Approximate Labeling via the Primal-Dual Schema

Nikos Komodakis and Georgios Tziritas

Technical Report
CSD-TR-2005-01

February 1, 2005

Approximate Labeling via the Primal-Dual Schema

Nikos Komodakis and Georgios Tziritas

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

Technical Report
CSD-TR-2005-01

February 1, 2005

Abstract

A linear programming based framework is presented which is capable of providing combinatorial-based approximation algorithms to a certain class of NP-complete classification problems. The resulting algorithms utilize tools from the duality theory of linear programming and have guaranteed optimality properties. Finally, it is shown that state-of-the-art classification techniques can be derived merely as a special case of the considered framework.

Contents

1 Introduction

The Metric Labeling Problem, introduced by Kleinberg and Tardos [1], can capture a broad range of classification problems that arise in early vision (e.g. image restoration, stereo matching, image segmentation etc.). According to this problem, the task is to classify a set V of n objects by assigning to each object a label from a given set L of labels. Each labeling, i.e. a function $f : V \to L$, is associated with a certain cost which has 2 components. On one hand, for each $p \in V$ there is a label cost $c_{p,a} \geq 0$ for assigning label $a = f(p)$ to p. On the other hand, for each pair of objects p, q there is a so-called *separation cost* for assigning labels $a = f(p), b = f(q)$ to them. This separation cost is equal to $w_{pq}d_{ab}$ where the quantities w_{pq} are the edge weights of a graph $G = (V, E)$ and represent the strength of the relationship between p, q while d_{ab} is a distance function between labels which is assumed to be a metric[1]. Thus, the total cost equals $C(f) = \sum_{p \in V} c_{p,f(p)} + \sum_{(p,q) \in E} w_{pq}d_{f(p)f(q)}$ and the goal is to find a labeling f with the minimum cost. For a connection between Metric Labeling and Markov Random Fields the reader is referred to [1].

According to one class of approximation algorithms [1, 2, 3] the Metric Labeling problem is formulated as the Linear Programming relaxation of an integer program. This LP relaxation is then solved and a randomized rounding technique is being used to extract a near the optimum integer solution. These algorithms have good theoretical properties but since they require the solution of a linear program which, in the case of early vision problems, can grow very large, this makes them impractical to use. On the other hand, a variety of combinatorial-based approximation algorithms [4, 5, 6, 7, 8] have been developed. These state-of-the-art techniques are very efficient and have been applied with great success to many problems in computer vision [9, 10, 11, 12, 13]. However, they have been interpreted only as greedy local search techniques up to now.

The major contributions of this paper are:

- A linear programming based framework that makes use of the primal-dual schema in order to provide efficient (i.e. combinatorial-based) approximation algorithms to the Metric Labeling problem, thus bridging the gap between the two classes of approximation algorithms mentioned above.
- The derived algorithms have guaranteed optimality properties even in the more general case where d_{ab} is merely a semimetric[2]. These properties assert the existence of worst-case suboptimality bounds meaning that the minimum generated by any of the considered algorithms is always within a known factor of the global optimum.
- Graph-cut techniques introduced in [4] can be derived as a special case of our framework which is thus shedding further light on the essence of those algorithms. In particular, it is the first time that these (state of the art) algorithms are being interpreted not merely as greedy local search techniques but in terms of principles drawn from the theory of linear programming.
- In addition to the theoretical (worst-case) suboptimality bounds, the considered algorithms also provide per-instance suboptimality bounds for the generated solutions. This way one may better inspect how successful the convergence of the algorithm has been for each specific instance. In practice these per-instance bounds always prove to be much tighter than the theoretical (worst-case) ones, thus showing that the generated minimum is very close to the global optimum each time. Since graph-cut techniques can be included as a special case, this last fact explains in another way the great success that these techniques exhibit in practice.

