Dynamic Programming for Approximate Expansion Algorithm

Olga Veksler*

University of Western Ontario
London, Canada
olga@csd.uwo.ca
http://www.csd.uwo.ca/ olga/

Abstract. Expansion algorithm is a popular optimization method for labeling problems. For many common energies, each expansion step can be optimally solved with a min-cut/max flow algorithm. While the observed performance of max-flow for the expansion algorithm is fast, its theoretical time complexity is worse than linear in the number of pixels. Recently, Dynamic Programming (DP) was shown to be useful for 2D labeling problems via a "tiered labeling" algorithm, although the structure of allowed (tiered) is quite restrictive. We show another use of DP in a 2D labeling case. Namely, we use DP for an approximate expansion step. Our expansion-like moves are more limited in the structure than the max-flow expansion moves. In fact, our moves are more restrictive than the tiered labeling structure, but their complexity is linear in the number of pixels, making them extremely efficient in practice. We illustrate the performance of our DP-expansion on the Potts energy, but our algorithm can be used for any pairwise energies. We achieve better efficiency with almost the same energy compared to the max-flow expansion moves.

1 Introduction

Expansion algorithm [1] has been used for a variety of pixel labeling problems arising in vision and graphics [2–6]. In a pixel labeling problem the goal is to assign a label to each image pixel. In general, pixel labeling problems are NP-hard [7], but can be solved efficiently in some special cases [8–12].

Expansion algorithm is an approximate optimization method with certain optimality guarantees [1]. It works by iteratively visiting labels α and performing an α-expansion move on that label. An expansion move finds an optimal subset of pixels to switch to label α. Each expansion step is performed with a min-cut/max-flow algorithm [13]. There is a max-flow algorithm [14] that was specifically designed for vision problems and it performs very efficiently in practice for many instances. Nevertheless the best known worst-case time complexity of max-flow is worse than linear in the number of image pixels.

* This research was supported,in part, by NSERC, CFI, and ERA grants.

Recently, a "tiered labeling" [12] algorithm, based on Dynamic Programming (DP) was proposed for labeling of 2D scenes. This algorithm finds the exact global minimum, but only for scenes with a "tiered" label layout. In particular, the scene is assumed to be divided by two simple horizontal curves into three main tiers, top, middle, and bottom. The top and bottom tiers receive a fixed label, "T" and "B", respectively. The middle tier can be further subdivided into tiers by vertical straight lines, and each sub-tier has a choice of one out of k labels. This tiered scene layout is rather limited, but an optimal labeling can be found in $O(kn^{1.5})$ time for a square image of n pixels and the Potts energy model. Most importantly, [12] shows that dynamic programming can be useful for labeling 2D scenes, whereas traditionally, in computer vision, DP was applied mostly to one-dimensional or low treewidth structures [15].

Inspired by tiered labeling, we propose an approximate expansion algorithm based on fast DP. Just as in tiered labeling, the main idea of our DP-expansion moves is to allow only certain structure layout inside the image to switch to label α. This is in contrast to the classical α-expansion, that allows an arbitrary subset of pixels to switch to α. By limiting the structure of the subsets that can switch to α, we are modifying the problem to make it suitable for dynamic programming. In fact, our moves are even more restrictive than the "tiered labeling" structure, but their complexity is linear in the number of pixels, making them extremely efficient in practice. Our moves are based on the observation that instead of expanding on the optimal subset of pixels in a single step, it maybe sufficient, in practice, although clearly suboptimal, to find an expansion region where only the left (or right, or top, or bottom boundaries) are accurate. This allows linear efficiency in the number of pixels. To recover accurate boundaries in all directions, we apply our moves iteratively.

Like tiered labeling, due to DP, we are limited to pairwise energies with a 4 or 8 connected neighborhood structure. This is a clear limitation, but longer-range interactions are seldomly used. In addition to speed, a clear advantage of our DP-expansion moves over the max-flow expansion moves is that, like tiered labeling, we can handle arbitrary pairwise terms, again, due to DP. In fact, due to the simplicity of our moves, we can handle arbitrary terms with the same complexity as Potts pairwise terms, i.e. $O(n)$, where n is the number of pixels.

We illustrate the performance of our DP-expansion on the Potts energy, but our algorithm can be used for any pairwise energies. We achieve better efficiency with almost the same energy compared to the max-flow expansion moves.

This paper is organized as follows. We review the most related work in Section 2. We motivate and explain our DP-expansion moves in Section 3. We describe our efficient optimization algorithm in Section 4, present experimental results in Section 5, and conclude with a short discussion in Section 6.

2 Related Work

There has been a lot of interest in vision community to develop efficient optimization methods for pixel labeling problems, see a survey in [16]. The popular methods evaluated in [16] include the expansion and swap algorithms [7], loopy

belief propagation (LBP) [17], and sequential tree re-weighted message passing (TRW-S) [18]. One of the conclusions of [16] is that the expansion algorithm performs best in terms of speed and accuracy for Potts energies.

