Efficient Parallel Optimization for Potts Energy with Hierarchical Fusion

Olga Veksler
University of Western Ontario
London, Canada
olga@csd.uwo.ca

Abstract

Potts energy frequently occurs in computer vision applications. We present an efficient parallel method for optimizing Potts energy based on the extension of hierarchical fusion algorithm. Unlike previous parallel graph-cut based optimization algorithms, our approach has optimality bounds even after a single iteration over all labels, i.e. after solving only k-1 max-flow problems, where k is the number of labels. This is perhaps the minimum number of max-flow problems one has to solve to obtain a solution with optimality guarantees. Our approximation factor is $O(\log_2 k)$. Although this is not as good as the factor of 2 approximation of the well known expansion algorithm, we achieve very good results in practice. In particular, we found that the results of our algorithm after one iteration are always better than the results after one iteration of the expansion algorithm. We demonstrate experimentally the computational advantages of our parallel implementation on the problem of stereo correspondence, achieving a factor of 1.5 to 2.6 speedup compared to the serial implementation. These results were obtained with a small number of processors. The expected speedups with a larger number of processors are greater.

1. Introduction

Energies with Potts pairwise term are frequently used in various vision and graphics applications [7, 1, 23, 12, 3]. Potts model is popular because it is, arguably, the simplest useful model that imposes smoothness on the solution. Although optimizing Potts energy is NP-hard [7] , there are efficient approximate optimization algorithms [7, 14].

According to the energy evaluations in [29, 13], TRWS [14] and expansion and swap algorithms [7] optimize Potts energy particularly well. The expansion algorithm [7] has the best trade-off in terms of the speed and the accuracy of optimization. An additional advantage of expansion is that it has a factor of 2 approximation for Potts energy, and this factor is tight [8]. The focus of this paper is improving the computational efficiency of graph-cut optimization of Potts energy through parallel implementation.

There are many approaches to improve the running time of the expansion algorithm. One direction is to re-use information between different instances of max-flow computations, or by taking advantage of the structure of the optimization problem in some clever way. FastPD method of [17] exploits information not only from the original optimization problem but also from the dual one, intuitively reusing information from one max-flow computation to the next. In [2, 4], they obtain improvements that are similar to that in [17] by reusing the flow between min-cut computations, and by using good initializations. Another approach to speed-up is in [5], where they adaptively find the sequence of expansion moves that is likely to lead to the biggest energy decrease. This results in faster overall convergence of the expansion algorithm.

A different line of research is concerned with extracting part of the optimal solution [18, 27, 11]. Pixels for which the optimal label is extracted are removed from the optimization problem. The smaller problem is solved by a standard method, such as expansion [7] or FastPD [17]. For example, [18] solves k max-flow problems, where k is the number of labels, to obtain a partially optimal solution. In [11], they improve the efficiency of [18] by developing a method that only needs $1 + log_2 k$ max-flow computations. The approach in [11] is based on generalizing the algorithm of [10], which is an exact fast algorithm for optimizing the special case of energies with tree-metric label space. It is worth noting that partial optimality methods exist for models more general than Potts [28, 26], and they are useful beyond just computational speed-ups.

Partial optimality algorithms can significantly speed up computation. For example [11] reports $50 - 93\%$ of variables labeled by their approach. However, there are no a priori guarantees on the percentage of the solution that will be obtained. Partial optimality works best when unary terms are less ambiguous, namely when many pixels show strong preference to a specific label. For example, to get a better percentage of optimally labeled pixels, [11] uses 9×9 win-

dow aggregated SSD score for unary terms. Aggregating over windows may worsen results close to label discontinuities. Furthermore for some problems, strong unary terms may not be easy to design. In the extreme case, unary terms are based on hard-constraints, i.e. either zero or infinity, for example in image inpainting [29] or superpixel tessellation [30, 31], making partial optimality inapplicable.

We propose a graph-cut approach to Potts energy minimization that is easy to parallelize. Our approach is based on extending the hierarchical fusion (h-fusion) algorithm of [9], which is based on the fusion algorithm of [21]. The h-fusion algorithm of [9] is almost identical to the hierarchical algorithm of [19] in the absence of the label cost terms.

In [21] they also propose a graph-cut based parallel optimization that uses a mixture of expansion and fusion moves. However, their approach, in general, has no optimality guarantees. Our approach has optimality guarantees even after just one iteration over all labels, i.e. after solving $k-1$ max flow problems, where k is the number of labels. Intuitively, it seems plausible that one has to solve at least that number of max-flow problems to have any optimality guarantees.

Our approximation factor is $O(\log_2 k)$. Although this is not as good as the factor of 2 approximation of the expansion algorithm, we show that in practice we achieve very good results. In particular, the results of our algorithm after one iteration are always better than the results after one iteration of the expansion algorithm. In fact, we show that the expansion algorithm does have an approximation factor after just one iteration, if initialized properly. However, this factor is $O(k)$, much worse than that of our algorithm. This helps to explain why we get better results after one iteration.

The observation above is particularly useful given that the expansion algorithm gives excellent results for Potts model even after just one iteration. That is for some applications, one iteration may be sufficient. Also, if one has a time-critical application, one iteration over all labels may be all that one can afford to execute.

We demonstrate computational advantages of our parallel implementation on the problem of stereo correspondence, achieving a factor of 1.5 to 2.6 speedup compared to serial implementation. These results are obtained with a small number of processors. The expected speedup with a larger number of processors is greater.

2. Related Work

2.1. Potts Energy

Many problems in computer vision can be formulated as labeling problems, where the task is to assign each pixel $p \in P$ a label x_p from some set $\mathcal{L}$. Let $\boldsymbol{x} = (x_p \,|\, p \in P)$ be a vector of all variables. One formulates an energy function $f(\boldsymbol{x})$ that typically consists of unary $f_p(x_p)$ and pairwise $f_{pq}(x_p, x_q)$ terms. Labeling $\boldsymbol{x}$ that minimizes (exactly or approximately) the energy function is taken as the solution.

With Potts model for pairwise terms, the energy is:

$$f(\boldsymbol{x}) = \sum_{p \in P} f_p(x_p) + \sum_{\{p,q\} \in E} w_{pq} \cdot [x_p \neq x_q], \quad (1)$$

where P is the set of image pixels, E is the set of neighboring pixel pairs, $[\cdot]$ is the Iverson bracket, and $w_{pq} \geq 0$.