2 The primal-dual schema

Consider the following primal-dual pair of linear programs:

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

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

[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. $d_{ab} = 0 \Leftrightarrow a = b,\ d_{ab} = d_{ba} \geq 0$

Primal-Dual Principle. *If x and y are integral-primal and dual feasible solutions satisfying:*

$$c^T x \leq f \cdot b^T y \tag{1}$$

then x is an f-approximation to the optimal integral solution x^* i.e. $c^T x \leq f \cdot c^T x^*$

The above principle, which is a a consequence of the Weak Duality Theorem, lies at the heart of any primal-dual technique. In fact, the various primal-dual methods mostly differ in the way that they manage to estimate a pair (x, y) satisfying the fundamental inequality (1). One very common way for that (but not the only one) is by relaxing the so-called primal complementary slackness conditions [14]:

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, y) also satisfies the Primal-Dual Principle with $f = \max_j f_j$ and therefore x is an f-approximation to the optimal integral solution.

Based on the above theorem, during a primal-dual f-approximation algorithm the following iterative schema is usually being used:

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

It should be noted that the exact slackness conditions (i.e. $f_j = 1$) are always satisfied by the primal-dual pair of optimal fractional solutions.

2.1 Metric Labeling as a linear program

Here we will consider the following integer programming formulation of the Metric Labeling problem, introduced in [2]:

$$\min \sum_{p \in V, a \in L} c_{p,a} x_{p,a} + \sum_{(p,q) \in E} w_{pq} \sum_{a,b \in L} d_{ab} x_{pq,ab} \tag{2}$$

$$\text{s.t.} \sum\nolimits_a x_{p,a} = 1 \qquad \forall\, p \in V \tag{3}$$

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

$$\sum\nolimits_b x_{pq,ab} = x_{p,a} \qquad \forall\, a \in L,\ (p,q) \in E \tag{5}$$

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

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 exactly the same thing and should coincide. So, in order to reduce the number of variables in the primal problem, we adopt the convention that for any neighbors p, q exactly one of $(p,q),\ (q,p) \in E$. This in turn implies that exactly one of the variables $x_{pq,ab},\ x_{qp,ba}$ is being used for each pair of neighbors p, q. The notation "$p \sim q$" will hereafter denote the fact that p, q are neighboring vertices and will mean "either $(p,q) \in E$ or $(q,p) \in E$". The first constraints (3) simply express the fact that each vertex must receive a label while constraints (4), (5) 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. The dual of that linear program has the following form:

$$\begin{aligned} \max\ & \sum\nolimits_p y_p \\ \text{s.t.}\ & y_p \leq c_{p,a} + \sum_{q : q \sim p} y_{pq,a} \qquad \forall p \in V,\ a \in L \\ & y_{pq,a} + y_{qp,b} \leq w_{pq} d_{ab} \qquad \forall a, b \in L, (p,q) \in E \end{aligned}$$

To each vertex p, there corresponds one dual variable y_p. Also, to each pair of neighboring vertices p, q (and any label a), there correspond 2 dual variables $y_{pq,a}$ and $y_{qp,a}$. Note that this is in contrast to the primal problem where, for each pair of neighboring vertices p, q (and any labels a, b), only one of the 2 variables $x_{pq_ab}, x_{qp,ba}$ is being used depending on whether $(p,q) \in E$ or $(q,p) \in E$. All the dual variables $\{y_{pq,a}\}_{p,q:p\sim q}^{a\in L}$[3] will be called "balance variables" hereafter while also for each pair $y_{pq,a}, y_{qp,a}$ of these balance variables we will say that $y_{pq,a}$ is the conjugate balance variable of $y_{qp,a}$ (and vice versa) or equivalently that $y_{pq,a}, y_{qp,a}$ are conjugate variables.

By defining the auxiliary variables $ht^y_{p,a}$ as:

$$ht^y_{p,a} \equiv c_{p,a} + \sum_{q:q\sim p} y_{pq,a} \tag{6}$$

the dual problem is trivially transformed into:

$$\begin{aligned} \max\ & \sum\nolimits_p y_p \\ \text{s.t. } & y_p \le ht^y_{p,a} && \forall p \in V,\ a \in L && (7) \\ & y_{pq,a} + y_{qp,b} \le w_{pq} d_{ab} && \forall a, b \in L, (p,q) \in E && (8) \end{aligned}$$

The dual variables $ht^y_{p,a}$ will be called "height variables" hereafter. 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 need to be specified. The auxiliary height variables $ht^y_{p,a}$ are then computed by (6).