Since our optimization is based on making approximate expansion moves, our work is related to the general direction of move-making algorithms. In addition to the expansion and swap algorithms [7], a number of new move making algorithms have been developed recently. In [19], they develop a fusion move that instead of expanding on a fixed label α, can expand on a different but single fixed label for each pixel in a move, thereby "fusing" the current solution and a new proposal solution together. In [20, 21], range moves that are designed specifically for energies tuned to truncated convex pairwise terms are developed. These moves, unlike the previous ones, are multi-label, they allow each pixel a choice of more than one label to switch to. In [22] they develop multi-label moves for more general energy functions. The approach in [23] also uses multi-label moves for fast approximate optimization of a tractable (submodular) multi-label energies that can, in fact, be exactly optimized by such methods as [10], but the time and space complexities of the exact method are heavy.

Another line of related methods are those optimization techniques that limit the structure of the allowed labellings. Tiered labeling [12], which inspired our work, uses a limited scene layout and relies on DP for optimization. Another very related approach is [24], where they use move-making algorithm based on tiered labeling for approximate optimization of more general energies. Their time complexity is rather large, since using tiered labeling for a single move has cost higher than linear in the number of pixels. We construct a simpler move than that in [24], but our efficiency is much better. Another related method is [25]. In [25] they propose a min-cut based approach for a five-label energy function with certain label structure layout. This layout is motivated by the application of indoor geometric scene labeling. The approach in [25] is approximate, unlike [12], even though for a four-connected neighborhood system tiered labeling layout is more general than that in [25]. Recently, [26] showed how to optimize the energy in [25] globally an optimally with DP. Another DP-based optimization method on 2D domains is [27], although their objective function is suitable only for object/background segmentation, not a more general scene labeling problem.

Finally, since our goal is to speed-up the expansion algorithm, there are related methods in this direction as well, which significantly improve efficiency of expansion. In [28], they attain speed-ups by considering the dual formulation. In [29], they exploit several techniques, such as recycling information from previous instances, clever intialization, etc. In [30], they find the order of iterations over labels likely to lead to the fastest energy decrease. Our approach explores a previously unexplored direction, namely we limit the allowed move structure.

3 DP-expansion Moves

In this section, we explain and motivate our DP-expansion moves.

3.1 Energy Function

In a pixel labeling problem we are given a set of pixels $\mathcal{P}$ and a set of labels $\mathcal{L}$. We need to assign a label from the label set to each image pixel $p \in \mathcal{P}$. Let l_p be the label assigned to pixel p and l be the collection of all label assignments for all pixels in $\mathcal{P}$. The goal is to find a labeling minimizing the following energy:

$$E(l) = \sum_{p \in P} D_p(l_p) + \sum_{(p,q) \in \mathcal{N}} V_{pq}(l_p, l_q), \tag{1}$$

where $D_p(l_p)$ and $V_{pq}(l_p, l_q)$ are unary and pairwise energy terms, and $\mathcal{N}$ is a neighborhood system on $\mathcal{P}$. We assume $\mathcal{N}$ is 4-connected with ordered pairs (p, q) such that p is directly to the left or above q. Our algorithm can be easily extended to the 8-connected neighborhood that includes diagonal edges with almost no increase in computational time. This is unlike max-flow expansion, where including diagonal links noticeably increases computational time.

The unary term $D_p(\alpha)$ is the cost for assigning label α to pixel p. It usually depends on the value of pixel p in the observed image. The pairwise term $V_{pq}(\alpha, \beta)$ specifies the cost for assigning labels α and β to pixels p and q. This term is usually used to enforce some smoothness constraints on a low-cost labeling. Unlike most work on optimizing Eq. (1), we impose no constraints on unary and pairwise terms. In particular, $V_{pq}(\alpha, \beta)$ does not have to be submodular [31].

3.2 Review of Tiered Labeling

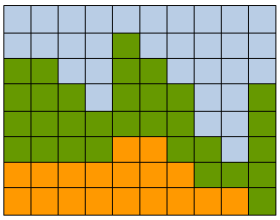


Fig. 1. Simplified version of tiered labeling: the middle tier is only allowed one label.

We first review the tiered labeling algorithm [12] that inspired our approach. Let T and B be the special labels allowed only for the top and bottom tiers and $\mathcal{M}$ be the set of labels allowed for the middle tier. Therefore $\mathcal{M} = \mathcal{L} \backslash \{T, B\}$. Let $p = (i, j)$ be the pixel in row i column j and $l_{i,j}$ be its label. A labeling is tiered if in each image column, there exists integers t and b s.t. all pixels strictly above t are labeled as T, all pixels below b are labeled as B, and all other pixels take one of the labels from the label set $\mathcal{M}$. That is for each column j, $\exists t \leq b$ s.t. $\forall i < t,\ \ l_{i,j} = T$, $\forall i \geq b,\ \ l_{i,j} = B$, and $\forall t \leq i < b,\ \ l_{i,j} \in \mathcal{L}$. Notice that if $t = b$, then there is no middle tier label in column j, only labels T and B. This

property is important for our DP-expansion moves. A simplified tiered structure, where the middle tier has only one (green) label, is illustrated in Figure 1.