2.2. Expansion Algorithm

Given an initial labeling $\boldsymbol{x}$ and some label $\alpha \in \mathcal{L}$, an α-expansion move lets each variable to either keep its current label x_p, or switch to α. Let $\mathcal{M}^\alpha(\boldsymbol{x})$ denote the set of all α-expansion moves (labelings)

$$\mathcal{M}^\alpha(\boldsymbol{x}) = \left\{ \boldsymbol{x}' \,:\, x'_p \neq x_p \;\Rightarrow\; x'_p = \alpha \right\}. \quad (2)$$

The problem of finding an optimal move in $\mathcal{M}^\alpha(\boldsymbol{x})$ is formulated as minimization of binary energy [7]. For Potts model this binary energy is submodular [16], and, therefore, can be optimized with a single graph-cut [6]. The expansion algorithm then iterates over all labels in $\mathcal{L}$, performing expansion moves, until no move improves the energy.

Algorithm 1: α-EXPANSION

1 $\boldsymbol{x}' :=$ arbitrary labeling
2 **repeat**
3 **for each** $\alpha \in \mathcal{L}$
4 $\boldsymbol{x}' := \operatorname{argmin}_{\boldsymbol{x} \in \mathcal{M}^\alpha(\boldsymbol{x}')} f(\boldsymbol{x})$
5 **until** converged
6 **return** $\boldsymbol{x}'$

Lines $(3-4)$ in Algorithm 1 cause $\mathcal{L}$ max-flow problems to be executed, and these must be executed in a serial fashion. Note that the factor of two approximation is guaranteed to hold only after the algorithm has converged fully, i.e. after the **repeat** loop has terminated. There is, in general, no theoretical bound[1] on the number of **repeat** loops needed for the expansion algorithm to converge.

Note that we can use the analysis in Sec. 3.3 to show that the expansion algorithm has a factor of $O(k)$ approximation after one iteration over all labels. This factor is much worse compared to our factor of $O(\log_2 k)$ approximation.

2.3. Fusion Algorithm

We now describe fusion moves, introduced in [20, 21]. The idea behind fusion moves is to 'stitch' solutions from multiple algorithms to obtain the best combined result.

Given two candidate labellings $\boldsymbol{x}^1$ and $\boldsymbol{x}^2$, we say that a labeling $\hat{\boldsymbol{x}}$ is a *fusion* of $\boldsymbol{x}^1, \boldsymbol{x}^2$ if $\forall p,\ \ \hat{x}_p \in \{x^1_p, x^2_p\}$. We use $Fuse(\boldsymbol{x}^1, \boldsymbol{x}^2)$ to denote the set of all fusions of $\boldsymbol{x}^1, \boldsymbol{x}^2$.

Suppose we have n candidate labellings $\boldsymbol{x}^1, ..., \boldsymbol{x}^n$. Fusion algorithm, summarized below, starts with some initial labeling $\boldsymbol{x}'$ and then repeatedly fuses $\boldsymbol{x}'$ with $\boldsymbol{x}^i$ for all i.

[1] Besides the obvious one that depends on the energy value.

Algorithm 2: FUSION

1 $\boldsymbol{x}' :=$ arbitrary labeling
2 **repeat**
3 **for each** $i \in \{1, 2, ..., n\}$
4 $\boldsymbol{x}' := \text{argmin}_{\boldsymbol{x} \in Fuse(\boldsymbol{x}', \boldsymbol{x}^i)} f(\boldsymbol{x})$
5 **until** converged
6 **return** $\boldsymbol{x}'$

Like an expansion move, a fusion move can be formulated as binary energy minimization. Let us be specific about this formulation. Let $\boldsymbol{x}^1$ and $\boldsymbol{x}^2$ be two labellings we wish to 'fuse'. For each pixel p, we introduce a binary variable y_p and store these binary variables in a binary vector $\boldsymbol{y}$. If $y_p = 0$, pixel p is assigned label x_p^1 by the fusion move. If $y_p = 1$, pixel p is assigned label x_p^2. We formulate the following binary energy $h(\boldsymbol{y})$:

$$h_p(0) = f_p(x_p^1) \qquad h_{pq}(0,0) = f_{pq}(x_p^1, x_q^1)$$
$$h_p(1) = f_p(x_p^2) \qquad h_{pq}(0,1) = f_{pq}(x_p^1, x_q^2)$$
$$h_{pq}(1,0) = f_{pq}(x_p^2, x_q^1)$$
$$h_{pq}(1,1) = f_{pq}(x_p^2, x_q^2)$$

A fusion move is submodular [16], and, therefore, can be optimized with a graph cut if

$$f_{pq}(x_p^0, x_q^0) + f_{pq}(x_p^1, x_q^1) \leq f_{pq}(x_p^0, x_q^1) + f_{pq}(x_p^1, x_q^0) \quad (3)$$

Eq. (3) is not guaranteed to be satisfied for arbitrary $\boldsymbol{x}^1$ and $\boldsymbol{x}^2$ even in the case of Potts energy. In [21], they use QPBO [15, 22] for non-submodular energy minimization.

Most related for our work, in [21] they propose a parallel implementation of the expansion algorithm using a mixture of expansion and fusion moves. For example, to implement expansion on two processors, the label set $\mathcal{L}$ is evenly split into $\mathcal{L}_1$ and $\mathcal{L}_2$. Two initial labellings $\boldsymbol{x}^1$ and $\boldsymbol{x}^2$ are chosen and one iteration of expansion is performed in parallel on subset $\mathcal{L}_1$ starting with $\boldsymbol{x}^1$ and subset $\mathcal{L}_2$ starting with $\boldsymbol{x}^2$. Suppose the results of these parallel expansions are stored in the same variables, namely $\boldsymbol{x}^1$ and $\boldsymbol{x}^2$. Two results are then combined using a fusion move: $\boldsymbol{x}^1 = \text{argmin}_{\boldsymbol{x} \in Fuse(\boldsymbol{x}^1, \boldsymbol{x}^2)} f(\boldsymbol{x})$. Then two parallel expansions are repeated again, now starting with 'fused' $\boldsymbol{x}^1$ for both cases. This process is repeated until convergence. The algorithm is summarized below, where the **for** loop on lines $(4-6)$ is performed in parallel. This algorithm can be generalized from two processors to m processors by splitting the label set into m subsets and combining the results of m expansion sub-problems with the fusion moves.

