partition moves are defined over a subset \( S \subseteq V \) for which all edges crossing its

\[
\begin{align*}
\text{end of cut phase} & \quad (5 \text{ sub-problems}) \\
\text{cut & glue phase} \quad \text{work on pair (3,4)} \\
\text{cut & glue phase} \quad \text{pair (3,4) result}
\end{align*}
\]

Repeat steps for all pairs of adjacent sub-problems

**5. Partition Moves**

In this section we define a class of moves which can be used to iteratively improve segmentations. The Partition Move Theorem shows that under some technical conditions, the improvement of the local sub-problem leads to monotonous improvement of the global energy.

**Theorem: Partition Moves.** Let \( y^t \in MC_G \) represent the segmentation of \( G \) at step \( t \). Local partition moves are defined over a subset \( S \subseteq V \) for which all edges crossing its

Borders have to be labeled one: \( \forall e \in E_{G - S} : y^t_e = 1 \).

Let \( G_S \) be the respective subgraph of \( G \) and \( \hat{y} \in MC_{G_S} \) represent a given valid segmentation of \( S \). For the combined segmentation

\[
y^{t+1} := \begin{cases} 
\hat{y}_e & e \in E_S \\
y^t_e & \text{else}
\end{cases}
\]

the following holds:

(i) \( y^{t+1} \in MC_G \).

(ii) \( \text{CUT}_{G_S}(\hat{y}_{E_S}) \leq \text{CUT}_{G_S}(y^{t+1}_{E_S}) \Rightarrow \text{CUT}_G(y^{t+1}) \leq \text{CUT}_G(y^t) \).

**Proof.** (i) Let us extend the local segmentation \( \hat{y} \) to \( G \) with

\[
\hat{y}_G := \begin{cases} 
\hat{y}_e & \forall e \in E_S \\
1 & \forall e \in E_{G - \overline{S}} \\
0 & \forall e \in E_{\overline{S}}
\end{cases}
\]

Since (a) \( \hat{y}_G \in MC_G \), (b) \( y^{t+1} = \text{OR}(\hat{y}_G, y^t) \), and (c) the set of multicut is closed under the OR operation, (i) holds.

(ii) For \( \text{CUT}_{G_S}(\hat{y}_{E_S}) \leq \text{CUT}_{G_S}(y^{t+1}_{E_S}) \), the energy after the move can be written as:

\[
\begin{align*}
\text{CUT}_G(y^{t+1}) &= \sum_{e \in E_S} w(e)y^{t+1}_e + \sum_{e \in E \setminus E_S} w(e)y^{t+1}_e \\
&\leq \sum_{e \in E_S} w(e)y^t_e + \sum_{e \in E \setminus E_S} w(e)y^{t+1}_e \\
&= \sum_{e \in E_S} w(e)y^t_e + \sum_{e \in E \setminus E_S} w(e)y^{t+1}_e = \text{CUT}_G(y^t)
\end{align*}
\]

To obtain an alternative segmentation \( \hat{y} \) on \( G_S \), one can either (a) solve the multicut problem over \( G_S \) (which is smaller than \( G \)) or (b) restrict the subset of possible segmentations, e.g. to all two-colorable segmentations of \( G_S \). If we solve the problem to optimality and the current segmentation is in the feasible set of the move, it is guaranteed that
\begin{algorithm}[h]
\caption{Cut, Glue & Cut algorithm}
\begin{algorithmic}[1]
\Require weighted graph $G = (V, E, w)$
\Ensure approx. solution $Q$ to (1) for $G$
\State $Q^0 \leftarrow$ segmentation of $G$ into positively connected components
\For{$n = 1 \ldots n_{\text{iter}}$}
\State $Q^n \leftarrow \text{cut\_phase}(G, Q^{n-1})$ \hspace{0.5em} // Alg. 2
\State $Q^n \leftarrow \text{glue\_cut\_phase}(G, Q^n)$ \hspace{0.5em} // Alg. 3
\If{$Q^n = Q^{n-1}$}
\State exit
\EndIf
\EndFor
\end{algorithmic}
\end{algorithm}

$y^{t+1}$ never increases the energy. For approximate solutions this is not the case; here the move should only be accepted in case of energy decrease. While (b) restricts the set of possible moves, the problems become easier and in most cases, a specific energy-decreasing sequence of two-coloring moves (b) can generate the same segmentation as one single $N$-coloring move (a) as sketched in Fig. 2. However, even if an energy-decreasing sequence exists, finding this sequence itself is a hard combinatorial optimization problem, which is why we choose a greedy optimization approach.