Similar to [24], we can use tiered labeling to find an approximate α-expansion move in the following manner. Let l' be the current labeling. The bottom and the top tiers will correspond to the pixels that do not switch their label in the α-expansion move. That is for each image pixel p, labels B and T correspond to the old label of pixel p, namely l'_p. We allow only the label α for the middle tier, that is $\mathcal{M} = \{\alpha\}$. The optimal tiered labeling under such construction corresponds to the optimal set of pixels with restricted structure to switch to α. Namely, we can find the best contiguous set of pixels in each column to switch. Some columns can choose not to switch any pixels to α. If no column makes a switch, the optimal α-expansion is empty and all pixels retain old labels. This is important for ensuring that the energy never goes up in a move-making algorithm.

This approach is clearly inferior to the max-flow expansion that can find an unrestricted subset of pixels to switch to α. However, the combined result of repeated applications of this move can find rather complex regions to switch to α, see [24][1]. The move described modifies the top and bottom boundaries of the region to switch to α. We can improve the accuracy further by repeating this move in the orthogonal direction, by turning the image by $\pi/2$, finding the optimal contiguous subset of pixels in each row to switch to α.

The approach outlined above (basically following [24]) is expensive. Tiered labeling complexity is $O(n^2m)$, where n is the number of rows and m is the number of columns in the image. Tiered labeling works by collapsing each image column to a single state. Each column has two possible break points t and b, so the number of possible states is $O(n^2)$. The algorithm in [12] develops a clever speed-up strategy that allows to reduce the DP complexity from $O(n^4m)$ to just $O(n^2m)$. Still, the complexity is worse than linear.

3.3 DP-expansion

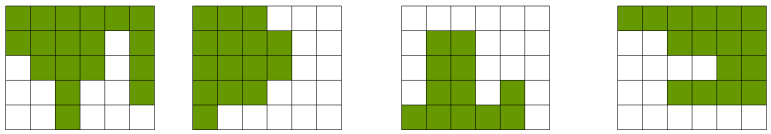
To speed up, we need to reduce the number of states in each column further. Observe that in the approach outlined in Section 3.2 a single move is capable of "carving" out a relatively complex upper and lower boundaries of a region to switch to α, leaving left and right boundaries relatively simpler. Being able to carve out complex boundaries is clearly important for faithfully recovering object boundaries. However, it may be sufficient to work on one boundary at a time. Our main idea is to develop an even simpler move. Our move allows a complex boundary in only one direction. The trade-off is that the state space is reduced and we can achieve a linear time complexity DP.

This motivates the following DP-based moves. We have four different move types, top-anchored, left-anchored, bottom-anchored, and right-anchored. For the top-anchored move, if any pixels in column j decide to switch their labels to α, they must form a contiguous set. Furthermore this set must contain the topmost, row 0, pixel. That is, for each column j, only pixel subsets of form

[1] Actually, [24] introduced more powerful moves than we just described, but we believe these moves will also work almost as well as those in [24].

$\{(i,j)|0 \leq i < b\}$ for some $b \leq n$ are allowed to switch to label α. Here n is the number of rows in the image. This move is illustrated in Figure 2(a). It allows recovering of a region with a complex boundary on the bottom.

The other moves, illustrated in Figure 2(b,c,d) are similar. They require any switching to α start on the left, bottom, right of the image, respectively, and they are capable of recovering a region with complex boundaries on the right, top, and left, respectively.



(a) top-anchored (b) left-anchored (c) bottom-anchored (d) right-anchored

Fig. 2. Four DP-expansion moves. The old label is white and the new label α is green. Top-anchored move in (a): if any pixels in a column switch to label α, they must form a contiguous region and the top pixel must switch to α. Left-, bottom-, and right-anchored moves require switching to α start at the left, bottom, right of the image.

We show how to implement these moves in time linear in the number of pixels in Section 4. While very fast, these moves would not be powerful if they were applied only to the whole image, since any switch to a new label must start at the image top (or bottom, left, right). We found it effective to apply these moves to image blocks of different sizes in random order.

When applying a DP-expansion move to an image block, only the unary and pairwise terms that depend on pixels entirely within the block are involved. However, the pairwise terms V_{pq} that involve one pixel in the block and one pixel outside the block are also effected by the changes within the block, and must be addressed to ensure the energy strictly decreases. Like previous work [1], the fix is simple. Let l' be the old labeling. Let V_{pq} have p inside the block and q outside the block. We add $V_{pq}(l'_p, l'_q)$ to $D_p(l'_p)$ and $V_{pq}(\alpha, l'_q)$ to $D_p(\alpha)$.

