APPEARED IN THE PROCEEDINGS OF IEEE ICCV 2005

A New Framework for Approximate Labeling via Graph Cuts

Nikos Komodakis, Georgios Tziritas
Computer Science Department, University of Crete
E-mails:{komod, tziritas}@csd.uoc.gr

Abstract

A new framework is presented that uses tools from duality theory of linear programming to derive graph-cut based combinatorial algorithms for approximating NP-hard classification problems. The derived algorithms include α-expansion graph cut techniques merely as a special case, have guaranteed optimality properties even in cases where α-expansion techniques fail to do so and can provide very tight per-instance suboptimality bounds in all occasions.

1 Introduction

The Metric Labeling Problem (ML) [3] can capture a broad range of classification problems that arise in early vision (e.g. image restoration, stereo matching etc.). Given a set V of objects and a set L of labels the goal is to find a labeling $f : V \rightarrow L$ with a minimum cost $C(f)$. The cost $C(f) = \sum_{p \in V} c_{p,f(p)} + \sum_{(p,q) \in E} w_{pq} d_{f(p)f(q)}$ consists of 2 parts. On one hand, for each $p \in V$ there is a label cost $c_{p,f(p)} \geq 0$ for assigning label $f(p)$ to p. On the other hand, for each pair of objects p, q that are adjacent in a graph $G = (V, E)$ there is a so-called *separation cost* for assigning labels $a = f(p), b = f(q)$ to them. This separation cost equals to $w_{pq} d_{ab}$ where w_{pq} is the weight of edge $(p, q) \in E$ and represents the strength of similarity between p, q while d_{ab} is a metric[1] distance between labels.

One class of approximation algorithms [3, 2] solves the ML problem by formulating it as the Linear Programming relaxation of an integer program. However these methods require the solution of a very large linear program and are therefore impractical to use. On the other hand, a variety of combinatorial algorithms based on graph cuts [1, 8, 4, 9] have been developed which are fast but that have been interpreted only as greedy local search techniques up to now.

The framework presented in this paper bridges the gap between these two classes of algorithms and makes use of the *primal-dual schema* of linear programming to derive combinatorial approximation algorithms for the ML problem. Its major contributions are: **1)** The derived algorithms have (for the first time) guaranteed optimality properties even in the more general case where d_{ab} is merely a semi-metric[2], a case which is often encountered in problems of early vision. **2)** Besides these theoretical optimality properties, the considered algorithms also provide per-instance suboptimality bounds that prove to be much tighter in practice, thus showing that a nearly optimal solution is always obtained. **3)** α-expansion graph cut techniques [1] are included merely as a special case of our framework. This fact along with the previous remark explains the great success these techniques exhibit in practice. Furthermore, for the first time these (state-of-the-art) expansion techniques are interpreted not merely as greedy local search but in terms of principles drawn from the theory of linear programming.

We describe our algorithms in sections 2-5 while we show experimental results in section 6.

2 The primal-dual schema

Consider the primal-dual pair of linear programs:

$$\text{PRIMAL: } \min c^T x \text{ s.t. } Ax = b, x \geq 0 \qquad \text{DUAL: } \max b^T y \text{ s.t. } A^T y \leq c$$

where $A = [a_{ij}]$ is an $m \times n$ matrix and b, c are vectors of size m, n respectively. We want to find an optimal integral solution to the primal program but since this is in general an NP-complete problem we must settle with estimating approximate solutions. A primal-dual f-approximation algorithm achieves that by use of the following principle [7]:

Theorem (Relaxed Complementary Slackness). *If the pair (x, y) of integral-primal and dual feasible solutions satisfies the so-called relaxed primal complementary slackness conditions: $\forall\, x_j > 0 \Rightarrow \sum_{i=1}^{m} a_{ij} y_i \geq c_j / f_j$*
then x is an f-approximation to the optimal integral solution x^* (i.e. $c^T x \leq f \cdot c^T x^*$) with $f = \max_j f_j$.

Based on this theorem, a primal-dual f-approximation algorithm usually exploits the following iterative schema:

Primal-Dual Schema. *Keep generating pairs of integral-primal, dual solutions $\{(x^k, y^k)\}_{k=1}^{t}$ until the elements x^t, y^t of the last pair are both feasible and satisfy the relaxed primal complementary slackness conditions.*

2.1 Metric Labeling as a linear program

Here we will use the following integer programming formulation of the ML problem, introduced in [2]:

$$\min \sum_{p \in V, a \in L} c_{p,a} x_{p,a} + \sum_{(p,q) \in E} w_{pq} \sum_{a,b \in L} d_{ab} x_{pq,ab} \qquad (1)$$

[1] i.e. $d_{ab} = 0 \Leftrightarrow a = b,\ d_{ab} = d_{ba} \geq 0,\ d_{ab} \leq d_{ac} + d_{cb}$
[2] i.e. $d_{ab} = 0 \Leftrightarrow a = b,\ d_{ab} = d_{ba} \geq 0$

$$\text{s.t.} \sum_a x_{p,a} = 1 \qquad \forall\, p \in V \tag{2}$$

$$\sum_a x_{pq,ab} = x_{q,b} \qquad \forall\, b \in L,\ (p,q) \in E \tag{3}$$

$$\sum_b x_{pq,ab} = x_{p,a} \qquad \forall\, a \in L,\ (p,q) \in E \tag{4}$$

$$x_{p,a},\ x_{pq,ab} \in \{0,1\} \qquad \forall\, p \in V,\ (p,q) \in E,\ \ a,b \in L$$

The $\{0,1\}$-variable $x_{p,a}$ indicates that vertex p is assigned label a while the $\{0,1\}$-variable $x_{pq,ab}$ indicates that vertex p is labeled a and vertex q is labeled b. The variables $x_{pq,ab}$, $x_{qp,ba}$ therefore indicate the same thing. So, in order to eliminate one of them and reduce the number of variables in the primal problem, we assume w.l.o.g. that only one of (p,q), (q,p) belongs to E for any neighbors p,q. The notation "$p \sim q$" will hereafter denote that p,q are neighbors i.e. "either $(p,q) \in E$ or $(q,p) \in E$". The first constraints (2) simply express the fact that each vertex must receive a label while constraints (3), (4) maintain consistency between variables $x_{p,a}, x_{q,b}$ and $x_{pq,ab}$ in the sense that if $x_{p,a} = 1, x_{q,b} = 1$ they force $x_{pq,ab} = 1$ as well.