It is easy to see that the fusion step on line 7 is not guaranteed to be submodular even for the Potts model. No matter how the initial $\boldsymbol{x}^1$ and $\boldsymbol{x}^2$ are chosen, at the second execution of the fusion step on line 7, current $\boldsymbol{x}^1$ and $\boldsymbol{x}^2$ can be arbitrary, i.e. can contain any label from $\mathcal{L}$. Therefore, for a pair of neighbors p, q, it can happen that

$$x_p^1 = \alpha, \quad x_q^1 = \beta, \quad x_p^2 = \beta, \quad x_q^2 = \alpha,$$

and so submodularity condition in Eq. 3 does not hold. Therefore, no optimality guarantees hold for the parallel fusion algorithm proposed in [21].

Algorithm 3: PARALLEL FUSION

1 $\boldsymbol{x}^1, \boldsymbol{x}^2 :=$ arbitrary labelings
2 $\mathcal{L} = \mathcal{L}_1 \cup \mathcal{L}_2$ // even split of the label set
3 **repeat**
4 **for each** $i = 1, 2$ **in parallel do**
5 **for each** $\alpha \in \mathcal{L}_i$
6 $\boldsymbol{x}^i := \text{argmin}_{\boldsymbol{x} \in \mathcal{M}^\alpha(\boldsymbol{x}^i)} f(\boldsymbol{x})$
7 $\boldsymbol{x}^1 := \text{argmin}_{\boldsymbol{x} \in Fuse(\boldsymbol{x}^1, \boldsymbol{x}^2)} f(\boldsymbol{x})$
8 $\boldsymbol{x}^2 := \boldsymbol{x}^1$
9 **until** converged
10 **return** $\boldsymbol{x}^1$

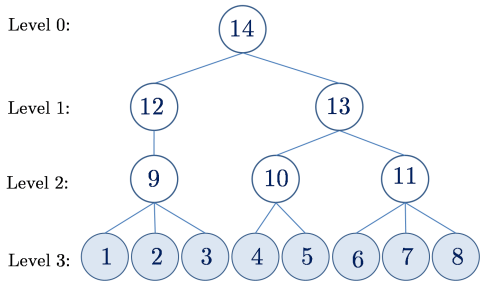

Figure 1. Label tree illustration. Here the label set $\mathcal{L} = \{1, 2, ..., 8\}$. Each label corresponds to a leaf of the tree. The tree is organized in 4 levels. $C(i)$ is the set of indexes of children for a non-leaf node i. For example, $C(11) = \{6, 7, 8\}$. $C_l(i)$ is the first (leftmost) child of node i. For example, $C_l(14) = 12$.

2.4. Hierarchical Fusion Algorithm

The hierarchical fusion (h-fusion) algorithm of [9] was developed for a special class of energies with *h-metric* pairwise terms. This algorithm is essentially the same[2] as the hierarchical graph cut algorithm of [19], although the goals are different. In [19], they develop graph-cut approximation algorithms with better optimality bounds[3] for energies with the metric and semi-metric pairwise terms. In [9], the goal is better optimization of energies with h-metric terms.

We will not describe what h-metric pairwise terms are, since our goal is optimization of Potts energy. Instead, we will describe how h-fusion algorithm works. First, we need to organize the label set into a 'label tree' (a hierarchy), see

[2] This statement is true in the absence of label costs in the energy.

[3] They achieve the same optimality bounds for metric and semi-metric energies as LP relaxation base solvers, but using more efficient graph-cut based optimization.

Fig. 1. Without loss of generality, let us assume the label set is $\mathcal{L} = \{1, 2, .., k\}$. The labels correspond to the leaves of the tree. A 'label tree' is then a collection of nodes with parent-child relationships. Each non-leaf node of the tree can have from 1 to k children. Each node, except the root, has one parent. We index nodes with integers $1, ..., r$, where r is the number of nodes in the tree. We assume the root has the largest index. For each non-leaf node i, let $C(i)$ be the set of the indexes of its children, and $C_l(i)$ be the index of the first (leftmost) child. Let $Level_d$ be the set of all nodes at level d. Let the largest level of the tree be D, for the total number of levels $D + 1$. It is convenient to assume that all leaves are at the same (maximum) level.

Hierarchical fusion starts at tree level $D - 1$ and proceeds upward to level 0, see Fig. 2. For each node j at level $D - 1$, its children correspond to the labels of our optimization problem. Therefore for each $j \in Level_{D-1}$, we perform α-expansion moves for all $\alpha \in C(j)$. For shorter pseudo-code, it is convenient to express these expansions as fusion moves. Therefore we associate a 'constant' labeling $\boldsymbol{x}^i = (x_p = i | p \in P)$ with each leaf node i. We perform fusion moves with $\boldsymbol{x}^i, i \in C(j)$. The result of this expansion (fusion) is stored in labeling $\boldsymbol{x}^j$, which is associated with node j. Now the processing for nodes j at levels smaller than $D - 1$ proceeds identically. For each node, a fusion among its children nodes is performed and the results are stored in $\boldsymbol{x}^j$. The root node r stores the final result in $\boldsymbol{x}^r$. The pseudo-code for hierarchical fusion is given below.

Algorithm 4: HIERARCHICAL FUSION

```
// initialize k constant labellings
for each i = 1, ..., k do
    for each p ∈ P do
        x_p^i = i
// perform fusions from the bottom up
for each d = D − 1, ..., 1, 0 do
    for each j ∈ Level_d in parallel do
        c := C_l(j)
        x^j = x^c
        repeat
            for each c ∈ C(j) do
                x^j := argmin_{x ∈ Fuse(x^j, x^c)} f(x)
        until converged
return x^r
```

3. Efficient Parallel Implementation

We now develop our efficient parallel algorithm based on hierarchical fusion discussed in Sec. 2.4. We explain how we choose the label tree structure, prove optimality bounds, and extend the hiearchical fusion algorithm to be iterative, since the original algorithm in [9, 19] is not iterative.

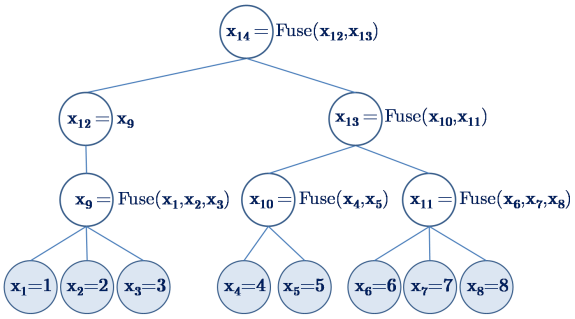

Figure 2. Illustration of hierarchical fusion. See text for details.