2.2 Relaxed complementary slackness conditions

The relaxed primal complementary slackness conditions, related to the specific pair of primal-dual linear programs of the previous section, are:

$$x_{p,a} > 0 \Rightarrow y_p \ge c_{p,a}/f_1 + \sum\nolimits_{q:q\sim p} y_{pq,a} \tag{9}$$

$$x_{pq,ab} > 0 \Rightarrow y_{pq,a} + y_{qp,b} \ge w_{pq} d_{ab}/f_2 \tag{10}$$

During the primal-dual schema we will be considering only feasible $\{0,1\}$-primal solutions. It is not difficult to see that such solutions can be completely specified (i.e. all primal variables $x_{p,a}, x_{pq,ab}$ can be estimated) once we know what label has been assigned to each vertex. For this reason, 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, $x_{p,a} > 0$ (i.e. $x_{p,a} = 1$ since we are dealing only with $\{0,1\}$ solutions) is equivalent to $x_p = a$ and so the relaxed primal complementary slackness conditions (9) are trivially reduced to:

$$y_p \ge c_{p,x_p}/f_1 + \sum\nolimits_{q:q\sim p} y_{pq,x_p} \tag{11}$$

In a similar fashion, $x_{pq,ab} > 0$ is equivalent to $x_p = a$ and $x_q = b$ and the complementary slackness conditions (10) are trivially reduced to:

$$x_p \ne x_q \Rightarrow y_{pq,x_p} + y_{qp,x_q} \ge w_{pq} d_{x_p x_q}/f_2 \tag{12}$$

$$x_p = x_q = a \Rightarrow y_{pq,a} + y_{qp,a} = 0 \tag{13}$$

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

During any of the algorithms that will follow our objective will be to find feasible solutions x, y satisfying the above complementary slackness conditions (11), (12) and (13). The last slackness conditions (13) simply express the fact that conjugate balance variables should be opposite to each other. For this reason we set by definition:

$$y_{qp,a} \equiv -y_{pq,a} \quad \forall\, (p,q) \in E,\ a \in L \tag{14}$$

Therefore slackness condition (13) will always be true hereafter and so we will have to take care for fulfilling only conditions (11) and (12).

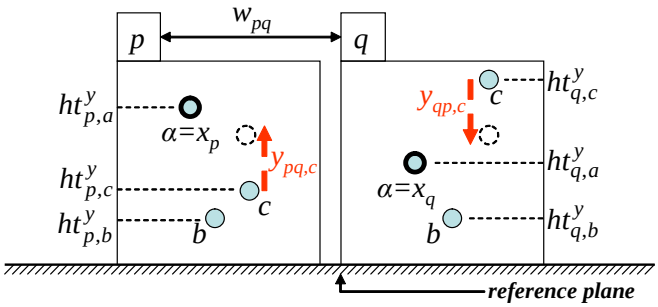

Fig. 1: A visualization of the dual for a very simple instance of the Metric Labeling Problem where the graph G consists of just 2 neighboring vertices p, q and the set of labels L is equal to $\{a, b, c\}$. To each of the vertices p, q there corresponds a separate set of labels $\{a, b, c\}$ (each label is represented by a circle) and all of these labels are located at certain heights relative to a common reference plane. The values of these heights are set equal to the dual variables ht and therefore depend on the balance 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 labels which are currently assigned to vertices p, q are drawn with a thicker line.

2.3 An intuitive view of the dual variables and some extra terminology

A way of viewing/visualizing the dual variables, that will prove useful when designing our approximation algorithms later, is the following: for each vertex p, we consider a separate copy of the complete set of labels L. One then may assume 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 (6), 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 (14)), 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 that will be used frequently throughout this document. Let x, y be any pair of integral-primal, dual solutions. We will call label x_p the assigned label to p or equivalently the *active label at* p. We will also refer to the height of an assigned label to vertex p (i.e. ht^y_{p,x_p}) as merely *the height of* p. Based on this definition, a function (denoted as $APF^{x,y}$ hereafter) which will play an important role in all of the considered primal-dual algorithms is the sum of the heights of all vertices i.e. $APF^{x,y} = \sum_p ht^y_{p,x_p}$. If x, y satisfy the exact (i.e. $f_1 = 1, f_2 = 1$) slackness conditions (11),(12), then it is easy to prove that APF coincides with the value of the primal objective function while if the relaxed slackness conditions hold then it is also easy to prove that APF stays close to the actual value of the primal objective function. For this reason APF will be called the *"Approximate Primal Function"* hereafter.