By relaxing the $\{0,1\}$ constraints to $x_{p,a} \geq 0,\ x_{pq,ab} \geq 0$ we get a linear program. Its dual has the following form:

$$\max \sum_p y_p$$

$$\text{s.t. } y_p \leq ht^y_{p,a} \qquad \forall p \in V,\ a \in L \tag{5}$$

$$y_{pq,a} + y_{qp,b} \leq w_{pq} d_{ab} \quad \forall a,b \in L, (p,q) \in E \tag{6}$$

$$\text{where: } ht^y_{p,a} \equiv c_{p,a} + \sum_{q:q\sim p} y_{pq,a} \tag{7}$$

To each vertex p, there corresponds one dual variable y_p. Also, to each edge $(p,q) \in E$ (and any label a), there correspond 2 dual variables $y_{pq,a}$ and $y_{qp,a}$. All the dual variables $y_{pq,a}, y_{qp,a}$ will be called "balance variables" hereafter and we will also say that $y_{pq,a}$ is the conjugate balance variable of $y_{qp,a}$ (and vice versa). The auxiliary variables $ht^y_{p,a}$ will be called "height variables". The reason for this as well as for introducing these redundant variables will become clear in the sections that are following. For defining a dual solution, only the balance variables $y_{pq,a}$ as well as the y_p variables must be specified. The height variables $ht^y_{p,a}$ are then computed by (7).

Since we will be considering only feasible $\{0,1\}$-primal solutions, a primal solution x will hereafter refer to a set of labels $\{x_p\}_{p\in V}$ where x_p denotes the label assigned to vertex p. Under this notation, $x_{p,a} = 1$ is equivalent to $x_p = a$ and so it is not difficult to see that the relaxed slackness conditions related to the $x_{p,a}$ variables are reduced to:

$$y_p \geq c_{p,x_p}/f_1 + \sum_{q:q\sim p} y_{pq,x_p} \tag{8}$$

while the slackness conditions of the $x_{pq,ab}$ variables are:

$$x_p \neq x_q \Rightarrow y_{pq,x_p} + y_{qp,x_q} \geq w_{pq} d_{x_p x_q}/f_2 \tag{9}$$

$$x_p = x_q = a \Rightarrow y_{pq,a} + y_{qp,a} = 0 \tag{10}$$

where we consider the cases $a \neq b$ and $a = b$ separately.

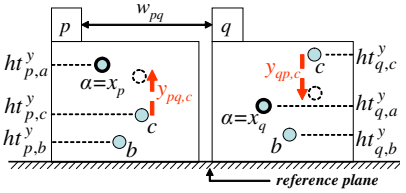


Fig. 1: Visualization of the dual for a graph G consisting of just 2 neighbors p,q while $L = \{a,b,c\}$. Each vertex holds a copy of all labels in L and all these labels are represented by circles which are located at certain heights specified by the ht variables. Label c at p is pulled up due to the increase of the balance variable $y_{pq,c}$ and so the corresponding label at neighboring vertex q is pulled down due to the decrease of the conjugate variable $y_{qp,c}$. The active labels of p,q are drawn with a thicker line.

Our objective will therefore be to find feasible solutions x, y satisfying the above conditions (8), (9) and (10). Conditions (10) simply say that conjugate balance variables are opposite to each other. For this reason we set by definition:

$$y_{qp,a} \equiv -y_{pq,a} \quad \forall\, (p,q) \in E,\ a \in L \tag{11}$$

and so no worry about conditions (10) is needed hereafter.

2.2 An intuitive view of the dual variables

A way of viewing/visualizing the dual variables, that will prove useful when later designing our approximation algorithms, is the following: for each vertex p, we consider a separate copy of the complete set of labels L. It is then assumed that all of these labels are objects which are located at certain heights relative to a common reference plane (see Fig. 1). The height of label a at vertex p is given by the dual variable $ht^y_{p,a}$. Expressions like "label a at p is below/above label b" imply $ht^y_{p,a} \lesseqgtr ht^y_{p,b}$. The role of the balance variables is to contribute to the increase or decrease of a vertex's height. In particular, due to (7), the height of a label a at p can be altered only if at least one of the balance variables $\{y_{pq,a}\}_{q:q\sim p}$ is altered as well. In addition, due to the fact that conjugate balance variables are opposite to each other (see (11)), changes in the height of label a at p also affect the height of that label at a neighboring vertex. In Fig. 1, for example, each time we increase the height of label c at p, say by increasing balance variable $y_{pq,c}$, the height of c at the neighboring vertex q is decreased by the same amount due to the decrease of the conjugate variable $y_{qp,c}$.

Before proceeding let us also define some terminology. Let x, y be any pair of integral-primal, dual solutions. We will call the label assigned to p by x (i.e. x_p) the *active label at* p. We will also refer to the height of the active label at p (i.e. ht^y_{p,x_p}) as merely *the height of* p. We will call the sum of the heights of all vertices the *"Approximate Primal Function"* (or APF for short) i.e. $APF^{x,y} = \sum_p ht^y_{p,x_p}$. This function's name comes from the fact that if x, y satisfy the relaxed slackness conditions then it is easy to prove that APF approximates the primal objective function.

Also, any balance variable in the set $\{y_{pq,x_p}\}_{q:q\sim p}$ (i.e. any balance variable of the form y_{pq,x_p} with q adjacent to

1: $k \leftarrow 1; x^k \leftarrow$INIT_PRIMALS( ); $y^k \leftarrow$INIT_DUALS( );
2: $LabelChange \leftarrow 0$
3: **for** each label c in L **do**
4: $\bar{y}^k \leftarrow$ PREEDIT_DUALS(c, x^k, y^k);
5: $[x^{k+1}, \bar{y}^{k+1}] \leftarrow$UPDATE_DUALS_PRIMALS$(c, x^k, \bar{y}^k)$;
6: $y^{k+1} \leftarrow$POSTEDIT_DUALS$(c, x^{k+1}, \bar{y}^{k+1})$;
7: **if** $x^{k+1} \neq x^k$ **then** $LabelChange \leftarrow 1$
8: k++;
9: **end for**
10: **if** $LabelChange = 1$ **then** goto 2;
11: **if** algorithm $\neq$ PD1 **then** $y^{fit} \leftarrow$DUAL_FIT(y^k);

Fig. 2: The basic structure of the algorithms PD1, PD2 and PD3.

p) will be called an *active balance variable at vertex* p. Another important quantity is the *load* between two neighbors p, q ($load_{pq}^{x,y}$) which is defined as the sum of the 2 active balance variables y_{pq,x_p}, y_{qp,x_q} i.e. $load_{pq}^{x,y} = y_{pq,x_p} + y_{qp,x_q}$. If relaxed slackness conditions (9) hold, then due to (9) and (6) it is easy to see that the load between p, q can be thought of as a virtual separation cost which approximates the actual separation cost $w_{pq}d_{x_px_q}$ between p, q.