3.1. Our Tree Label Structure

We would like to achieve results with a guaranteed factor from the optimal with the smallest possible number of max-flow operations. We propose to use the tree structure where each node has at most two children, that is a binary tree. An example of such tree structure for the case $|\mathcal{L}| = 7$ is illustrated in Fig. 3. We also construct a binary tree that is of smallest height possible, that is $\log_2 k$, since our optimality properties depend on the height of the tree.

The advantage of this structure is that for each internal node, fusion needs to be performed just one time, that is no iterations are needed. Thus there are exactly $|\mathcal{L}| - 1$ max-flow computations that need to be performed. This is perhaps the smallest number of iterations one needs to perform for an answer with optimality guarantees.

Our optimality guarantees depend on the height of the tree. If we allow more than two children per node, the tree will be of smaller height. However with more than two children per node, computations at each node have to be iterated until convergence. And there is no "useful" (i.e. small) known bound on the number of iterations needed to be performed until convergence is achieved.

Unlike that in Fig. 3, the order of the leaves does not have to be consecutive. In practice, we found consecutive order to work best, on average. We will get some insights on why this is the case when we prove the optimality bounds.

It is easy to see that with the structure above, for Potts model, the fusion move is performed optimally at each node. That is the corresponding binary problem is submodular. This is because by construction, any two labelings that are to be fused do not share any common label. Therefore

$$f_{pq}(x_p^0, x_q^1) = f_{pq}(x_p^1, x_q^0) = w_{pq},$$

so the right side of Eq. 3 is never smaller than the left side.

3.2. Our Algorithm

Algorithm 4 can be parallelized in the obvious manner. Starting at the deepest level, all fusions at a fixed level l are solved in parallel. However, this does not reach the best

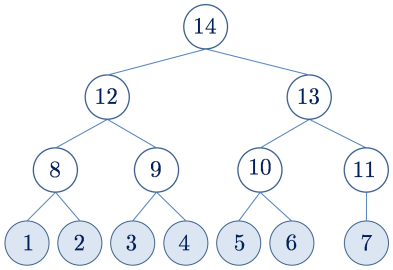

Figure 3. Our label tree structure.

level of parallelization. For example, consider Fig. 3. Suppose a processor fusing labellings at nodes 8 and 9 is done, and, therefore, fusion at node 12 is ready to be executed. However, if all fusions at level 2 are not done yet, the fusion at node 12 will not be invoked.

To achieve maximum parallel performance, we repeatedly iterate over all non-leaf nodes in the label tree, check if the fusion at the children nodes has already completed, and, if yes, signal that fusion at this node is ready to proceed. Although the root node can be excluded from this loop, we leave it in for a more compact pseudo-code, which is in Alg. 5. In the pseudo-code, $C_l(j)$ and $C_r(j)$ denote the left and right children of node j, respectively. The innermost **repeat** loop of Alg. 4 is not needed in our version.

Algorithm 5: OUR HIERARCHICAL FUSION

for each $i = 1, ..., k$ **do**
  for each $p \in P$ **do**
    $x_p^i = i$
for each $i = 1, ..., k$ **do**
  $ready(i)$:= true
for each $i = k+1, ..., 2k$ **do**
  $ready(i)$:= false
// perform fusions
for each $j = k+1, k+2, ..., r$ **in parallel do**
  $a := C_l(j)$
  $b := C_r(j)$
  if $ready(a)$ = true **and** $ready(b)$ = true
    $\boldsymbol{x}^j := \operatorname{argmin}_{\boldsymbol{x} \in Fuse(\boldsymbol{x}^a, \boldsymbol{x}^b)} f(\boldsymbol{x})$
    $ready(j)$= true
return $\boldsymbol{x}^r$

3.3. Optimality Bounds

We now prove that our algorithm is within a factor of $O(\log k)$ from the optimal solution. The proof also gives insights on choosing a good order of labels for the label tree, and for developing an iterative hierarchical fusion. Our proof is derived using a strategy similar to that in [7, 9].

Theorem 1. *Let f_{pq} be Potts pairwise term. Then the solution found with Alg. 5 is within a factor of $O(\log k)$ from the global optimum.*

Proof. Let $\boldsymbol{x}^*$ be the global minimum of Potts energy, and $\hat{\boldsymbol{x}}$ be the solution returned by Alg. 5 . As above, we assume the leaf nodes are numbered $1, 2, ..., k$ and non-leaf nodes have larger indexes, with the root being the largest. For each non-leaf node j in our label tree, we solve a fusion problem that involves a subset of labels from $\mathcal{L}$. Let us call the subset at node j as $\mathcal{L}_j$. For a leaf node i, $\mathcal{L}_i = \{i\}$.

For a given $\mathcal{L}_j$, Let P^j be the set of all pixels have labels in $\mathcal{L}_j$ in the optimal solution. That is

$$P^j = \{p \in P \mid x_p^* \in \mathcal{L}_j\}.$$

Consider the top three levels of the hierarchy, illustrated in Fig. 4. Let a, b be the left right children of the root node r. Further down, c, d and e, f are the children of nodes a and b, respectively. Recall that for node j, we defined $\boldsymbol{x}^j$ as the result of fusing two children labelings of j. We illustrate $\boldsymbol{x}^r = \hat{\boldsymbol{x}}, \boldsymbol{x}^a, \boldsymbol{x}^b, \boldsymbol{x}^c, \boldsymbol{x}^d, \boldsymbol{x}^e, \boldsymbol{x}^f$ with distinct colors.

In the final step of our algorithm, we find the optimal fusion of $\boldsymbol{x}^a$ and $\boldsymbol{x}^b$ and set it to be the final labeling $\boldsymbol{x}^r = \hat{\boldsymbol{x}}$. Let $\mathcal{L}_a, \mathcal{L}_b, P^a$ and P^b be as defined above. $P^a \cap P^b = \emptyset$ are illustrated in Fig. 4. Define a new labeling $\boldsymbol{x}^{ab}$:

$$x_p^{ab} = \begin{cases} x_p^a & \text{if } p \in P^a \\ x_p^b & \text{if } p \in P^b \end{cases}$$