4 Fast Optimization

We now explain how to do fast optimization for the DP-expansion moves defined in Sec. 3.3. Even though most of our moves are performed on image blocks, it is easier to explain how optimization works assuming it is done on the whole image, avoiding the extra sub-indexing notation. The fix that needs to be done when performing the moves on image blocks was explained at the end of Section 3.3.

Our fast DP derivation closely follows the tiered labeling speed-ups in [12]. However, our speed-up of DP is simpler since we have a less complex state space. We explain optimization for the case of top-anchored moves. The other cases are handled by appropriately rotating the image.

For most derivations in this paper, for pixel $p = (i,j)$ it is convenient to denote D_p as D_{ij}, and l_p as $l_{i,j}$. Also, if $p = (i,j)$ and $q = (i+1,j)$, that is a vertical term, then we denote V_{pq} as $V^v_{i,j}$. If $p = (i,j)$, and $q = (i,j+1)$, that is a

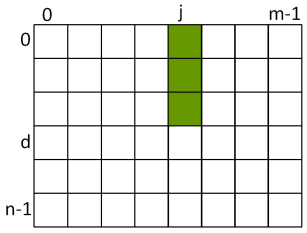


Fig. 3. Site j in state d. Pixels above row d (in green) are labeled as α in column j. Pixels below and including row d (in white) in column j retain the old label.

horizontal term, we denote V_{pq} as $V^h_{i,j}$. Let image have n rows, indexed from 0 to $n-1$, and m columns, indexed from 0 to $m-1$, see Fig. 3. Let the old labeling be l' and we want to find the best top-anchored α expansion move. That is, in each column, we want to find the best contiguous subset of pixels to switch to α, provided that any switch in column j, must begin at row 0, i.e. at pixel $(0, j)$.

4.1 Reduction to Dynamic Programming

We represent each image column j with a single site j, visually "collapsing" it vertically. Thus there are m sites indexed by $0, 1..., m-1$. The state of site j is denoted by s_j. The number of possible states s_j corresponds to the number of possible labellings in column j. A possible state for column j can be represented with one integer $0 \leq d \leq n$, which is decoded as follows. If $s_j = d$, then all pixels above row d are labeled as α and all pixels at row d and below retain their old label. See Figure 3 for illustration. If $d = 0$ than no pixels in column j get label α. The total number of states for column j is $n+1$. Let s be the assignment of states for all sites.

Let l be a labeling satisfying the top-anchored move constraints and s be the corresponding assignment of states of the one-dimensional problem. The energy function in Equation (1) can be written as:

$$E(s) = \sum_{j=0}^{m-1} U_j(s_j) + \sum_{j=0}^{m-2} B_j(s_j, s_{j+1}), \tag{2}$$

where $U_j(s_j)$ adds up all the unary terms and pairwise terms of the energy in Equation (1) that are contained entirely in column j, and $B_j(s_j, s_{j+1})$ adds up all the pairwise terms of the energy in Equation (1) that rely on exactly one pixel from column j and one pixel from column $j+1$. Let us define a labeling $l^{j,d}$ as follows. Pixels in columns other than j can have any labels. Pixels in column j and rows 0 to $d-1$ have label α. Pixels in column j and rows d to $n-1$ have the same label as they have in the old labeling l'. Then

$$U_j(s_j) = \sum_{i=0}^{n-1} D_{ij}(l_{i,j}^{j,s_j}) + \sum_{i=0}^{n-2} V_{i,j}^{v}(l_{i,j}^{j,s_j}, l_{i+1,j}^{j,s_j}), \tag{3}$$

and

$$B_j(s_j, s_{j+1}) = \sum_{i=0}^{n-1} V_{i,j}^{h}(l_{i,j}^{j,s_j}, l_{i,j+1}^{j+1,s_{j+1}}). \tag{4}$$

As in [12], any U_j and B_j can be computed in $O(1)$ time with precomputed cumulative sums of data terms, horizontal and vertical terms separately in each column. This is like the well known integral image technique [32], only applied to columns and therefore slightly simpler.

The standard DP for optimizing the energy in Equation (2) proceeds as follows. We build look-up tables, E_j, for $0 \leq j < m$. $E_j(t)$ is the cost of the best assignment to the first $j+1$ states, namely states $(0, 1, ..., j)$ where the last state is fixed to have value t, i.e. $s_{j+1} = t$. E_j are computed in order of increasing j using the following recurrence, and for all $t \in \{0, 1, ..., n\}$:

$$E_0(t) = U_0(t), \tag{5}$$

$$E_j(t) = U_j(t) + \min_{t'} \left[E_{j-1}(t') + B_{j-1}(t', t) \right]. \tag{6}$$

We also compute the optimum previous state in a look-up table $P_j(t)$. It is used to trace back the optimal sequence of states, starting from the last state and going backwards. That is the optimal state for the last site is computed with $s_{m-1}^* = \text{argmax}_t E_{m-1}(t)$. After that, the previous optimal state is computed with $s_{j-1}^* = P_j(s_j^*)$, where s_j^* is the optimal state for site j.