Another significant concept is that of an *active balance variable*. We define as an *active balance variable at a vertex* p any balance variable belonging to the following set $\{y_{pq,x_p}\}_{q:q\sim p}$ (i.e. any balance variable of the form y_{pq,x_p} where q is any neighboring vertex of p). Based on the "active balance variable" concept, we may also introduce another very important quantity which is called the *load* between two neighbors p, q ($load^{x,y}_{pq}$) and is equal to the sum of the 2 active balance variables y_{pq,x_p}, y_{qp,x_q} i.e. $load^{x,y}_{pq} = y_{pq,x_p} + y_{qp,x_q}$. If relaxed slackness conditions (12) are to be satisfied, then it is easy to see that the load between p, q can be thought of as a virtual separation cost which is always a rough approximation of the actual separation cost $w_{pq}d_{x_px_q}$ between p, q. This can be verified as follows: if the separation cost between p, q is zero (i.e. $x_p = x_q$) then so is $load^{x,y}_{pq}$ due to (14). While if the separation cost is not zero (i.e. $x_p \neq x_q$) then it holds that $w_{pq}d_{x_px_q}/f_2 \leq load^{x,y}_{pq} \leq w_{pq}d_{x_px_q}$ where the 1^{st} inequality is due to slackness conditions (12) and the 2^{nd} one is due to the dual constraints (8).

Another useful thing to note is that there exists a direct relationship between the value of the APF function and the loads. In particular, it holds that:

$$APF^{x,y} = \sum_p c_{p,x_p} + \sum_{(p,q)\in E} load^{x,y}_{pq} \tag{15}$$

[3] According to our notation the set $\{y_{pq,a}\}^{a\in L}_{p,q:p\sim q}$ equals the set $\{y_{pq,a}, y_{qp,a}\}^{a\in L}_{p,q:(p,q)\in E}$

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** $LabelChange = 1$ (i.e. at least one vertex has changed its label) **then** goto 2;
11: **if** algorithm $\neq$ PD1 **then** $y^{fit} \leftarrow$DUAL_FIT(y^k);

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

One can very easily verify the above equation as follows:

$$APF^{x,y} = \sum_p ht^y_{p,x_p} = \sum_p \Big(c_{p,x_p} + \sum_{q:q\sim p} y_{pq,x_p} \Big) = \sum_p c_{p,x_p} + \sum_{(p,q)\in E} \Big(y_{pq,x_p} + y_{qp,x_q} \Big)$$
$$= \sum_p c_{p,x_p} + \sum_{(p,q)\in E} load^{x,y}_{pq}$$

2.4 Applying the primal-dual schema to Metric Labeling

The majority of the approximation algorithms that will be presented here achieve an approximation factor of $f_{app} = 2\frac{d_{max}}{d_{min}}$ with $d_{min} \equiv min_{a\neq b} d_{ab}$ and $d_{max} \equiv max_{a\neq b} d_{ab}$. The distinction between the considered algorithms will be lying in the exact values they assign to the constants f_1, f_2 that are used in the relaxed complementary slackness conditions (11),(12). Another important difference will be that some of them are applicable even in the more general case of d_{ab} being a semimetric[4].

The basic structure of any of the considered algorithms can be seen in Fig. 2. The initial primal-dual solutions are generated inside INIT_PRIMALS and INIT_DUALS respectively. During each 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). It should be noted that among all balance variables of y^k (i.e. $\{y^k_{pq,a}\}^{a\in L}_{p,q:p\sim q}$) only the balance variables of the c labels (i.e. $\{y^k_{pq,c}\}_{p,q:p\sim q}$) 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 if no vertex changes its label during the current outer iteration the algorithm then terminates.