2.3 Applying the primal-dual schema to ML

Most of our approximation algorithms can achieve an approximation factor of $f_{app} = 2\frac{d_{max}}{d_{min}}$ with $d_{min} \equiv min_{a \neq b} d_{ab}$ and $d_{max} \equiv max_{a \neq b} d_{ab}$. Their basic structure can be seen in Fig. 2. The initial primal-dual solutions are generated inside INIT_PRIMALS and INIT_DUALS. During an inner iteration (lines 4-8 in Fig. 2) a label c is selected and a new primal-dual pair of solutions (x^{k+1}, y^{k+1}) is generated by updating the current pair (x^k, y^k). During this iteration, among all balance variables of y^k (i.e. $\{y_{pq,a}^k\}_{p,q:p\sim q}^{a \in L}$) only the balance variables of the c labels (i.e. $\{y_{pq,c}^k\}_{p,q:p\sim q}$) are modified. We call this a c-iteration of the algorithm. $|L|$ such iterations (one c-iteration for each label c in the set L) make up an outer iteration (lines 2-9 in Fig. 2) and the algorithm terminates if no vertex changes its label during the current outer iteration.

During an inner iteration the main update of the primal and dual variables takes place inside UPDATE_DUALS_PRIMALS while PREEDIT_DUALS and POSTEDIT_DUALS modify the dual variables before and after the main update. The algorithms to be considered are named PD1, PD2, PD3 and the DUAL_FIT routine, which is used only in the last two of them, serves only the purpose of applying a scaling operation to the last dual solution.

3 The PD1 algorithm

During this section we will assume that d_{ab} is a semimetric. In the case of the PD1 algorithm our goal will be to find feasible x, y satisfying slackness conditions (8), (9) with $f_1 = 1$ and $f_2 = f_{app}$. By replacing $f_1 = 1$ in (8) that condition becomes $y_p \geq ht_{p,x_p}^y$. Since it also holds that $y_p \leq \min_a ht_{p,a}^y$ (by the dual constraints (5)), it is easy to see that (8) reduces to the following 2 equations:

$$y_p = \min_a ht_{p,a}^y \tag{12}$$

$$ht_{p,x_p}^y = \min_a ht_{p,a}^y \tag{13}$$

In addition, $load_{pq}^{x,y} = y_{pq,x_p} + y_{qp,x_q}$ (by definition) and so by replacing $f_2 = f_{app}$ in (9) that condition reduces to:

$$x_p \neq x_q \Rightarrow load_{pq}^{x,y} \geq w_{pq}d_{x_px_q}/f_{app} \tag{14}$$

Therefore the objective of PD1 is to find feasible x, y satisfying conditions (12)-(14) and for this, PD1 uses the following strategy: At each iteration it makes sure that conditions (12) and (14) are automatically satisfied by the current primal-dual pair. In addition, it makes sure that the current dual solution is feasible (primal solutions are always integral-feasible by construction). To this end it always imposes the following constraints to the current dual solution:

$$y_{pq,a} \leq w_{pq}d_{min}/2 \quad \forall\, a \in L,\ p \sim q \tag{15}$$

To see that (15) ensures feasibility, it is enough to observe that due to this constraint the following inequality can be derived: $y_{pq,a} + y_{qp,b} \leq 2w_{pq}d_{min}/2 = w_{pq}d_{min} \leq w_{pq}d_{ab}$ and so the dual constraints (6) hold true. This implies that solution y is indeed feasible since the other dual constraints (5) already hold true due to condition (12).

All that remains then for PD1 to achieve its goal is just to ensure that after a finite number of iterations slackness conditions (13) are satisfied as well. These conditions simply require that the active label of any vertex must be "lower" than all other labels at that vertex. So assuming that at the start of an outer iteration all conditions (12)-(15) except for (13) are satisfied, the update of the primal and dual variables roughly proceeds as follows:

DUAL VARIABLES UPDATE: Given the current primal solution (i.e. the current label assignment), we try to update the balance variables so that for each vertex its active label stays at the same height while the rest of the labels at that vertex are raised above the vertex's active label. Since we must also ensure that the balance variables do not increase too much (or else constraints (15) would become violated), this cannot be achieved for all vertices.

PRIMAL VARIABLES UPDATE: This implies that after the rearrangement of the labels' heights, there might still be some vertices violating condition (13). We select a suitable subset of these vertices and assign to them new active labels with lower heights so that the resulting primal solution is taken closer to satisfying (13). The reason we may not be able to do that for all the vertices is that we must still take care that conditions (14) are maintained as well.

However, by keep repeating this procedure, the number of vertices violating (13) decreases per iteration and so in the end all conditions (12)-(15) will hold true. It should be noted that it is always trivial to enforce conditions (12) (we simply set each dual variable y_p equal to $\min_a ht_{p,a}^y$).

The rearrangement of the labels' heights takes place in groups, one group per inner iteration. In particular, during a c-iteration only the heights of the c labels are rearranged so that as many of these labels as possible are raised

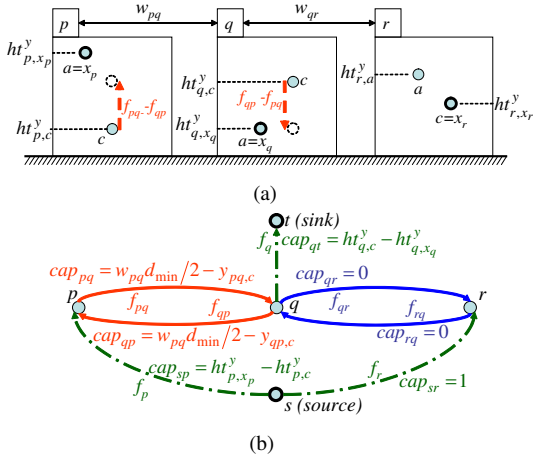


Fig. 3: **(a)** A arrangement of labels (represented by circles) for a graph G with 3 vertices p, q, r and 2 edges pq, qr of weights w_{pq}, w_{qr}. The label set is $L = \{a, c\}$. The circles with the thicker line represent the active labels. Also, the red arrows indicate how the c labels will move in respond to the update of the dual variables while the circles with the dashed line show the final position of those labels after the update. **(b)** The corresponding graph $G_c^{x,y}$ that will be used for the update of the dual variables. Interior/exterior edges are drawn with solid/dashed lines respectively.

above the corresponding active labels. To this end solution y is changed into solution y' by changing only variables $\{y_{pq,c}\}_{p,q:p\sim q}$ (i.e. the balance variables of all c labels) into $\{y'_{pq,c}\}_{p,q:p\sim q}$. The new heights will therefore be $ht^{y'}_{p,c}$. We must be careful, though, during this update of the c-heights. For example, in Fig. 3(a), if we try to raise c at vertex p (by increasing $y_{pq,c}$) so as to reach x_p then label c at q may go below x_q due to the decrease of the conjugate variable $y_{qp,c}$, thus breaking condition (13) for q. It turns out that the optimal update can be simulated by pushing the maximum amount of flow through a directed graph $G_c^{x,y} = (V_c^{x,y}, E_c^{x,y}, C_c^{x,y})$ with capacities $C_c^{x,y}$ where x, y are the primal-dual solutions at the start of the c-iteration (see Fig. 3(b) for an example of such a graph).