For a fixed j, we have to compute $E_j(t)$ for $n+1$ possible values of t . For each fixed j and t, we need to search over all previous states t', and there are $n+1$ of them. Therefore, computing $E_j(t)$ is $O(n)$ for a fixed j and t. Computing $E_j(t)$ is $O(n^2)$ for all $n+1$ values of t, and, finally, computing $E_j(t)$ for all m values of j is $O(n^2 m)$, which is too expensive. We explain how to speed it up to $O(nm)$, i.e. linear in the number of pixels in Section 4.2.

4.2 Fast Dynamic Programming

We now explain how, for a fixed j and t, compute $E_j(t)$ in $O(1)$ time instead of the straightforward $O(n)$ time. This will result in the total time complexity $O(nm)$, i.e. linear in the number of pixels.

The expensive part of $E_j(t)$ computation is the search over the previous states t'. Let us give this part its own name $F_j(t)$:

$$F_j(t) = \min_{t'} \left[E_{j-1}(t') + B_{j-1}(t', t) \right]. \tag{7}$$

We can break optimization of $F_j(t)$ in two cases, for $t' \leq t$ and $t' > t$:

$$F_j(t) = \min \left\{ T_j(t), L_j(t) \right\}, \tag{8}$$

where

$$T_j(t) = \min_{t' \leq t} [E_{j-1}(t') + B_{j-1}(t', t)] \tag{9}$$

and

$$L_j(t) = \min_{t' > t} [E_{j-1}(t') + B_{j-1}(t', t)] \tag{10}$$

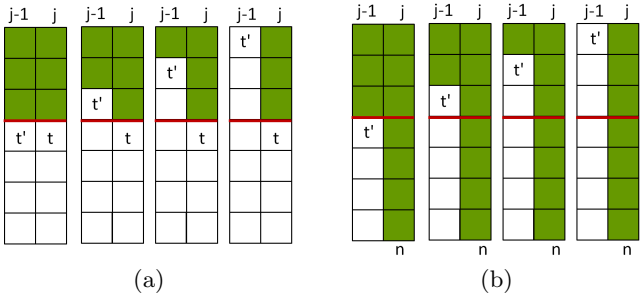


Fig. 4. Case 1: Minimization over $t' \leq t$. (a) The state of column j is fixed to t, which means that all pixels above row t in column j take new label α. All possible choices for t' in column $j-1$ are illustrated. We have to find the smallest cost choice. Below the red line, all four choices have exactly the same labels, therefore the costs are different only for the energy terms that include pixels above the red line. (b) is the same as in (a) but now column j is in state n. Still the costs of four possible states differs only in the energy terms above the red line, since labels below the red line are the same.

We are going to minimize these two cases separately, and each one in $O(1)$ time. Consider first computation of $T_j(t)$, i.e. minimization is over $t' \leq t$. In this case, we are assigning α to pixels $(0, j), (1, j), ...(t-1, j)$ and the rest of pixels in column j retain their old label. We have to find the best possible state $t' \leq t$ in the previous column $j-1$. That is the previous column we can have α in rows 0 through $t'-1$, where t' can range from 0 to t. This is illustrated in Figure 4(a).

When considering which $t' \in 0, ..., t$ is the optimal state to go through in column $j-1$, observe that no matter which particular value we choose for t', both in column $j-1$ and column j, in rows including and below t (below the red line in Figure 4(a)), all the pixels have the same labels (the old labels they had in l'). Therefore all pairwise terms V and unary terms D of the original 2D labeling that depend only on pixels in rows below and including t are the same for all choices of $t' \in 0, ..., t$. This means that we can find the optimal t' only considering the unary and pairwise terms that involve pixels above row t.

Furthermore, consider Figure 4(b). It illustrates the case when we are computing $T_j(n)$, that is the jth column has all pixels labeled as α. Notice that out of $t' \leq t$, the best t' is the same for both (a) and (b). This is because again, the best t' is independent of the labels of pixels in rows below the red line. And above the red line, all cases for (a) and (b) are identical. Thus t' minimizing (a) and (b) is the same, but the values at the minimum t' are different. The

difference between minimum values depends only on t, let us denote it by $k(t)$. To compute $k(t)$, we just have to add up all the energy terms in (a) that depend on pixels below the red line and subtract all the energy terms in (b) that depend on pixels below the red line. Therefore we get:

$$T_j(t) = \min_{t' \leq t} \left[E_{j-1}(t') + B_{j-1}(t', t) \right] = k(t) + \min_{t' \leq t} \left[E_{j-1}(t') + B_{j-1}(t', n) \right], \quad (11)$$

where

$$k(t) = \sum_{i=t}^{n-1} \left[V^h_{i,j-1}(l'_{i,j-1}, l'_{i,j}) - V^h_{i,j-1}(l'_{i,j-1}, \alpha) \right].$$

We can compute $k(t)$ in $O(1)$ time using cumulative sums on the columns that store pairwise costs V^h, in the same way as B_j terms in Equation (4).