6. Cut, Glue & Cut Algorithm

Overview The CGC algorithm (Alg. 1) always maintains a valid partitioning of the graph, which is iteratively improved by local moves which act on single or neighboring regions. It works in two distinct phases: (i) recursive cut phase (ii) glue & cut phase.

The cut phase recursively splits regions by solving max-cut problems (3), finally yielding a finer, lower energy segmentation driven by the problem’s weights. In the glue & cut phase, any two neighboring segments are first merged and then a new cut is sought between them by solving (3). These two phases are repeated and the process stops when the energy cannot decrease any further.

Cut Phase In the cut phase, the regions are recursively split into smaller and smaller regions tuned to the weights $w$

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{cut_phase.png}
\caption{Top: Solving the labeling problem (1) with more than two colors allows for T-junctions. Bottom: A sequence of two colorings can model the same T-junction and is often energy-decreasing.}
\end{figure}

until a local optimum is found. As cut moves we consider the max-cut solution (3) for a single given region $R$ to find a better segmentation of the region $\hat{y} \in \text{MC}_{G_R}$, and accept the local move if the energy could be decreased compared to the previous solution $\text{CUT}_{G_R}(y_R) = 0$. The Partition Move Theorem then guarantees a monotonically decreasing global energy for the segmentation.

The cut phase is illustrated in the left part of Fig. 1, and is given as Algorithm 2. All regions are first inserted into a queue $Q$. While $Q$ is not empty, take a region $R \in Q$, then solve the max-cut problem (3) for $G_R$. Finally, a connected component labeling of the solution defines a new set of regions\footnote{Note that in general there may be more than two connected regions, cf. Fig. 1, initial 2-coloring.}. If the suggested cut reduces the local energy, we add the induced regions to $Q$. Otherwise, the energy of region $R$ cannot further be reduced by this type of move; region $R$ is marked as “done” and added to a list $Q'$.

\begin{algorithm}[h]
\caption{Cut phase}
\begin{algorithmic}[1]
\Require weighted graph $G = (V, E, w)$, segmentation into regions given as queue $Q$
\Ensure segmentation into smaller regions $Q'$
\State $Q' \leftarrow \emptyset$
\While{$Q \neq \emptyset$}
\State $R \leftarrow \text{queue\_pop}(Q)$
\State $y_{E_R} \leftarrow \text{solve\_max-cut}(3)$ for $G_R$
\If{$\text{CUT}_{G_R}(y_{E_R}) < 0$}
\State $Q \leftarrow Q \cup \text{connected\_components}(y_{E_R})$
\Else
\State $Q' \leftarrow Q' \cup R$
\EndIf
\EndWhile
\end{algorithmic}
\end{algorithm}

Glue & Cut Phase In this phase, we consider Glue & Cut moves of pairs of adjacent segments until the energy cannot be decreased any further, yielding a better solution.

Intuitively, we expect that two common local operations can decrease the energy: (i) merging two segments or (ii) moving the boundary between two adjacent segments. This motivates our algorithm:

We again apply the Partition Move Theorem from Sec. 5. Given two adjacent regions $R_1$, $R_2$ in the current segmentation, we consider the merged region $R = R_1 \cup R_2$ (glue), and find a new segmentation $\hat{y} \in \text{MC}_{G_R}$ (cut) by solving the max-cut problem (3). The local move is accepted if the energy $\text{CUT}_{G_R}(y)$ is lower than the energy of the previous cut. The Partition Move Theorem then guarantees a monotonically decreasing energy for the segmentation.

Formally, (Alg. 3 and Fig. 1, right), the glue & cut phase starts from a given segmentation $Q$. We first obtain its edge
The following is repeated (line 3): For each pair \( R \) chosen subsequently, a glue & cut move could alter region \( R \) between \((R_1, R_2)\) of adjacent regions we pick a single representative edge from the shared boundary \( E \cap (R_1 \times R_2) \) to form the set \( E' \) (line 5). Then, for each such representative \( e = (i, j) \) with \( y_e = 1 \) and which is marked dirty, we find the best glue & cut move as described above. If, after processing all \( e \in E' \), no move could be performed, we break out of the outer loop (line 22).