The role of the routines which are being executed during an inner c-iteration is as follows: the role of PREEDIT_DUALS is to edit solution y^k into solution $\bar{y}^k$ that is going to be used as an input to the UPDATE_DUALS_PRIMALS routine. That routine is responsible for the main update of the primal and dual variables and to this end it generates the pair of solutions $(x^{k+1}, \bar{y}^{k+1})$. Finally, POSTEDIT_DUALS applies further modifications to $\bar{y}^{k+1}$ thus producing the next dual solution y^{k+1}. This solution y^{k+1} along with x^{k+1} constitute the next primal-dual pair of solutions. The algorithms to be considered are named PD1, PD2, PD3 and the DUAL_FIT routine, which is being used only in the last two of them, serves only the purpose of applying a scaling operation to the last dual solution (as we shall later see).

Since we will be dealing only with approximation algorithms, we may hereafter assume w.l.o.g. that all coefficients $c_{p,a}, w_{pq}, d_{ab}$ of the Metric Labeling integer program (2) are nonnegative integers. If this was not true, then we could approximate (to any precision) those coefficients by rational numbers thus generating an instance of the Metric Labeling problem which in turn can be trivially transformed into an integer program (2) with integral coefficients. We could then apply all of our approximation algorithms to this last integer program.

[4]The linear programming formulation of Metric Labeling is still valid

3 The PD1 algorithm

During this section we will assume that d_{ab} is a semimetric. In the particular case of the PD1 algorithm our goal will be to find feasible solutions x, y satisfying slackness conditions (11), (12) with $f_1 = 1$ and $f_2 = f_{app}$. By replacing $f_1 = 1$ in (11) that condition becomes $y_p \geq ht^y_{p,x_p}$. Since it also holds that $y_p \leq \min_a ht^y_{p,a}$ (by the dual constraints (7)), it is easy to see that (11) reduces to the following 2 equations:

$$y_p = \min_a ht^y_{p,a} \tag{16}$$

$$ht^y_{p,x_p} = \min_a ht^y_{p,a} \tag{17}$$

In addition, by making use of the definition of the load as $load^{x,y}_{pq} = y_{pq,x_p} + y_{qp,x_q}$ and by replacing $f_2 = f_{app}$ in (12) that condition becomes equivalent to:

$$x_p \neq x_q \Rightarrow load^{x,y}_{pq} \geq w_{pq} d_{x_p x_q} / f_{app} \tag{18}$$

Therefore the objective of PD1 is to find feasible x, y satisfying conditions (16)-(18). PD1 uses the following strategy to achieve its goal: during its execution it generates a series of primal-dual pairs of solutions, one primal-dual pair per iteration. At each iteration it makes sure that conditions (16) and (18) 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 enforces that the current dual solution always satisfies the following constraints:

$$y_{pq,a} \leq w_{pq} d_{min}/2 \quad \forall\, a \in L,\ p \sim q \tag{19}$$

To see that (19) ensures feasibility, it is enough to observe that due to this constraint the following inequality can be derived: $y_{pq,a} + y_{qp,b} \leq 2 w_{pq} d_{min}/2 = w_{pq} d_{min} \leq w_{pq} d_{ab}$ and so the dual constraints (8) hold true. This implies that solution y is indeed feasible since the other dual constraints (7) already hold true due to condition (16).

All that remains then for PD1 to achieve its goal is just to ensure that after a finite number of iterations slackness conditions (17) are satisfied as well. This last objective (i.e. driving the primal-dual pairs towards satisfying (17)) will be the final and key issue for the success of the considered algorithm. To this end, PD1 will be trying to ensure that the number of vertices p for which equation (17) holds true increases after each one of its iterations.

3.1 An intuitive understanding of the algorithm

Let us now give some "feel" for how PD1 is really trying to achieve that last objective. Before proceeding, it should be noted that enforcing condition (16) is always a trivial thing to do (we simply need to set each dual variable y_p equal to $\min_a ht^y_{p,a}$), so we do not really have to worry about that condition throughout PD1.