Labeling $\boldsymbol{x}^{ab}$ is illustrated in Fig. 4, using colors to specify which parts come from $\boldsymbol{x}^a$ and which from $\boldsymbol{x}^b$. Since $\boldsymbol{x}^{ab} \in Fuse(\boldsymbol{x}^a, \boldsymbol{x}^b)$, but not necessarily an optimal one, we get:

$$f(\hat{\boldsymbol{x}}) \leq f(\boldsymbol{x}^{ab}) \tag{4}$$

For a subset of image pixels A, define $f(\boldsymbol{x} \mid A)$ as the energy terms that depend only on pixels in set A:

$$f(\boldsymbol{x} \mid A) = \sum_{p \in A} f_p(x_p) + \sum_{\{p,q\} \in E : p \in A, q \in A} f_{pq}(x_p, x_q)$$

Let $A \cap B = \emptyset$ be pixel subsets. Define $f(\boldsymbol{x} \mid A, B)$ as the energy terms that depend on both pixels in A and in B:

$$f(\boldsymbol{x} \mid A, B) \quad = \sum_{\substack{\{p,q\} \in E : p \in A, q \in B \\ or\ q \in A, p \in B}} f_{pq}(x_p, x_q)$$

Since for any pair $\{p, q\} \in E$ s.t. $p \in P^a$ and $q \in P^b$ we have that $x_p^* \neq x_q^*$ and $x_p^{ab} \neq x_q^{ab}$, we have $f(\boldsymbol{x}^* | P^a, P^b) = f(\boldsymbol{x}^{ab} | P^a, P^b)$. Using this fact and splitting $f(\boldsymbol{x}^{ab})$ into three parts (terms that depend only on P^a, only on P^b, and on both P^a and P^b) we get:

$$f(\boldsymbol{x}^{ab}) = f(\boldsymbol{x}^{ab} | P^a) + f(\boldsymbol{x}^{ab} | P^b) + f(\boldsymbol{x}^* | P^a, P^b). \tag{5}$$

Now we need to bound terms $f(\boldsymbol{x}^{ab} | P^a)$ and $f(\boldsymbol{x}^{ab} | P^b)$. Let c, d be the left and right children of node a. At node a,

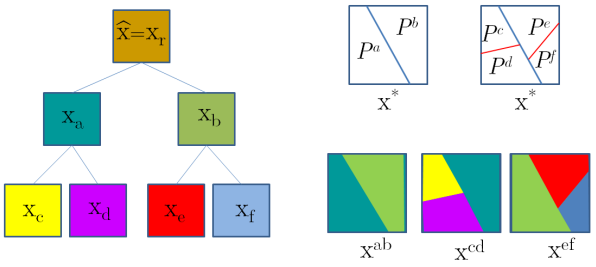

Figure 4. Illustration for theorem 1. Best viewed in color.

we find the optimal fusion $\boldsymbol{x}^a$ of labellings $\boldsymbol{x}^c$ and $\boldsymbol{x}^d$. Note that by definition, $P = P^c \cup P^d \cup P^b$ and P^c, P^d, P^b do not overlap, see Fig. 4. Define labeling $\boldsymbol{x}^{cd}$, illustrated in Fig. 4, as:

$$x_p^{cd} = \begin{cases} x_p^c & \text{if } p \in P^c \\ x_p^d & \text{if } p \in P^d \\ x_p^a & \text{if } p \in P^b \end{cases}$$

Labeling $\boldsymbol{x}^{cd} \in Fuse(\boldsymbol{x}^c, \boldsymbol{x}^d)$, but not necessarily an optimal one. Therefore,

$$f(\boldsymbol{x}^a) \leq f(\boldsymbol{x}^{cd}). \tag{6}$$

Splitting $f(\boldsymbol{x}^{cd})$ over sets P^c, P^d , and P^b we get:

$$\begin{aligned} f(\boldsymbol{x}^{cd}) &= f(\boldsymbol{x}^{cd}|P^c) + f(\boldsymbol{x}^{cd}|P^d) + f(\boldsymbol{x}^{cd}|P^c, P^d) \\ &+ f(\boldsymbol{x}^{cd}|P^a, P^b) + f(\boldsymbol{x}^{cd}|P^b). \end{aligned} \tag{7}$$

Using Eqs. 6, 7, and the fact that $f(\boldsymbol{x}^{cd}|P^b) = f(\boldsymbol{x}^a|P^b)$:

$$f(\boldsymbol{x}^a \mid P^a) + f(\boldsymbol{x}^a \mid P^a, P^b) \leq \tag{8}$$

$$f(\boldsymbol{x}^{cd} \mid P^a) + f(\boldsymbol{x}^{cd} \mid P^a, P^b) = f(\boldsymbol{x}^{cd}|P^c) + \tag{9}$$

$$f(\boldsymbol{x}^{cd}|P^d) + f(\boldsymbol{x}^{cd}|P^c, P^d) + f(\boldsymbol{x}^{cd}|P^a, P^b) \tag{10}$$

Using Eq. 8 and the following facts,

$$f(\boldsymbol{x}^{ab}|P^a) = f(\boldsymbol{x}^a|P^a)$$

$$f(\boldsymbol{x}^{cd}|P^c) = f(\boldsymbol{x}^c|P^c), f(\boldsymbol{x}^{cd}|P^d) = f(\boldsymbol{x}^d|P^d)$$

$$f(\boldsymbol{x}^{cd}|P^a, P^b) = f(\boldsymbol{x}^*|P^a, P^b)$$

$$f(\boldsymbol{x}^{cd}|P^c, P^d) = f(\boldsymbol{x}^*|P^c, P^d)$$

we get:

$$\begin{aligned} f(\boldsymbol{x}^{ab} \mid P^a) &= f(\boldsymbol{x}^a \mid P^a) \\ &\leq f(\boldsymbol{x}^c|P^c) + f(\boldsymbol{x}^d|P^d) \\ &+ f(\boldsymbol{x}^*|P^a, P^b) + f(\boldsymbol{x}^*|P^c, P^d) \end{aligned} \tag{11}$$

Defining $\boldsymbol{x}^{ef}$ analogously to $\boldsymbol{x}^{cd}$, and arguing similarly to Eq. 6 through 11 we get:

$$\begin{aligned} f(\boldsymbol{x}^{ab} \mid P^b) &= f(\boldsymbol{x}^b \mid P^b) \\ &\leq f(\boldsymbol{x}^e|P^e) + f(\boldsymbol{x}^f|P^f) \\ &+ f(\boldsymbol{x}^*|P^a, P^b) + f(\boldsymbol{x}^*|P^e, P^f) \end{aligned} \tag{12}$$

Combining Eq. 4, 5, 11, and 12 we get

$$\begin{aligned} f(\hat{\boldsymbol{x}}) &\leq f(\boldsymbol{x}^c|P^c) + f(\boldsymbol{x}^d|P^d) + f(\boldsymbol{x}^e|P^e) \\ &+ f(\boldsymbol{x}^f|P^f) + f(\boldsymbol{x}^*|P^c, P^d) \\ &+ f(\boldsymbol{x}^*|P^e, P^f) + 3f(\boldsymbol{x}^*|P^a, P^b) \end{aligned} \tag{13}$$

We can apply Eq. 13 recursively until we reach the bottom of the hierarchy. For nodes v at the deepest hierarchy level we have $f(x^v|P^v) = f(x^*|P^v)$, since $\forall p,\ x_p^v = v$.