**Book-keeping of modified boundaries.** In order to avoid re-solving the same problem multiple times, the algorithm marks edges as “dirty” or “clean”. If two adjacent regions are only separated by clean edges, a glue & cut move is not considered for this pair. In Alg. 3, statements related to book-keeping are marked with \((*)\).

Imagine an accepted glue & cut move on region \( R \) which yields exactly two regions \( R_1 \) and \( R_2 \). We mark the boundary between \( R_1 \) and \( R_2 \), \( E' \cap (R_1 \times R_2) \), as clean, since re-solving (3) for \( R_1 \cup R_2 \) would again lead to the same two-coloring into \( R_1 \) and \( R_2 \). However, if an adjacent pair \((R_1, R_3)\) is chosen subsequently, a glue & cut move could alter region \( R_1 \) into \( R'_1 \). Therefore, a move between \( R'_1 \) and \( R_2 \) could improve the energy again. To allow this move it is necessary to mark the boundary of \( R'_1 \cup R_2 \) as dirty. Note that for the case where an accepted move yields more than two regions, bookkeeping as above would not be correct and the internal edges of \( R \) have to be marked dirty.

### 7. Experiments

We evaluate the performance of our Cut, Glue & Cut algorithm on two different 2D segmentation benchmarks as well as on a new 3D volume segmentation benchmark.

**Algorithms.** For our CGC algorithm, we consider three variants: CGC-B does not do book-keeping, CGC-P does not use the globally optimal Blossom solver for planar max-cut problems, but rather uses the approximate QPBO-I. CGC-PB does not do book-keeping while using QPBO-I as max-cut solver.

We compare against the following algorithms: Kernighan-Lin (KL, [16]), Planar Correlation Clustering (PlanarCC, [23]), Expand-and-Explore [7], LP-based cutting plane method of a relaxed problem (MC-R), integer linear program based cutting plane method (MC-I), which always finds the globally optimal solution. MC-R and MC-I use facet-defining separation procedures and bounding techniques as described in [15]. We use the publicly available C++ implementation in OpenGM 2.1.1 [3] for KL, MC-R and MC-I. For Expand-and-Explore, we use the publicly available code\(^4\) of the corresponding authors. For PlanarCC, we kindly obtained the implementation by the authors of [23]. While both are implemented in MATLAB, all computation-heavy parts are delegated to C++ functions via MEX wrappers, such that our comparison is fair.

For PlanarCC, we follow the suggestion of the authors to stop the algorithm after 40 iterations for better runtime. Without this, for several instances, PlanarCC does not converge after one hour. With more iterations, the energy of the solutions improves slightly but at the expense of significantly longer runtime.

Note that in the Expand-and-Explore algorithm, each binary sub-problem includes by construction unary terms, which leads to non-planar sub-problems even when the original problem is planar.

**2D segmentation** We consider planar 2D segmentation problems derived from the Berkeley segmentation database (BSD300): (i) models from [4] (test data, 100 instances) and (ii) models from [23] (training data, 200 instances). While the former uses local edge likelihoods learned by a Random Forest, the latter uses global probability of boundary (gPb). Furthermore, they obtain edge weights \( w \) from the edge

---

\(^4\)http://www.wisdom.weizmann.ac.il/~bagon
Figure 3: Comparison of the different multicut algorithms for a model from [4], based on the superpixel segmentation in (b). The colored boundaries indicate true positives (yellow), false negatives (red), false positives (blue) and true negatives (invisible) with respect to the globally optimal solution (f). **Top:** The image (a) is partitioned into superpixels (b). A single two-coloring leads to (c) and the cut phase ends with (d). The final output of the CGC algorithm is (e). **Bottom:** Results of various competitive methods.