Let x, y be the current pair of integral-primal and dual feasible solutions satisfying all required conditions (16)-(19) except for (17). That condition simply requires that the label x_p assigned to any vertex must be "lower" than all other labels at that vertex. So let p be a vertex for which this condition fails i.e. one of its labels, say c, is located "below" the active label x_p at that vertex. To restore (17) we need to raise label c up to x_p by increasing one of the balance variables $\{y_{pq,c}\}_{q:q\sim p}$. But as already mentioned, each time we increase $y_{pq,c}$ its conjugate variable $y_{qp,c}$ decreases and so the height of c at the neighboring vertex q decreases as well. This may have as a result that label c at q gets below the active label x_q. Therefore we must be careful which balance variables we choose to increase and by how much or otherwise we may break condition (17) for some neighboring vertex q. This means that the increase/decrease of the balance variables must proceed in an optimal way so that condition (17) is restored for as many vertices as possible. Based on this observation, during any iteration of the algorithm 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 keep the heights of all active labels fixed and then for each vertex we try so that all of its labels are raised above the vertex's active label. Of course, we only need to do that for the labels that are "below" the active label. To this end we need to update the dual balance variables in an optimal way. As we shall see any update of the balance variables can be simulated by pushing flow through an appropriately constructed capacitated graph while the optimal update can be achieved by pushing the maximum flow through that graph. Of course, care must also be taken so that constraints (19) do not become violated during this update of the dual variables or else the resulting dual solution might not be feasible. Constraints (19) impose upper bounds on the values of the balance variables, restricting this way the maximum allowed increase that we may apply to the height of a label.

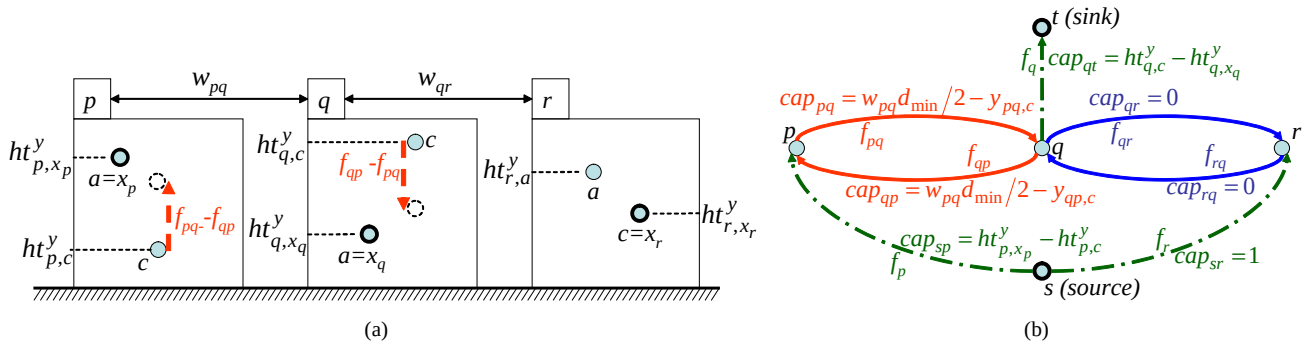

Fig. 3: (a) An arrangement of labels (represented by circles) for a simple instance of the Metric Labeling problem consisting of 3 vertices p, q, r and 2 edges pq, qr with weights w_{pq}, w_{qr}. The label set is $L = \{a, c\}$. The circles with the thicker line represent the active labels. Also, the 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 capacitated graph $G_c^{x,y}$ is shown. A maximum flow algorithm is applied to this graph for updating the dual variables. Interior edges are drawn with a solid line while exterior edges are drawn with a dashed line. Capacities of both interior and exterior edges are also shown.

- Primal variables update: after the optimal rearrangement of the labels' heights, there might still be some vertices whose active labels are not the ones with the lowest height (among all labels at the vertex), violating this way condition (17). We select a suitable subset of these vertices and assign to them new labels which are at lower heights than the previous active labels so that the resulting primal solution is taken closer to satisfying (17) too. The reason we may not be able to do that for all the vertices is that we must still take care that the other slackness conditions (18) are maintained as well. Nevertheless, the number of vertices violating (17) decreases per iteration and so by keep repeating this update of the primal and dual variables it can be shown that in the end the active label of each vertex will have the lowest height at that vertex and the last primal-dual pair will therefore satisfy all required conditions (16)-(19).