The nodes $V_c^{x,y}$ of $G_c^{x,y}$ consist of all the nodes of graph G (these are the internal nodes) plus two special external nodes the source s and the sink t. The nodes of $G_c^{x,y}$ are connected by two types of edges: interior and exterior edges (drawn with solid/dashed lines respectively in Fig. 3(b)).

Interior edges: For each edge $(p, q) \in G$, we insert 2 directed interior edges pq and qp in graph $G_c^{x,y}$. The amount of flow f_{pq} coming out of p through pq represents the increase of the balance variable $y_{pq,c}$ while the reverse flow f_{qp} represents the decrease of the same variable $y_{pq,c}$. The total change of $y_{pq,c}$ will therefore be:

$$y'_{pq,c} = y_{pq,c} + f_{pq} - f_{qp} \tag{16}$$

The total change in $y_{qp,c}$ is defined symmetrically (since any flow coming out of p through pq will enter q and vice versa) and so $y'_{pq,c} = -y'_{qp,c}$ i.e. conjugate balance variables remain opposite to each other as they should.

Based on (16), it is obvious that the capacity cap_{pq} of edge pq represents the maximum allowed increase of $y_{pq,c}$ (attained if $f_{pq} = cap_{pq}, f_{qp} = 0$) while a similar conclusion holds for cap_{qp}. Based on this observation cap_{pq}, cap_{qp} are assigned as follows: if the active label of p (or q) is equal to c then the height of c at p (or q) must stay fixed at this iteration and so we want no increase in $y_{pq,c}, y_{qp,c}$:

$$x_p = c \text{ or } x_q = c \Rightarrow cap_{pq} = cap_{qp} = 0 \tag{17}$$

Otherwise (i.e. $x_p \neq c, x_q \neq c$), we set cap_{pq}, cap_{qp} so that the values of the new balance variables $y'_{pq,c}$, $y'_{qp,c}$ can never exceed $w_{pq}d_{min}/2$ and feasibility conditions (15) are therefore maintained for the new dual solution y' i.e.:

$$y_{pq,c} + cap_{pq} = w_{pq}d_{min}/2 = y_{qp,c} + cap_{qp} \tag{18}$$

Capacities of blue/red edges in Fig.3(b) are set by (17)/ (18).

Exterior edges: Each internal node p connects to either the source node s or the sink node t (but not to both of them) through an exterior edge. There are 3 possible cases:

Case 1: If c is "below" x_p (i.e. $ht^y_{p,c} < ht^y_{p,x_p}$) then we would like to raise label c by exactly as much as needed so that it reaches label x_p. To this end, we connect the source node s to node p through a directed edge sp. The flow f_p passing through that edge represents the total increase in the height of label c i.e.:

$$ht^{y'}_{p,c} = ht^y_{p,c} + f_p \tag{19}$$

To verify this, it suffices to combine the flow conservation at node p which can be easily seen to reduce to $f_p = \sum_{q:q\sim p}(f_{pq} - f_{qp})$ and the fact that $f_{pq} - f_{qp}$ represents the total change of the balance variable $y_{pq,c}$ i.e. $f_{pq} - f_{qp} = y'_{pq,c} - y_{pq,c}$ (see (16)). Based on (19), the capacity cap_{sp} of the edge sp represents the maximum allowed raise in the height of c. Since we need to raise c only as high as the current active label of p but not higher than that, we therefore set: $cap_{sp} = ht^y_{p,x_p} - ht^y_{p,c}$ (see edge sp in Fig. 3(b)).

Case 2: If c is not "below" x_p (i.e. $ht^y_{p,c} \geq ht^y_{p,x_p}$) and is also not the active label of p (i.e. $c \neq x_p$) then we can afford a decrease in the height of c as long as c remains "above" x_p. To this end, we connect p to sink node t through directed edge pt. This time the flow f_p through edge pt equals the total decrease in the height of c i.e. $ht^{y'}_{p,c} = ht^y_{p,c} - f_p$ and so cap_{pt} represents the maximum decrease. We therefore set: $cap_{pt} = ht^y_{p,c} - ht^y_{p,x_p}$ (see edge qt in Fig. 3(b)).

Case 3: Finally, if c is the active label of p (i.e. $c = x_p$) then we want to keep the height of c fixed at the current iteration. As in case 1, we again connect the source node s to p through directed edge sp. This time, however, no flow passes through any interior edge incident to p (due to (17)). So $f_p = 0$ (due to the flow conservation at p) and the height of label c will not change (see (19)), as was intended. By convention we set: $cap_{sp} = 1$ (see edge sr in Fig. 3(b)).

3.1 Update of the primal and dual variables

We are now ready to describe what actions are performed by each of the main routines of PD1 during a c-iteration.

PREEDIT_DUALS(c, x^k, y^k): For all of the considered algorithms this routine's role is to edit current solution y^k into solution $\bar{y}^k$ that will be used (along with x^k) as input for the construction of the graph $G_c^{x^k,\bar{y}^k}$ of the previous section. No editing is needed in the case of PD1 and so $\bar{y}^k = y^k$.