Probabilities $p_e$ using different parameterizations. In [4],

$$w_e = \log \left( \frac{p(y_e = 0)}{p(y_e = 1)} \right) + \log \frac{1 - \beta}{\beta}$$  \hspace{1cm} (5)

is chosen while [23] use

$$w_e = \log \left( \frac{1 - gb_p_e}{gb_p_e} \right) + \gamma.$$  \hspace{1cm} (6)

Results for both datasets and different values of $\beta$ and $\gamma$ are shown in Fig. 4b and 4a. MC-I finds the global optimum for all instances. The left plots show the average energy distance (mean gap) to the optimum. Among all approximate methods, CGC performs best. Concerning runtimes, CGC has robustly low runtime for a wide range of parameters $\beta, \gamma$.

A more detailed comparison for $\beta = 0.33$, as used in [13], and $\gamma = 0.3$ can be found in Tab. 2, left and right, respectively. These tables additionally show how often the methods find the global optimum (best), and how often they were able to verify the optimality by themselves (ver. opt.). For Tab. 2, left, groundtruth segmentations and superpixels were available, such that we can calculate the Variation of Information (VI, [20]). Interestingly, the globally optimal solution does not necessarily give the best segmentation as measured by the VI (but approximately, VI distance gets larger with increasing energy as expected). This means that in practice, it is sufficient to run the much faster CGC algorithm, see also Fig. 3.

### Table 1: Performance of various multicut solvers on instances derived from [5] (8 instances, cube length $N = 400$ voxels). See Tab. 2 for a column legend. ver. opt. is seven for MC-I, else zero.

<table>
<thead>
<tr>
<th>algorithm</th>
<th>runtime</th>
<th>value</th>
<th>bound</th>
<th>best</th>
<th>VI</th>
</tr>
</thead>
<tbody>
<tr>
<td>ogm-KL</td>
<td>85.06 sec</td>
<td>-53476.75</td>
<td>$\infty$</td>
<td>0</td>
<td>4.6930</td>
</tr>
<tr>
<td>Expand&amp;Explore</td>
<td>1087.84 sec</td>
<td>-57054.25</td>
<td>$\infty$</td>
<td>0</td>
<td>3.1228</td>
</tr>
<tr>
<td>CGC−$P$</td>
<td>64.95 sec</td>
<td>-57206.18</td>
<td>$\infty$</td>
<td>1</td>
<td>2.1514</td>
</tr>
<tr>
<td>MC-R</td>
<td>4121.08 sec</td>
<td>-20111.68</td>
<td>-58774.97</td>
<td>0</td>
<td>4.0600</td>
</tr>
<tr>
<td>MC-I</td>
<td>745.53 sec</td>
<td>-57319.41</td>
<td>-57386.73</td>
<td>7</td>
<td>1.8539</td>
</tr>
</tbody>
</table>

For 3D segmentation, recently Kappes et al. [13] released a benchmark dataset. However, this only includes one small and one huge instance. Therefore, we derive a more comprehensive set of instances with volumes of $30^3$ up to $450^3$ voxels from the models in [5], available as part of the OpenGM benchmark [12]. Results are shown
Figure 4: Evaluation on various benchmark datasets. **Left:** Gap to the global optimal energy (MC-I) averaged over all instances. **Right:** Mean runtime of the methods. For both 2D and 3D images, CGC outperforms all competitors in terms of runtime. On 2D images, CGC gives better results in terms of energy than all competitive approximative methods. Only Integer Multicut (MC-I) gives better partitions in terms of energy but is significantly slower. For 3D volume image segmentation, the approximate MC-R method beats CGC-\(\mathcal{P}\), though at significant runtime cost. CGC is not applicable due to lack of planarity.

Based on the energy difference of CGC and CGC-\(\mathcal{P}\) for the planar models, we expect reduced performance for the non-planar case of 3D volume image segmentation (Fig. 4c). Still, with a runtime as fast as KL (the fastest algorithm considered), CGC-\(\mathcal{P}\) is able to obtain much lower-energy solutions and lies between MC-I, MC-R and Expand & Explore. For instances with 400\(^3\) voxels, MC-I was only able to find for 7 out of 8 instances the global optimum within one hour (Tab. 1).

8. Conclusion

We have presented a new approximate solver, called Cut, Glue & Cut (CGC) for planar and non-planar multicut problems based on the move-making paradigm. It works in two phases: in the Cut phase, a low energy segmentation tuned to the problem’s weights is created by recursively solving 2-coloring problems, either using the Blossom method (for planar problems) or QPBO-I. In the Glue & Cut phase, two adjacent sub-problems are first merged (glue) and then a possibly better cut is sought between them (cut). This process is
repeated until no energy improvement is possible anymore.
The experimental evaluation shows that CGC is considerably faster than existing methods while able to match an exact solver in quality, as measured by the Variation of Information on 2D images.