3.2 Constructing the capacitated graph $G_c^{x,y}$

The rearrangement of the label heights takes place in groups. Given as input a current primal-dual pair of solutions (x, y) and a label c, UPDATE_DUALS_PRIMALS(c, x, y) rearranges only the heights of the c labels. To this end it changes solution y into solution y' by changing only the balance 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 goal of this update will be to have the resulting heights $ht^{y'}_{p,c}$ rearranged in an optimal way so that as many of the c labels as possible end up being above the currently active labels. In addition, we need to make sure that the new dual solution y' does not break conditions (19).

In the simple case presented in Fig. 3(a), for example, we would like to have label c at p move at least as high as label a at p (the active label of p) without, at the same time, label c at q gets lower than label a at q (the active label of q). Label c at r does not need to move at all since it is already the active label of vertex r and has the lowest height in there. It turns out that in the general case the update of both the balance variables and the labels' heights can be simulated by pushing flow through an appropriately constructed directed graph $G_c^{x,y} = (V_c^{x,y}, E_c^{x,y}, C_c^{x,y})$ with capacities $C_c^{x,y}$. In fact, as we shall see later, the optimal update corresponds to pushing the maximum amount of flow through that graph. Such a capacitated graph, associated to the simple problem of Fig. 3(a), is presented in Fig. 3(b).

Let us now explain how such a graph can be constructed as well as how pushing flow through that graph relates to the update of the dual variables. 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 edges (drawn with solid lines in Fig. 3(b)) and exterior edges (drawn with dashed lines 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} leaving p through pq represents the increase of the balance variable $y_{pq,c}$ while the amount of flow f_{qp} entering p through interior edge qp represents the decrease of the same variable $y_{pq,c}$. The total change of $y_{pq,c}$ will therefore be:

$$y'_{pq,c} = y_{pq,c} + f_{pq} - f_{qp} \tag{20}$$

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). It

is then obvious that $y'_{pq,c} = -y'_{qp,c}$ and so conjugate balance variables remain opposite to each other as they should.

Based on (20), it is also easy to see that the capacity cap_{pq} of an interior edge pq represents the maximum allowed increase of the $y_{pq,c}$ variable (attained if $f_{pq} = cap_{pq}, f_{qp} = 0$) and therefore the quantity $y_{pq,c} + cap_{pq}$ represents the maximum value of the new balance variable $y'_{pq,c}$. Similar conclusions can be drawn regarding the capacity cap_{qp} of the reverse edge qp. Based on these observations the capacities cap_{pq}, cap_{qp} are assigned as follows: if the current label of p (or q) is already c then we want to keep the height of c at p (or q) fixed during the current iteration and so the capacity for all interior edges in or out of p (or q) must be zero. Therefore:

$$x_p = c \text{ or } x_q = c \Rightarrow cap_{pq} = cap_{qp} = 0 \tag{21}$$

Otherwise (i.e. $x_p \neq c, x_q \neq c$), we set the capacity of the edges pq, 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 (19) are therefore maintained for the new dual solution y'. For this reason we set:

$$y_{pq,c} + cap_{pq} = w_{pq}d_{min}/2 = y_{qp,c} + cap_{qp} \tag{22}$$

Exterior edges: Each internal node p will be connected to either the source node s or the sink node t through an exterior edge. Whether we choose the source or the sink depends on the relative heights at vertex p of the labels c and x_p (the active label of p). 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 (as explained in section 3.1) we would like to raise label c by exactly as much as needed so that it reaches label x_p. This is the case, for example, with vertex p in Fig. 3(a) where we would like that label c at p reaches label a at 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 has the following interpretation: it represents the total increase in the height of label c (taking into account the contribution from the change of all balance variables):

$$ht^{y'}_{p,c} = ht^y_{p,c} + f_p \tag{23}$$

To verify this, it is enough to combine the flow conservation constraint at node p which can be easily seen to reduce to:

$$f_p = \sum_{q:q\sim p} \left(f_{pq} - f_{qp}\right) \tag{24}$$

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 (20)). Indeed, it then follows that:

$$\begin{aligned} ht^y_{p,c} + f_p &= \Big(c_{p,c} + \sum_{q:q\sim p} y_{pq,c}\Big) + f_p \\ &= \Big(c_{p,c} + \sum_{q:q\sim p} y_{pq,c}\Big) + \sum_{q:q\sim p} \left(f_{pq} - f_{qp}\right) \\ &= \Big(c_{p,c} + \sum_{q:q\sim p} y_{pq,c}\Big) + \sum_{q:q\sim p} \left(y'_{pq,c} - y_{pq,c}\right) = c_{p,c} + \sum_{q:q\sim p} y'_{pq,c} = ht^{y'}_{p,c} \end{aligned}$$