UPDATE_DUALS_PRIMALS$(c, x^k, \bar{y}^k)$: The primal-dual pair $x^{k+1}, \bar{y}^{k+1}$ is generated inside this routine. For the generation of $\bar{y}^{k+1}$, the graph $G_c^{x^k,\bar{y}^k}$ is constructed and a maximum flow algorithm is applied to it. The resulting flows are used in updating only the balance variables of the c labels as explained in the previous section (see (16)) i.e.:

$$\bar{y}^{k+1}_{pq,c} = \bar{y}^k_{pq,c} + f_{pq} - f_{qp} \tag{20}$$

Therefore the heights of all c labels will also change as:

$$ht^{\bar{y}^{k+1}}_{p,c} = ht^{\bar{y}^k}_{p,c} + \begin{cases} f_p & \text{if } p \text{ is connected to node } s \\ -f_p & \text{if } p \text{ is connected to node } t \end{cases} \tag{21}$$

Based on the new heights, we now need to update x^k into x^{k+1} i.e. assign new labels to vertices. Since only the heights of c labels have changed, this amounts to deciding whether a vertex keeps its current active label or is assigned label c. On one hand, this should depend on whether the c label of a vertex managed to go "above" the active label at the same vertex or not. On the other hand, we must also ensure that conditions (14) will still hold true for $x^{k+1}, \bar{y}^{k+1}$. It turns out that both criteria can be fulfilled by use of the following rule:

REASSIGN RULE. *Label c will be the new label of p (i.e. $x_p^{k+1} = c$) $\Leftrightarrow$ $\exists$ unsaturated[3] path between the source and node p (otherwise p keeps its current label i.e. $x_p^{k+1} = x_p^k$).*

Based on the reassign rule, the following properties can be shown [5] to hold for the resulting solutions $x^{k+1}, \bar{y}^{k+1}$:

(a) If at least one vertex changes its active label during a c-iteration then $APF^{x^{k+1},\bar{y}^{k+1}} < APF^{x^k,\bar{y}^k}$

(b) $ht^{\bar{y}^{k+1}}_{p,x_p^{k+1}} \leq ht^{\bar{y}^{k+1}}_{p,c}$

(c) If $c = x_p^{k+1} \neq x_q^{k+1}$ then $\bar{y}^{k+1}_{pq,c} = \bar{y}^k_{pq,c} + cap_{pq}$

The first property can be used to ensure the algorithm's termination while the second one asserts that for any vertex its new active label is always below its c label (as intended). Furthermore, the last property can be used to prove that conditions (14) remain true [5].

POSTEDIT_DUALS$(c, x^{k+1}, \bar{y}^{k+1})$: However, for maintaining conditions (14) during the next iterations as well, it turns out that POSTEDIT_DUALS needs to change $\bar{y}^{k+1}$ into y^{k+1} so that all active balance variables of solution y^{k+1} become nonnegative while also neither the APF nor any of the *loads* are altered during this change. It can be shown that in the case of PD1, the active balance variables $\bar{y}^{k+1}_{pq,x_p^{k+1}}, \bar{y}^{k+1}_{qp,x_q^{k+1}}$ may be negative during a c-iteration only if $x_p^{k+1} = x_q^{k+1} = c$. In this case POSTEDIT_DUALS simply needs to set $y^{k+1}_{pq,c} = y^{k+1}_{qp,c} = 0$. No other differences between $\bar{y}^{k+1}, y^{k+1}$ exist.

[3] A path is unsaturated if $flow < capacity$ for all forward arcs and $flow > 0$ for all backward arcs

INIT_PRIMALS: initialize x^k by a random label assignment

INIT_DUALS
$y^k = 0$
for each pair $(p, q) \in E$ with $x_p^k \neq x_q^k$ **do**
$y^k_{pq,x_p^k} = -y^k_{qp,x_p^k} = w_{pq} d_{min}/2 = y^k_{qp,x_q^k} = -y^k_{pq,x_q^k}$
$y_p^k = \min_a ht^{y^k}_{p,a} \ \forall p \in V$ {***imposes conditions (12)***}

PREEDIT_DUALS(c, x^k, y^k): $\bar{y}^k = y^k$ {***generates*** $\bar{y}^k$}

UPDATE_DUALS_PRIMALS$(c, x^k, \bar{y}^k)$ {***generates*** $x^{k+1}, \bar{y}^{k+1}$}
$x^{k+1} = x^k$, $\bar{y}^{k+1} = \bar{y}^k$
Apply max-flow to $G_c^{x^k,\bar{y}^k}$ and compute flows f_p, f_{pq}
$\bar{y}^{k+1}_{pq,c} = \bar{y}^k_{pq,c} + f_{pq} - f_{qp} \ \ \forall p, q : p \sim q$
$\forall p \in V \ x_p^{k+1} = c \Leftrightarrow \exists$ unsaturated path $s \rightsquigarrow p$ in $G_c^{x^k,\bar{y}^k}$

POSTEDIT_DUALS$(c, x^{k+1}, \bar{y}^{k+1})$ {***generates*** y^{k+1}}
$y^{k+1} = \bar{y}^{k+1}$
for each pair $(p, q) \in E$ with $x_p^{k+1} = x_q^{k+1} = c$ **do**
if $\bar{y}^{k+1}_{pq,c} < 0$ **or** $\bar{y}^{k+1}_{qp,c} < 0$ **then** $y^{k+1}_{pq,c} = y^{k+1}_{qp,c} = 0$
$y_p^{k+1} = \min_a ht^{y^{k+1}}_{p,a} \ \ \forall p \in V$ {***imposes conditions (12)***}

Fig. 4: Pseudocode for the PD1 algorithm.

PD1's pseudocode is shown in Fig. 4. Based on this definition the following theorem can be proved asserting that PD1 always leads to an f_{app}-approximate solution [5]:

Theorem 3.1. *The final primal-dual solutions generated by PD1 satisfy all conditions (12) - (15) and so they also satisfy the relaxed slackness conditions with $f_1 = 1$, $f_2 = f_{app}$.*

4 The PD2 algorithm

Algorithm PD2 (unlike PD1) can be applied only if d_{ab} is a metric. In fact, PD2 represents a family of algorithms parameterized by a variable $\mu \in [\frac{1}{f_{app}}\ 1]$. PD2$_\mu$ will achieve slackness conditions (8), (9) with $f_1 = \mu f_{app}$ and $f_2 = f_{app}$. The reason for $\mu \geq \frac{1}{f_{app}}$ is because $f_1 < 1$ can never hold.

A main difference between algorithms PD1 and PD2$_\mu$ is that the former is always generating a feasible dual solution at any of its inner iterations while the latter will allow an intermediate dual solution of becoming infeasible. However, PD2$_\mu$ ensures that the (probably) infeasible dual solutions are "not too far away" from feasibility. This practically means that if these solutions are divided by a suitable factor, they will become feasible again. This method (i.e. turning an infeasible dual solution into a feasible one by division) is also known as *"dual-fitting"* [7] in the linear programming literature.