The boundary between each pair of neighboring regions P^v and P^w will first appear at some level of the hierarchy where it will be counted once. Then this boundary will appear at most twice at each of the deeper levels of hierarchy. For example, the boundary between P^a and P^b appears at the highest level where it is counted once. It appears at the next lower level and is counted twice there. A boundary may be counted less than twice at the lower levels of the hierarchy if the hierarchy tree is not complete.

Using the facts above, and also the fact that the depth of hierarchy is $\log_2 k$, we apply Eq. 13 recursively, to get:

$$\begin{aligned} f(\hat{\boldsymbol{x}}) &\leq \sum_{i \in \{1,2,\ldots,k\}} f(\boldsymbol{x}^*|P^i) \\ &+ 2\log_2(k) \sum_{i,j \in \{1,2,\ldots,k\}} f(\boldsymbol{x}^*|P^i, P^j) \\ &\leq 2\log_2(k) \cdot f(x^*). \end{aligned} \tag{14}$$

□

From the proof we observe that the number of times a boundary is added to the final bound is inversely proportional to its depth. In particular, the heaviest counted boundary is that between pixels of the optimal solution that are assigned labels in the left and right subtrees of the root node r, illustrated in blue in Fig. 4, top right. Conversely, the boundary between pixels that have labels involved in the fusion at the deepest level are counted only once. Therefore, ideally, at the deepest level we should put nearby labels that have the longest boundary in the optimal solution.

Of course, we do not know the optimal solution before hand. We can use statistics from an inexpensive approximation, such as based on unary terms only. In many problems, such as stereo, it is reasonable to assume that labels with smaller difference are more likely to have a longer boundary. Therefore, arranging labels as in Fig. 3 is meaningful.

Note that we can view one iteration over all labels of the standard expansion algorithm as an h-fusion over a degenerate label tree, see Fig. 5. Here, the initial labeling for the expansion is assumed to be a constant labeling with all pixels assigned to label 1. Thus the expansion algorithm has a factor of $O(k)$ approximation after one iteration, which is much worse than our factor $O(\log_2 k)$ approximation. This helps to explain why we get better results after one iteration of our h-fusion algorithm, compared to that of expansion.

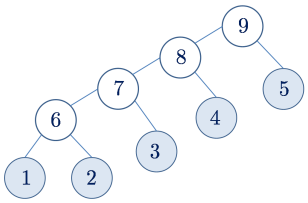

Figure 5. Expansion tree for a problem with 5 labels. The height of the tree is linear in the number of labels, leading to a much worse approximation bound of $O(k)$ after one iteration.

3.4. Iterative Hierarchical Fusion

Our Alg. 5, based on the original hierarchical fusion algorithm in [9, 19] is not iterative. It is not immediately obvious how to make it iterative, since it does not even make use of any initial solution, unlike most iterative algorithms.

To make it iterative, we make the following observation. Let $\hat{\boldsymbol{x}}$ be the solution obtained by Alg. 5. Let

$$E' = \{\{p,q\} \in E \mid \hat{\boldsymbol{x}}_p \neq \hat{\boldsymbol{x}}_q\}$$

We can execute Alg. 5 for a new energy function g, which is defined as follows. The unary terms are left unchanged:

$$g_p(x_p) = f_p(x_p),$$

and the pairwise terms are changed as:

$$g_{pq}(x_p, x_q) = \begin{cases} f_{pq}(x_p, x_q) & \text{for } \{p,q\} \in E \setminus E' \\ 0 & \text{for } \{p,q\} \in E' \end{cases} \tag{15}$$

We now prove the following proposition:

Proposition 1. *Let $\boldsymbol{x}^f = \hat{\boldsymbol{x}}$ the the labeling Alg. 5 returns when optimizing $f(\boldsymbol{x})$. Let $g(\boldsymbol{x})$ be as defined in Eq. 15, and let $\boldsymbol{x}^g$, be the labelling Alg. 5 returns when optimizing $g(\boldsymbol{x})$. Then $f(\boldsymbol{x}^g) \leq f(\boldsymbol{x}^f)$.*

Proof. First we show that $g(\boldsymbol{x}^g) \leq g(\boldsymbol{x}^f)$. This fact is not immediately obvious because Alg. 5 is not guaranteed to find a global optimum of $g(\boldsymbol{x})$. However, we can re-do the proof of Theorem 1, with f replaced by g, $\hat{\boldsymbol{x}}$ replaced by $\boldsymbol{x}^g$, and $\boldsymbol{x}^*$ replaced by $\boldsymbol{x}^f$. In all the equations, all the boundary terms, i.e terms of type $g(\boldsymbol{x}|P^a, P^b)$ are equal to zero now, because g is defined to be zero on the boundary of $\boldsymbol{x}^f$. The equivalent of Eq. 14 in our case becomes:

$$g(\boldsymbol{x}^g) \quad \leq \sum_{a \in \{1,2,\ldots,k\}} g(\boldsymbol{x}^f | P^a) = g(\boldsymbol{x}^f) \tag{16}$$

The rest of the proof is easy. Let $A \subset E$ and define

$$f(\boldsymbol{x} \mid A) = \sum_{\{p,q\} \in A \cap E} f_{pq}(x_p, x_q).$$

Image	k	expansion energy	expansion time	h-fusion energy	h-fusion time
tsukuba	16	1074813	0.552	1068527	0.294
venus	20	2329733	1.227	2319501	0.593
teddy	60	3961242	3.247	3906396	1.32
cones	60	4624147	3.207	4586747	1.216
sawtooth	20	2740786	0.987	2730520	0.499
bull	20	2028894	0.969	2018992	0.505
barn1	20	2590260	0.898	2582261	0.482

Figure 6. Result comparison after one iteration of the expansion algorithm vs. one iteration of our approach (h-fusion). Running times are in seconds.