References


Table 2: Mean runtime, energy and bound (if available) for different multicut solvers. Left: Summary over the models from [4, 100 instances. The VI column reports the Variation of Information [20]. Right: Summary over the models from [23], 200 instances. Without superpixel maps available, VI could not be calculated.

<table>
<thead>
<tr>
<th>algorithm</th>
<th>runtime</th>
<th>value</th>
<th>bound</th>
<th>best</th>
<th>ver.</th>
<th>opt</th>
<th>VI</th>
</tr>
</thead>
<tbody>
<tr>
<td>KL</td>
<td>4.96 sec</td>
<td>4608.57</td>
<td>−∞</td>
<td>0</td>
<td>0</td>
<td>2.6431</td>
<td></td>
</tr>
<tr>
<td>Expand &amp; Explore</td>
<td>2.90 sec</td>
<td>4486.57</td>
<td>−∞</td>
<td>1</td>
<td>0</td>
<td>2.9153</td>
<td></td>
</tr>
<tr>
<td>CGC-PP</td>
<td>6.35 sec</td>
<td>4466.80</td>
<td>−∞</td>
<td>1</td>
<td>0</td>
<td>2.5027</td>
<td></td>
</tr>
<tr>
<td>CGC-CP</td>
<td>5.35 sec</td>
<td>4466.80</td>
<td>−∞</td>
<td>1</td>
<td>0</td>
<td>2.5027</td>
<td></td>
</tr>
<tr>
<td>CGC-BO</td>
<td>0.63 sec</td>
<td>4445.06</td>
<td>−∞</td>
<td>23</td>
<td>0</td>
<td>2.5355</td>
<td></td>
</tr>
<tr>
<td>CGC-BO</td>
<td>0.42 sec</td>
<td>4445.06</td>
<td>−∞</td>
<td>23</td>
<td>0</td>
<td>2.5355</td>
<td></td>
</tr>
<tr>
<td>MC-R</td>
<td>5.16 sec</td>
<td>4447.47</td>
<td>4442.34</td>
<td>35</td>
<td>35</td>
<td>2.5490</td>
<td></td>
</tr>
<tr>
<td>Planar-CC</td>
<td>5.20 sec</td>
<td>4450.73</td>
<td>4437.29</td>
<td>9</td>
<td>8</td>
<td>2.5603</td>
<td></td>
</tr>
<tr>
<td>MC-I</td>
<td>2.20 sec</td>
<td>4442.64</td>
<td>4442.64</td>
<td>100</td>
<td>100</td>
<td>2.5363</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>algorithm</th>
<th>runtime</th>
<th>value</th>
<th>bound</th>
<th>best</th>
<th>ver.</th>
<th>opt</th>
<th>VI</th>
</tr>
</thead>
<tbody>
<tr>
<td>KL</td>
<td>0.04 sec</td>
<td>−73.41</td>
<td>−∞</td>
<td>45</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Expand &amp; Explore</td>
<td>0.03 sec</td>
<td>−89.90</td>
<td>−∞</td>
<td>130</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CGC-PP</td>
<td>0.03 sec</td>
<td>−89.45</td>
<td>−∞</td>
<td>104</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CGC-CP</td>
<td>0.03 sec</td>
<td>−89.45</td>
<td>−∞</td>
<td>104</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CGC-BO</td>
<td>0.03 sec</td>
<td>−92.25</td>
<td>−∞</td>
<td>185</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CGC-BO</td>
<td>0.03 sec</td>
<td>−92.25</td>
<td>−∞</td>
<td>185</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MC-R</td>
<td>3.48 sec</td>
<td>−91.70</td>
<td>−92.39</td>
<td>181</td>
<td>180</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Planar-CC</td>
<td>0.36 sec</td>
<td>−92.16</td>
<td>−92.39</td>
<td>184</td>
<td>174</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MC-I</td>
<td>29.80 sec</td>
<td>−92.35</td>
<td>−92.35</td>
<td>200</td>
<td>200</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>