More specifically, the PD2$_\mu$ algorithm generates a series of intermediate pairs of primal-dual solutions with the following properties: all of them satisfy slackness condition (9) as an equality with $f_2 = \frac{1}{\mu}$ i.e.:

$$x_p \neq x_q \Rightarrow load^{x,y}_{pq} = \mu w_{pq} d_{x_p x_q} \tag{22}$$

In addition, the last intermediate pair satisfies the exact (i.e. $f_1 = 1$) slackness condition (8) which, as explained in section 3, reduces to:

$$y_p = \min_a ht^y_{p,a} \quad (23)$$

$$ht^y_{p,x_p} = \min_a ht^y_{p,a} \quad (24)$$

However, the dual solution of this last pair may be infeasible since (although it satisfies dual constraints (5) due to (23)) in place of constraints (6) it can be shown to satisfy only:

$$y_{pq,a} + y_{qp,b} \leq 2\mu w_{pq} d_{max} \quad \forall a,b \in L,\ (p,q) \in E \quad (25)$$

Nevertheless these conditions ensure that the last dual solution, say y, is not "too far away" from feasibility. This means that by replacing y with $y^{fit} = \frac{y}{\mu f_{app}}$ it takes only elementary algebra to show that y^{fit} is feasible and that the primal-dual pair (x, y^{fit}) (x being the last primal solution) satisfies the relaxed slackness conditions (8), (9) with $f_1 = \mu f_{app}$ and $f_2 = f_{app}$, thus leading to an f_{app}-approximate solution. This generation of y^{fit} (given y) is exactly what the DUAL_FIT routine does.

The main routines of PD2$_\mu$ are mostly similar to the ones of PD1 with only minor differences (see [5] for more details). The most important difference lies in the assignment of capacities to those interior edges pq, qp of $G_c^{x^k,\bar{y}^k}$ whose endpoints p, q have labels $\neq c$ at the start of the current c-iteration i.e. $x^k_p = a \neq c$ and $x^k_q = b \neq c$. Then, in place of (18), we instead define:

$$cap_{pq} = \mu w_{pq}(d_{ac} + d_{cb} - d_{ab}) \quad cap_{qp} = 0 \quad (26)$$

This also explains why d_{ab} must be a metric (or $cap_{pq} < 0$).

It can then be shown that PD2$_\mu$ indeed generates an f_{app}-approximate solution [5]. Furthermore, it holds that all PD2$_\mu$ algorithms with $\mu < 1$ are non-greedy algorithms meaning that neither the primal (nor the dual) objective function necessarily decreases (increases) per iteration. Instead, it is APF which constantly decreases but since APF is always kept close to the primal function the decrease in APF is finally reflected to the values of the primal function as well. However, a notable thing happens if $\mu = 1$. In that case, due to (22), the load between any neighbors p, q equals exactly their separation cost (i.e. $load_{pq}^{x^k,y^k} = w_{pq} d_{x^k_p x^k_q}$) and so it can be proved that APF coincides with the primal function. In addition it can be shown that PD2$_{\mu=1}$ is actually equivalent to the c-expansion graph cut algorithm [1] (that was interpreted only as a greedy local search technique up to now). This is stated in the next theorem [5] :

Theorem 4.1. *The label assignment x^{k+1} selected during a c-iteration of PD2$_{\mu=1}$ has the minimum primal cost among all label assignments resulting after a c-expansion of x^k.*

5 PD3: extending PD2 to the semimetric case

By modifying PD2$_\mu$, three different variations (PD3$_a$, PD3$_b$, PD3$_c$) may result that are applicable even if d_{ab} is a semimetric. For simplicity we will consider only the $\mu = 1$ case i.e. only variations of PD2$_{\mu=1}$. We also recall a fact that will prove to be useful for explaining the rationale behind the algorithms' definition: (OPTIMALITY CRITERION) *the load between any p, q represents a virtual separation cost which should be equal to the actual separation cost of p, q if the current primal-dual solutions are optimal.*

The main difficulty of extending PD2$_{\mu=1}$ to the case of a semimetric relates to all edges pq with capacity defined by (26) during a c-iteration i.e. all interior edges pq whose endpoints p, q are currently assigned labels $\neq c$ (i.e. $x^k_p = a \neq c,\ x^k_q = b \neq c$) while in addition the following inequality holds: $d_{ab} > d_{ac} + d_{cb}$. Hereafter we will call any such pair (p, q) a "conflicting pair" and the corresponding labels (a, b, c) a "conflicting label-triplet". Depending on the way we deal with such a "conflicting pair" three different variations of PD2$_{\mu=1}$ may arise.

PD3$_a$ algorithm: We choose to set $cap_{pq} = 0$ in place of (26). In this case it can be shown [5] that if the pair of labels c, b is assigned to p, q (by x^{k+1}) then the resulting load of p, q is greater than the actual separation cost of p, q or equivalently their virtual separation cost overestimates their actual separation cost contrary to the OPTIMALITY CRITERION above (in all other cases there is no such overestimation). Therefore, in this case, POSTEDIT_DUALS modifies the dual variables so that the equality between load and separation cost is restored by the start of the next iteration. No other differences between PD2$_{\mu=1}$ and PD3$_a$ exist.

One can prove that what PD3$_a$ actually does is to restore the triangle inequality for the current "conflicting label-triplet" by approximating d with $\bar{d}$ where $\bar{d}_{cb} = d_{ab} - d_{ac} > d_{cb}$[4], $\bar{d}_{ab} = d_{ab}$ and $\bar{d}_{ac} = d_{ac}$ i.e. by overestimating only the distance between labels c, b. It can also be shown that the primal-dual solutions generated by both PD3$_a$ and PD2$_{\mu=1}$ satisfy exactly the same conditions (22)-(25) and so PD3$_a$ is always guaranteed to lead to an f_{app}-approximate solution as well. *Therefore PD3$_a$ directly generalizes $PD2_{\mu=1}$ to the case of a semimetric d_{ab}.*

PD3$_b$ algorithm: We choose to set $cap_{pq} = +\infty$ and no further differences between PD3$_b$ and PD2$_{\mu=1}$ exist. This has the following important result: *the solution x^{k+1} produced at the current iteration can never assign the pair of labels c, b to the vertices p, q respectively (due to this fact we will call labels c, b the "excluded labels").* To prove this, it suffices to recall the "reassign rule" and observe that the directed edge pq can never become saturated by increasing its flow (since $cap_{pq} = +\infty$). Put otherwise, it is as if an infinite overestimation of the distance d_{cb} takes place by the algorithm. The price for that is that no guarantees about the algorithm's optimality can be provided. The reason is that the balance variables may now increase without bound (since $cap_{pq} = +\infty$) and so we cannot make sure that the generated dual solutions satisfy a "not too far away from feasibility" condition like (25). This in turn implies that no

[4] $d_{ab} - d_{ac} > d_{cb}$ holds since (a, b, c) is a "conflicting label-triplet".

dual-fitting technique can be applied in this case.