Then $g(\boldsymbol{x}) = f(\boldsymbol{x}) - f(\boldsymbol{x} \mid E')$. From Eq. 16 and the fact that $f(\boldsymbol{x}^f|E') \geq f(\boldsymbol{x}^g|E')$ we get:

$$f(\boldsymbol{x}^g) \leq f(\boldsymbol{x}^f) - f(\boldsymbol{x}^f|E') + f(\boldsymbol{x}^g|E') \leq f(\boldsymbol{x}^f).$$

□

Thus Alg. 5 can be iterated until convergence by constructing new set E' after each iteration.

4. Experimental Results

We evaluate our h-fusion approach to Potts energy minimization on the problem of stereo correspondence. We use the images from Middlebury evaluation dataset [24, 25].

We used energy settings commonly reported for this dataset. Unless stated otherwise, for each experiment we used $w_{pq} = \lambda = 20$ for Potts energy in Eq. 1. We used the absolute difference matching cost for the unary terms. That is $f_p(x_p) = |L(p) - R(p - x_p)|$, where $L(p)$ is the intensity of pixel p in the left image, $R(p - x_p)$ is the intensity of pixel p shifted by disparity x_p in the right image.

All experiments were run on a desktop machine with 16GB of RAM and a quad-core processor clocked at 3.60GHz. OpenMP was used for parallel implementation. Our algorithm does not require initialization. Expansion algorithm was initialized by setting all pixels to label 1.

Table 6 compares the results of our h-fusion optimization after one iteration vs. the performance of the expansion algorithm after one iteration. Our energy is always a little better and our running time is from 1.9 to 2.6 times faster.

Table 7 shows the same comparison but after four iterations of each algorithm. While our energies do improve through iterations, now the expansion algorithm achieves better energies. Our speedup factors are from 1.5 to 2.2, a little less because the the first iteration of the expansion algorithm is more expensive compared to other iterations.

Instead of performing h-fusion iteratively, we also experimented with performing the swap move algorithm [7], in parallel, after the first iteration of h-fusion. Swap moves are simple to implement in parallel fashion. We take the output of h-fusion, repeatedly group labels in pairs if they

Image	k	expansion energy	expansion time	h-fusion energy	h-fusion time
tsukuba	16	1051546	2.06	1064765	0.904
venus	20	2302235	3.83	2314932	2.09
teddy	60	3856692	9.49	3895149	4.42
cones	60	4534576	8.85	4572954	4.01
sawtooth	20	2708293	2.85	2724275	1.67
bull	20	2008368	3.03	2017643	1.822
barn1	20	2563738	2.68	2578775	1.73

Figure 7. Result comparison after four iterations of the expansion algorithm vs. four iterations of our approach (h-fusion). Running times are in seconds.

Image	k	expansion energy	expansion time	h-fusion energy	h-fusion time
tsukuba	16	1051546	2.06	1052694	1.04
venus	20	2302235	3.83	2302392	1.64
teddy	60	3856692	9.49	3863141	3.63
cones	60	4534576	8.85	4539305	3.44
sawtooth	20	2708293	2.85	2709406	1.34
bull	20	2008368	3.03	2009213	1.61
barn1	20	2563738	2.68	2564296	1.56

Figure 8. Result comparison after four iterations of the expansion algorithm vs. one iteration of h-fusion followed by three iterations of the swap moves. Running times are in seconds.

share a common boundary (i.e. if the regions assigned to these labels have neighboring pixels), and run swap moves on these pairs of pixels in parallel. This process is repeated until all neighboring label pairs have been offered a chance to 'swap'. The results of finishing h-fusion with parallel swap move are in Table 8. Three iterations of swap moves were performed, for a total of 4 iterations (one with h-fusion and 3 with swap moves). Notice that the energies are now almost as good as that of the expansion algorithm. The speedup factors are also improved compared to the iterative h-fusion. They are now in the range from 1.8 to 2.6.

We also tested using swap moves in parallel from a random initialization, instead of starting with the results from h-fusion, even though such approach does not have any optimality guarantees. Perhaps not surprisingly, using just swap moves is not competitive with expansion and h-fusion both in terms of running time or accuracy. It took much longer to converge, even in parallel, to an inferior solution.

Given that the energies after one iteration of either expansion or our algorithm are already low for Potts model, there may be little additional benefit of performing further iterations, at least for the stereo application, see Fig. 9.

The running time of methods based on partial optimality can be sensitive to the the value of parameter λ. The larger the value, the longer is the running time since a smaller percentage of optimal pixels is found. We tested the running time of our approach vs. parameter λ on the *tsukuba* stereo pair. There is some, but not a large amount of sensitivity. Testing the interval of $\lambda = 10, 20, ..., 100$, we found that the running time increases from 0.26 to 0.36. Some sensitivity is expected because the running time of the graph-cut algorithm [6] depends on the value of max-flow, and the larger the λ, the larger is the value of max-flow.

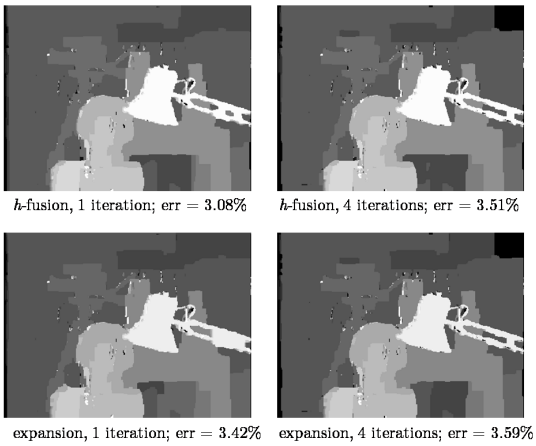

Figure 9. Disparity maps for *tsukuba* scene. Top row: results of h-fusion after one iteration and four iterations. Bottom row: results of expansion after one iteration and four iterations. There is no significant differences in the error percentages between these results. Thus one iteration may be sufficient for some applications.

We also tested the label tree where the leaf nodes are randomly, not consecutively, placed. This resulted in a small, but consistent worsening of the energy values.

5. Conclusions

We presented a parallel approach for optimizing Potts energies that has a factor of $O(\log k)$ approximation after one iteration over all labels. Even though the approach is parallel, the drawback is that it is more memory-intensive.

One interesting question to explore is its applicability to energies more general than Potts, such as with metric or semi-metric pairwise potentials. While the method described here is directly applicable to these more general energy functions, any optimality properties are lost because fusion steps are not guaranteed to be submodular.