Let us now turn to the efficient computation of the second term in Equation (11). First define a table C of size n, indexed starting at 0, as:

$$C(i) = E_{j-1}(i) + B_{j-1}(i, n)$$

We compute running minimum of C as another array C', also of size n:

$$C'(i) = \min_{i' \leq i} C(i). \quad (12)$$

We can compute C' in $O(n)$ time. Initialize $C'(0) = C(0)$. Then update, in order of increasing i, $C'(i) = \min\{C(i), C'(i-1)\}$. After the table C' is computed, the minimum in Equation (11) can be simply looked up as $C'(t)$. Since we reuse C' for all t in column j, computing $T_j(t)$ in Equation (11) takes $O(n)$ time for all $t \in \{0, 1, ..., n-1\}$.

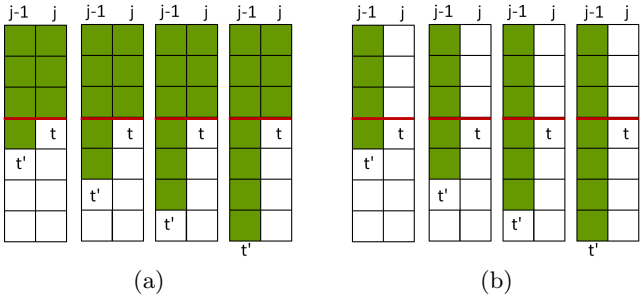
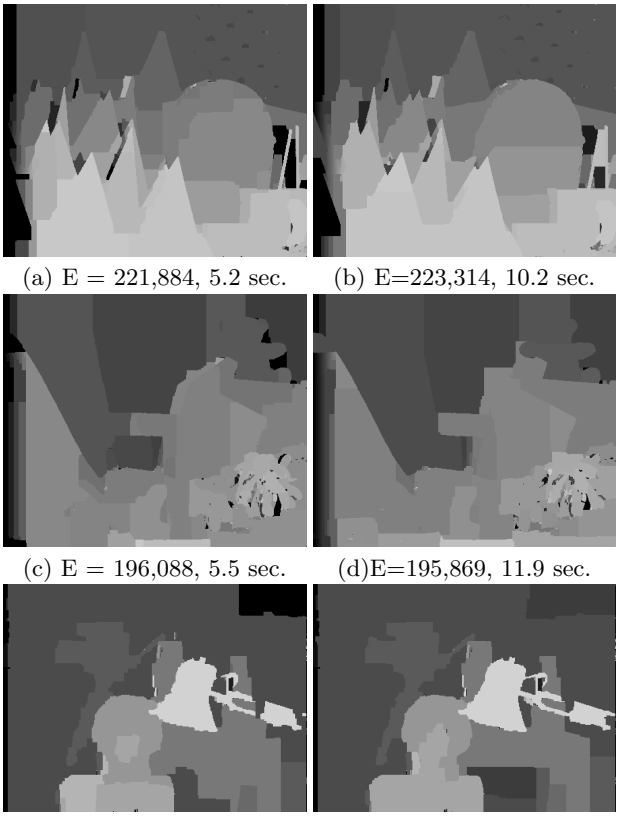


Fig. 5. Case 2: Minimization over $t' > t$. (a) When column j is fixed to state t, all possible choices for t' in column $j-1$ are illustrated. Above the red line, all four choices have exactly the same labels, therefore the costs are different only for the energy terms that include pixels below the red line. (b) is the same as in (a) but now column j is in state 0. Still the costs of four possible states differs only in the energy terms below the red line, since labels above the red line are the same.

The computation of $L_j(t)$, i.e. minimization is over $t' > t$ is handled very similarly. The illustration is in Figure 5. Observe that in rows smaller than t, all pixel labels are the same (and equal to α) for all different choices of t', see Figure 5(a). This means that we can find the optimal t' without considering any energy terms that depend entirely on pixels in rows smaller than or equal to t.



(a) E = 221,884, 5.2 sec. (b) E=223,314, 10.2 sec.

(c) E = 196,088, 5.5 sec. (d)E=195,869, 11.9 sec.

(e) E=350,958, 1.1 sec. (f) E = 346,085, 2.2 sec.

Fig. 6. Results on Middlebury stereo pairs. The left column shows our results, the right one max-flow expansion results. We report the energy and the running time in seconds for each case.