However, PD3$_b$ has a nice interpretation in the primal domain and can prove to be an excellent local minimizer due to the next theorem (that generalizes theorem 4.1):

Theorem 5.1. *[5] The solution x^{k+1} selected by PD3$_b$ during a c-iteration has the minimum primal cost among all solutions that result after a c-expansion of x^k, except for those that assign "excluded labels" to "conflicting pairs".*

Algorithm PD3$_c$: PD3$_c$ first adjusts (if needed) the dual solution y^k so that for any 2 neighbors p, q: $load_{pq}^{x^k,y^k} \leq w_{pq}(d_{ac} + d_{cb})$. After this initial adjustment the algorithm then continues in exactly the same way as the PD2$_{\mu=1}$ algorithm with the only difference being that d_{ab} in equation (26) is replaced with $\bar{d}_{ab}$ which is defined as: $\bar{d}_{ab} = load_{pq}^{x^k,y^k}/w_{pq}$. Obviously $\bar{d}_{ab} \leq d_{ac} + d_{cb}$ and so cap_{pq} in (26) is valid i.e. $cap_{pq} \geq 0$. No other differences exist.

It can be shown that $\bar{d}_{ab} \leq d_{ab}$ and so one can prove that PD3$_c$ actually tries to restore the triangle inequality for the current "conflicting label-triplet" by approximating d with $\bar{d}$ where $\bar{d}_{ab} \leq d_{ab}$, $\bar{d}_{ac} = d_{ac}$ and $\bar{d}_{cb} = d_{cb}$ i.e. PD3$_c$ works complementary to PD3$_a$: to restore the triangle inequality it underestimates $\bar{d}_{ab}$ instead of overestimating $\bar{d}_{cb}$ (like PD3$_a$). It can then be proved that PD3$_c$ always leads to an f'_{app}-approximate solution where this time $f'_{app} = f_{app} \cdot c_0$ with $c_0 = \max_{a \neq b} \frac{d_{ab}}{\hat{d}_{ab}}$ and $\hat{d}_{ab} = \min_{c \in L}(d_{ac} + d_{cb})$.

Finally, it should be noted that if d_{ab} is a metric then all algorithms PD3$_a$, PD3$_b$, PD3$_c$ reduce to algorithm PD2$_{\mu=1}$.

6 Experimental results

We have applied our algorithms to the stereo matching problem. In this case the graph G reduces to the image grid while the label cost for assigning disparity a to the image pixel p is set as: $c_{p,a} = |I_r(p + a) - I_l(p)|$ where I_l, I_r are the left and right images. We wanted to test how well our algorithms can handle both metric and semimetric distances d_{ab}. To this end we made use of the following property:

Given the pair of primal-dual programs in section 2, it can be shown [7] that any ratio $r = c^T x / b^T y$ (where x, y is a pair of integral-primal, dual feasible solutions) provides a suboptimality bound in the sense that x is guaranteed to be an r-approximation to the optimal integral solution. This fact proves to be very useful in practice: *By considering the minimum of all ratios $\{r^k = c^T x^k / b^T y^k\}_{k=1}^t$ obtained throughout the primal-dual schema, a suboptimality bound which is much tighter (i.e. much closer to 1) than the corresponding worst-case bound is usually obtained which allows one to better judge the goodness of a solution.*

This has been verified experimentally by using the well-known Tsukuba data set [6] as input to the stereo matching (see Fig. 5). No attempt has been made to model occlusions during stereo matching and all edge weights w_{pq} have been set equal to each other instead of properly adjusting their values based on image intensity edges (which would improve the results considerably for this specific example). This is so because our main goal was not to produce the best possible disparity by tweaking the input parameters but to test the tightness of the suboptimality bounds i.e. to test the effectiveness of these algorithms in minimizing the objective function. To this end, 3 different distances d_{ab} have been used during our experiments: the Potts distance $d_{1,ab} = 1 \; \forall a \neq b$ (a metric), the truncated linear distance $d^{\lambda}_{2,ab} = \min(\lambda, |a - b|)$ (also a metric) and the truncated quadratic distance $d^{\lambda}_{3,ab} = \min(\lambda, |a - b|^2)$ (a semimetric) where λ denotes the maximum allowed distance.

(a) (b) (c)

Fig. 5: (a) An image from the Tsukuba data set. **(b)** Disparities computed by algorithm PD1 **(c)** and PD2$_{\mu=1}$. 15 labels as well as the Potts distance (a metric) have been used in this example and so PD3$_a$, PD3$_b$, PD3$_c$ produce the same result with PD2$_{\mu=1}$.

Each experiment consisted of selecting one of our algorithms and a distance function and then using them to compute disparities for each of the Tsukuba stereo pairs. The averages of the obtained suboptimality bounds are shown in table 1. As can be seen from that table the per-instance suboptimality bounds are much tighter (i.e. much closer to 1) than the worst-case bounds f_{app} predicted in theory. *Therefore the algorithms have computed a nearly optimal solution in all cases, meaning that they can handle both metric and semimetric distances equally well.* Furthermore, these bounds explain in yet another way the great success that a-expansion techniques exhibit in practice since, as it was shown, the a-expansion algorithm is equivalent to PD2$_{\mu=1}$.

Besides testing the tightness of the per instance suboptimality bounds, we also wanted to test their accuracy i.e. how well they describe the true suboptimality of the generated solutions. To this end we applied our stereo matching algorithms to one image scanline at a time (instead of the whole image). In this case the graph G reduces to a chain and the true optimum can be computed using dynamic pro-

Distance	$\mathbf{f_{app}^{PD1}}$	$\mathbf{f_{app}^{PD2_{\mu=1}}}$	$\mathbf{f_{app}^{PD3_a}}$	$\mathbf{f_{app}^{PD3_b}}$	$\mathbf{f_{app}^{PD3_c}}$	$\mathbf{f_{app}}$
Potts	1.0104	1.0058	1.0058	1.0058	1.0058	2
Trunc. Linear$^{\lambda=5}$	1.0226	1.0104	1.0104	1.0104	1.0104	10
Trunc. Quad.$^{\lambda=5}$	1.0280	-	1.0143	1.0158	1.0183	10

Table 1: Average suboptimality bounds (colums 2-6) obtained for the Tsukuba data set. As expected they are much closer to 1 than the theoretical suboptimality bounds f_{app} listed in the last column. Therefore a nearly optimal solution is always obtained. Note that $PD2_{\mu=1}$ can be applied only if d_{ab} is a metric and in that case PD2$_{\mu=1}$, PD3$_a$, PD3$_b$ and PD3$_c$ (as well as their bounds) coincide.