References

[1] A. Agarwala, M. Dontcheva, M. Agrawala, S. Drucker, A. Colburn, B. Curless, D. Salesin, and M. Cohen. Interactive digital photomontage. pages 294 – 302, 2004. 1

[2] K. Alahari, P. Kohli, and P. H. Torr. Reduce, reuse & recycle: Efficiently solving multi-label mrfs. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008. 1

[3] K. Alahari, P. Kohli, and P. H. Torr. Dynamic hybrid algorithms for map inference in discrete mrfs. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(10):1846–1857, 2010. 1

[4] K. Alahari, P. Kohli, and P. H. Torr. Dynamic Hybrid Algorithms for MAP Inference in Discrete MRFs. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 32:1846–1857, 2010. 1

[5] D. Batra and P. Kohli. Making the right moves: Guiding alpha-expansion using local primal-dual gaps. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1865–1872. IEEE, 2011. 1

[6] Y. Boykov and V. Kolmogorov. An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision. *IEEE Transactions on Pattern Recognition and Machine Intelligence (TPAMI)*, 29(9):1124–1137, 2004. 2, 8

[7] Y. Boykov, O. Veksler, and R. Zabih. Fast Approximate Energy Minimization via Graph Cuts. *IEEE Transactions on Pattern Recognition and Machine Intelligence (TPAMI)*, 23(11):1222–1239, 2001. 1, 2, 5, 7

[8] Y. Boykov, O. Veksler, and R. Zabih. Optimizing multi-label mrfs by move making algorithms. In *Advances in Markov Random Fields for Vision and Image Processing, edited by A. Blake, P. Kohli and C. Rother*, pages 51–65. 2011. 1

[9] A. Delong, L. Gorelick, O. Veksler, and Y. Boykov. Minimizing energies with hierarchical costs. *International Journal of Computer Vision*, 100(1):38–58, 2012. 2, 3, 4, 5, 7

[10] P. F. Felzenszwalb, G. Pap, E. Tardos, and R. Zabih. Globally optimal pixel labeling algorithms for tree metrics. In *Computer Vision and Pattern Recognition, 2010. CVPR 2010. IEEE Conference on*, pages 3153–3160. IEEE, 2010. 1

[11] I. Gridchyn and V. Kolmogorov. Potts model, parametric maxflow and k-submodular functions. In *IEEE International Conference on Computer Vision, ICCV 2013, Sydney, Australia, December 1-8, 2013*, pages 2320–2327, 2013. 1

[12] J. Hays and A. A. Efros. Scene completion using millions of photographs. 26(3), 2007. 1

[13] J. H. Kappes, B. Andres, F. A. Hamprecht, C. Schnörr, S. Nowozin, D. Batra, S. Kim, B. X. Kausler, J. Lellmann, N. Komodakis, and C. Rother. A comparative study of modern inference techniques for discrete energy minimization problems. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1328–1335, 2013. 1

[14] V. Kolmogorov. Convergent Tree-Reweighted Message Passing for Energy Minimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 28(10):1568–1583, October 2006. 1

[15] V. Kolmogorov and C. Rother. Minimizing non-submodular functions with graph cuts—a review. *IEEE Transactions on Pattern Recognition and Machine Intelligence (TPAMI)*, 29(7), 2007. 3

[16] V. Kolmogorov and R. Zabih. What Energy Functions Can Be Optimized via Graph Cuts. *IEEE Transactions on Pattern Recognition and Machine Intelligence (TPAMI)*, 26(2):147–159, 2004. 2, 3

[17] N. Komodakis, G. Tziritas, and N. Paragios. Performance vs computational efficiency for optimizing single and dynamic mrfs: Setting the state of the art with primal-dual strategies. *Computer Vision and Image Understanding*, 112(1):14–29, 2008. 1

[18] I. Kovtun. Image segmentation based on sufficient conditions of optimality in np-complete classes of structural labelling problem. *Ukrainian. PhD thesis. IRTC ITS National Academy of Sciences, Ukraine*, 2004. 1

[19] M. P. Kumar and D. Koller. MAP estimation of semi-metric MRFs via hierarchical graph cuts. In *Conference on Uncertainty in Artificial Intelligence*, pages 313–320, June 2009. 2, 3, 4, 7

[20] V. Lempitsky, C. Rother, and A. Blake. Logcut-efficient graph cut optimization for markov random fields. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE, 2007. 2

[21] V. Lempitsky, C. Rother, S. Roth, and A. Blake. Fusion moves for markov random field optimization. *IEEE Transactions on Pattern Analysis and Machine Inference (TPAMI)*, 32:1392–1405, August 2010. 2, 3

[22] C. Rother, V. Kolmogorov, V. Lempitsky, and M. Szummer. Optimizing Binary MRFs via Extended Roof Duality. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2007. 3

[23] C. Rother, S. Kumar, V. Kolmogorov, and A. Blake. Digital tapestry. pages I: 589–596, 2005. 1

[24] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International journal of computer vision*, 47(1-3):7–42, 2002. 7

[25] D. Scharstein and R. Szeliski. High-accuracy stereo depth maps using structured light. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, volume 1, pages I–195. IEEE, 2003. 7

[26] A. Shekhovtsov. Maximum persistency in energy minimization. In *Conference on Computer Vision and Pattern Recognition,*, pages 1162–1169, 2014. 1

[27] P. Swoboda, B. Savchynskyy, J. Kappes, and C. Schnörr. Partial optimality via iterative pruning for the potts model. In *Scale Space and Variational Methods in Computer Vision*, pages 477–488. Springer Berlin Heidelberg, 2013. 1

[28] P. Swoboda, B. Savchynskyy, J. H. Kappes, and C. Schnörr. Partial optimality by pruning for map-inference with general graphical models. CVPR, 2014. 1

[29] R. Szeliski, R. Zabih, D. Scharstein, O. Veksler, V. Kolmogorov, A. Agarwala, M. Tappen, and C. Rother. A Comparative Study of Energy Minimization Methods for Markov Random Fields. In *European Conference on Computer Vision (ECCV)*, pages 16–29, 2006. 1, 2

[30] O. Veksler, Y. Boykov, and P. Mehrani. Superpixels and supervoxels in an energy optimization framework. In *Computer Vision–ECCV 2010*, pages 211–224. Springer Berlin Heidelberg, 2010. 2

[31] Y. Zhang, R. Hartley, J. Mashford, and S. Burn. Superpixels via pseudo-boolean optimization. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 1387–1394. IEEE, 2011. 2