Furthermore, consider Figure 5(b). It illustrates the case when we are computing $T_j(0)$, that is the jth column retains old labels for all pixels. Notice that the minimum state t' is the same between (a) and (b), when only $t' > t$ are considered. The difference in minimum values between (a) and (b) differ by $k'(t)$ that depends only on t, not t'. Therefore we get:

$$L_i(t) = \min_{t'>t} \left[E_{i-1}(t') + B_{i-1}(t', t)\right] = k'(t) + \min_{t' \leq t} \left[E_{i-1}(t') + B_{i-1}(t', 0)\right], \quad (13)$$

where

$$k'(t) = \sum_{i=0}^{t-1} \left[V^h_{i,j-1}(\alpha, \alpha) - V^h_{i,j-1}(\alpha, l'_{i,j}) \right].$$

We compute $k'(t)$ in $O(1)$ time using cumulative sums, similar to that of $k(t)$.

Let us define a table H of size n as:

$$H(i) = E_{j-1}(i) + B_{j-1}(i, 0).$$

We need to compute running minimum for array H, but now in the other direction than what we used for array C. Define running minimum H' as:

$$H'(i) = \min_{i' \geq i} H(i'). \tag{14}$$

We can compute H' in $O(n)$ time. Initialize $H'(n-1) = H(n-1)$. Then compute, in order of decreasing i, $H'(i) = \min\{H(i), H'(i+1)\}$. After the table H' is computed, the minimum in Equation (13) can be simply looked up as $H'(t)$. Since we reuse H' for all t in column j, computing $L_j(t)$ in Equation (11) takes $O(n)$ time for all $t \in \{0, 1, ..., n-1\}$.

The parent table P that is used to recover the optimal state in DP is computed very similar to the tables E, therefore we omit the description.

5 Experimental Results

In this section, we evaluate the performance of our DP-expansion algorithm and compare it to the max-flow expansion. We do not compare our method to previous work that improves efficiency of expansion, such as [28–30], since our goal is to introduce a previously unexplored direction for speed-ups, rather than compete with previous work. In addition, since our approach is orthogonal to [28–30], we can reuse some of their ideas.

Even though we can handle arbitrary pairwise potentials, we evaluate here only the Potts model, namely $V_{pq}(l_p, l_q) = w_{pq} \cdot \min\{|l_p - l_q|, 1\}$. As in previous work [1], w_{pq} depends on the strength of the gradient between pixels p and q in the observed image. The stronger the gradient, the smaller is w_{pq}.

We evaluate our performance on the application of stereo correspondence and on the set of ten Middlebury stereo pairs [33, 34]. For the data term, we use a simple truncated absolute value difference between the pixel in the left image and the pixel in the right image shifted according to the disparity label. That is if L is the left image and R is the right image, $D_p(l_p) = \min\{T, |L(p) - R(p - l_p)|\}$. We used $T = 20$. Since our goal is to evaluate our DP-expansion moves as an optimization method, we did not tune parameters for the best stereo correspondence performance. We ran the expansion algorithm for two iterations, since it has been observed that the energy decreases most during the first two iterations. We ran our algorithm for box sizes from 1/10 of the image to the largest image dimension, increasing them by a factor of 2 and making them overlap by half of their side length.

It has been shown [1] that first two iterations decrease the energy the most for the max-flow expansion. Thus to make comparisons more fair to max-flow

expansion, we run it for two iterations. On average, max-flow expansion finds an energy 1.3% lower, where we measured energy difference between DP-expansion and max-flow expansion relative to the energy computed by max-flow expansion. The largest relative energy difference was 3%, and in some case DP-expansion returned a slightly lower energy. This is only because max-flow expansion was not run until convergence. The running time of our approach was, on average, 1.8 times faster. Fig. 6 illustrates some of the results.

6 Discussion

We have presented DP-expansion moves that work by finding a restricted structure subset of pixels to switch to α. The restriction allows to implement them very efficiently, in linear time. There are many other aspects for future research. First, there are many variants of such moves to explore. For example, it is easy to extend our moves to the case when the "anchoring" happens not necessarily at the same row for each column. Right now, the anchoring is done in row 0 (or at the left, right, bottom of the picture). It is easy to see that as long as the one of the end points is fixed, not necessarily to the same value for each column, we can compute the move with with linear time efficiency. These moves can be used for taking a low-quality but cheap solution, exploring the connected sets of pixels assigned to the same label, and improving their boundaries one side at a time, leaving the other sides fixed. It is also easy to parallelize our algorithm, expanding on a non-overlapping set of boxes at the same time. Most of the boxes are small to medium in size and it is easy to come up with a non-overlapping subsets of them. Finally, just as there are several speed-ups to the basic max-flow expansion suggested in the literature [29, 30], based on clever heuristics, there are multiple speed-ups possible for our approach too.