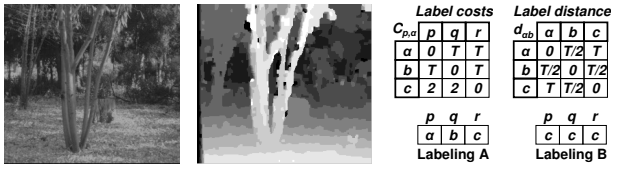
Distance	f_{app}^{PD1}	f_{true}^{PD1}	$f_{app}^{PD3_a}$	$f_{true}^{PD3_a}$	$f_{app}^{PD3_c}$	$f_{true}^{PD3_c}$
Potts	1.009	1.003	1.006	1.0004	1.006	1.0004
Trunc. Linear$^{\lambda=5}$	1.020	1.010	1.011	1.002	1.011	1.002
Trunc. Quad.$^{\lambda=5}$	1.025	1.013	1.013	1.001	1.016	1.003

Table 2: When applying stereo matching to one scanline at a time the obtained average suboptimalities (columns 2-4-6) are close to the true ones (columns 3-5-7) and can therefore be used for judging the goodness of generated solutions. Similar results also hold for the other algorithms but are not shown due to space limitations.

gramming. This way table 2 has been constructed which clearly shows that the per-instance bounds are relatively accurate and therefore reliable for judging the goodness of a generated solution. *Furthermore, this way we can always decide if a bad generated solution is the result of a bad minimization procedure or a bad modeling of the problem.*

We have also applied our algorithms to image pairs from the SRI tree image sequence (Fig. 6(a)). The selected pairs had a maximum disparity of 11 pixels. Given our algorithms' ability to handle both metrics and semimetrics equally well, the next semimetric has been used in this case: $d_{4,ab}^{\kappa,\lambda} = |a-b|$ if $|a-b| <= \kappa$, otherwise $d_{4,ab}^{\kappa,\lambda} = \lambda$. We always assume $\kappa < \lambda$. In this specific example we used $(\kappa, \lambda) = (2, 10)$. The rationale behind this distance is that it assigns a low penalty to small (i.e. $\leq \kappa$) changes in disparity (thus allowing surfaces with smoothly varying disparity like the slanted ground in the SRI image) but assigns a high penalty λ to large disparity gaps. Despite the fact that $d_{4,ab}^{\kappa,\lambda}$ is a semimetric our algorithms did not face any problem in efficiently minimizing the corresponding objective function and thus localizing the trees as well as the slanted ground in the SRI image (Fig. 6(b)). The average running times to convergence for the Tsukuba and SRI tree data sets have been 46 and 33 secs respectively on a 2.4GHz CPU.

The efficiency of our algorithms in the case where d_{ab} is a semimetric can be also illustrated by the synthetic example of Fig. 6(c). Although $PD3_a$, $PD3_b$ and $PD3_c$ are able to locate the global minimum for this example, the a-b swap algorithm of Boykov et al. [1] can get stuck at a local



(a) (b) (c)

Fig. 6: **(a)** One SRI tree image. **(b)** Computed disparities (11 labels) when using PD3$_a$ and semimetric $d_{4,ab}^{\kappa,\lambda}$ with $(\kappa, \lambda) = (2, 10)$. **(c)** A synthetic ML example where the graph G has 3 vertices $\{p, q, r\}$ and 2 edges $\{pq, qr\}$ while the labels L are $\{a, b, c\}$. Label costs c_{pa} and the distance d_{ab} (a semimetric) are shown. The a-b swap algorithm in [1] can get stuck in labeling A whose cost is T i.e. arbitrarily larger than the minimum cost which is 4 (labeling B). On the contrary $PD3_a$, $PD3_b$, $PD3_c$ can always locate the optimal labeling. Example taken from [1].

(a) (b) (c)

Fig. 7: **(a)** Noisy image. **(b)** Restored image with semimetric $d_{4,ab}^{\kappa,\lambda}$, $(\kappa, \lambda) = (2, 30)$ **(c)** and trunc. linear metric $d_{2,ab}^{\lambda}$, λ=30.

minimum arbitrarily far away from the true minimum.

Furthermore, the example of Fig. 7 illustrates the importance of using semimetric distances d_{ab} on the task of image restoration. The original image consists of 2 identical patterns placed vertically. Each pattern's intensity is kept constant along the horizontal direction and increases linearly with step 2 from top to bottom. The input image is then formed by corrupting the original image with white noise (Fig. 7(a)). Although our algorithms could restore the original image with only a few errors by use of the $d_{4,ab}^{\kappa,\lambda}$ semimetric (Fig. 7(b)), this was not the case when the truncated linear metric $d_{2,ab}^{\lambda}$ (or the potts metric) has been used despite the tweaking of the λ parameter. The best possible result with a metric is shown in Fig. 7(c). Analogous examples can be constructed for the stereo matching case.

Finally, it should be noted that for certain special cases of the ML problem our algorithms' approximation factors coincide with the so-called integrality gap [7] which is essentially the best possible approximation factor a primal-dual algorithm may achieve. Such is the case with the Generalized Potts model whose integrality gap is known to be 2 [3] i.e. equal to f_{app}. *This explains in yet another way why Graph-Cut techniques are so good in optimizing problems related to the Potts energy.* In conclusion, a new powerful optimization tool has been added to the arsenal of computer vision, capable of tackling a very wide class of problems.

References

[1] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE PAMI*, Nov. 2001.

[2] C. Chekuri, S. Khanna, J. Naor, and L. Zosin. Approximation algorithms for the metric labeling problem via a new linear programming formulation. In *SODA*, 2001.

[3] J. Kleinberg and E. Tardos. Approximation algorithms for classification problems with pairwise relaionships: metric labeling and markov random fields. *Journal of the ACM*, 2002.

[4] V. Kolmogorov and R. Zabih. What energy functions can be minimized via graph cuts? In *ECCV*, 2002.

[5] N. Komodakis and G. Tziritas. Approximate labeling via the primal-dual schema. Technical Report CSD-TR-05-01, Computer Science Department, February 2005.

[6] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *IJCV*, 47(1/2/3):7–42, April-June 2002.

[7] V. Vazirani. *Approximation Algorithms*. Springer, 2001.

[8] O. Veksler. *Efficient graph-based energy minimization methods in computer vision*. PhD thesis, Cornell University, 1999.

[9] R. Zabih and V. Kolmogorov. Spatially coherent clustering using graph cuts. In *CVPR*, 2004.