References

1. Boykov, Y., Veksler, O., Zabih, R.: Efficient approximate energy minimization via graph cuts. In: PAMI. Volume 26. (2001) 1222–1239
2. Wills, J., Agarwal, S., Belongie, S.: What went where. In: CVPR. (2003) I: 37–44
3. Agarwala, A., Dontcheva, M., Agrawala, M., Drucker, S., Colburn, A., Curless, B., Salesin, D., Cohen, M.: Interactive digital photomontage. In: ACM Transactions on Graphics. Volume 23. (2004) 294 – 302
4. Kwatra, V., Schdl, A., Essa, I., Turk, G., Bobick, A.: Graphcut textures: Image and video synthesis using graph cuts. SIGGRAPH 2003 **22** (2003) 277–286
5. Hoiem, D., Rother, C., Winn, J.: 3d layout crf for multi-view object class recognition and segmentation. In: CVPR. (2007) 1–8
6. Delong, A., Osokin, A., Isack, H.N., Boykov, Y.: Fast approximate energy minimization with label costs. IJCV **96** (2012) 1–27
7. Boykov, Y., Veksler, O., Zabih, R.: Fast approximate energy minimization via graph cuts. PAMI **23** (2001) 1222–1239
8. Greig, D., Porteous, B., Seheult, A.: Exact maximum a posteriori estimation for binary images. Journal of the Royal Statistical Society, Series B **51** (1989) 271–279
9. Ishikawa, H.: Exact optimization for markov random fields with convex priors. PAMI **25** (2003) 1333–1336

10. Schlesinger, D., Flach, B.: Transforming an arbitrary minsum problem into a binary one. Technical Report TUD-FI06-01, Dresden University of Technology (2006)
11. Kolmogorov, V., Rother, C.: Minimizing nonsubmodular functions with graph cuts - a review. PAMI **29** (2007) 1274–1279
12. Felzenszwalb, P.F., Veksler, O.: Tiered scene labeling with dynamic programming. In: CVPR. (2010)
13. Ford, L., Fulkerson, D.: Flows in Networks. Princeton University Press (1962)
14. Boykov, Y., Kolmogorov, V.: An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. PAMI **26** (2004) 1124–1137
15. Amini, A., Weymouth, T., Jain, R.: Using dynamic programming for solving variational problems in vision. PAMI **12** (1990) 855–867
16. Szeliski, R., Zabih, R., Scharstein, D., Veksler, O., Kolmogorov, V., Agarwala, A., Tappen, M.F., Rother, C.: A comparative study of energy minimization methods for markov random fields with smoothness-based priors. PAMI **30** (2008) 1068–1080
17. Pearl, J.: Probabilistic reasoning in intelligent systems: networks of plausible inference. Morgan Kaufmann (1988)
18. Kolmogorov, V.: Convergent tree-reweighted message passing for energy minimization. PAMI **28** (2006) 1568–1583
19. Lempitsky, V.S., Rother, C., Roth, S., Blake, A.: Fusion moves for markov random field optimization. PAMI **32** (2010) 1392–1405
20. Veksler, O.: Graph cut based optimization for mrfs with truncated convex priors. In: CVPR. (2007) 1–8
21. Kumar, M.P., Torr, P.H.S.: Improved moves for truncated convex models. In Koller, D., Schuurmans, D., Bengio, Y., Bottou, L., eds.: NIPS. (2009) 889–896
22. Gould, S., Amat, F., Koller, D.: Alphabet soup: A framework for approximate energy minimization. In: CVPR. (2009) 903–910
23. Carr, P., Hartley, R.: Solving multilabel graph cut problems with multilabel swap. In: DICTA. (2009)
24. Vineet, V., Warrell, J., Torr, P.: A tiered move-making algorithm for general pairwise mrfs. In: CVPR. (2012)
25. Liu, X., Veksler, O., Samarabandu, J.: Order preserving moves for graph cut based optimization. PAMI (2010)
26. Bai, J., Song, Q., Veksler, O., Wu, X.: Fast dynamic programming for labeling problems with ordering constraints. In: CVPR. (2012)
27. Asano, T., Chen, D., Katoh, N., Tokuyama, T.: Efficient algorithms for optimization-based image segmentation. IJCGA **11** (2001) 145–166
28. Komodakis, N., Tziritas, G., Paragios, N.: Fast, approximately optimal solutions for single and dynamic mrfs. In: CVPR. (2007)
29. Alahari, K., Kohli, P., Torr, P.H.S.: Reduce, reuse & recycle: Efficiently solving multi-label mrfs. In: CVPR. (2008)
30. Batra, D., Kohli, P.: Making the right moves: Guiding alpha-expansion using local primal-dual gaps. In: CVPR. (2011) 1865–1872
31. Kolmogorov, V., Zabih, R.: What energy function can be minimized via graph cuts? PAMI **26** (2004) 147–159
32. Viola, P.A., Jones, M.J.: Rapid object detection using a boosted cascade of simple features. In: CVPR (1). (2001) 511–518
33. Scharstein, D., Szeliski, R.: A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. IJCV **47** (2001) 7–42
34. Scharstein, D., Szeliski, R.: High-accuracy stereo depth maps using structured light. In: CVPR. (2003) 195–202