Based on this observation, the capacity cap_{sp} of the edge sp represents the maximum allowed raise in the height of c. Therefore, since we need to raise c only as high as the current label of p but not higher than that, we simply set this capacity as follows:

$$cap_{sp} = ht^y_{p,x_p} - ht^y_{p,c} \tag{25}$$

The capacity of edge sp in Fig. 3(b) is defined this way.

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. In such a case, we connect the node p to the sink node t through a directed edge pt. This time the flow passing through edge pt will reflect the total decrease in the height of c (taking again into account the contribution from the change of all balance variables):

$$ht^{y'}_{p,c} = ht^y_{p,c} - f_p \tag{26}$$

The capacity of this edge therefore represents the maximum decrease in the height of label c and since this label must remain "above" x_p, capacity cap_{pt} is defined, in a similar fashion to (25), as:

$$cap_{pt} = ht^y_{p,c} - ht^y_{p,x_p} \tag{27}$$

In Fig. 3(a), for example, label c at vertex q needs to remain above label a at q and so in Fig. 3(b) the capacity of edge qt is defined by applying (27) to vertex q.

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 above, we again connect the source node s to node p through directed edge sp. This time, however, no flow passes through the interior edges incident to p (i.e. $f_{pq} = f_{qp} = 0$ for any neighboring vertex q) since these edges have zero capacity now (see (21)). So $f_p = 0$ (due to (24)) and no flow passes through edge sp as well which in turn implies that the height of label c will not change (see (23)), as was intended. By convention we set the capacity cap_{sp} of edge sp equal to one:

$$cap_{sp} = 1 \tag{28}$$

Vertex r in Fig. 3(a) belongs to this case and this is the reason why we set $cap_{sr} = 1$ in the graph of Fig. 3(b).

3.3 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 the role of this routine will be 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 capacitated graph $G_c^{x^k,\bar{y}^k}$ of section 3.2. In the specific case of PD1, no update takes place inside PREEDIT_DUALS and so $\bar{y}^k = y^k$.

UPDATE_DUALS_PRIMALS$(c, x^k, \bar{y}^k)$: After the construction of the graph $G_c^{x^k,\bar{y}^k}$ (as explained in section 3.2), a maximum flow algorithm [15] is applied to it and the resulting flows on interior edges are used in updating the dual balance variables. More specifically, only the balance variables of the c labels are updated as follows (see (20)):

$$\bar{y}_{pq,c}^{k+1} = \bar{y}_{pq,c}^{k} + f_{pq} - f_{qp} \tag{29}$$

Therefore the heights of all c labels will also change as (see (23), (26)):

$$ht_{p,c}^{\bar{y}^{k+1}} = ht_{p,c}^{\bar{y}^{k}} + \begin{cases} f_p & \text{if } p \text{ is connected to node } s \\ -f_p & \text{if } p \text{ is connected to node } t \end{cases} \tag{30}$$

In the toy example of Fig. 4(a) you can see an initial arrangement of labels' heights based on the values that the dual variables take at the start of a c-iteration while in Fig. 4(b) you can see the resulting rearrangement of the labels' heights due to the update of the dual variables after applying the maximum-flow algorithm to the associated graph $G_c^{x^k,\bar{y}^k}$ of Fig. 4(d). The corresponding flows are also shown in that figure.

Based on the resulting heights, we now need to update the primal variables i.e. assign new labels to the vertices of G. Since only the heights of c labels have been altered, the latter amounts to deciding whether each vertex keeps its current label or is assigned the label c. On one hand, this must be done so that the labels of any vertex p are taken closer to satisfying (17) (i.e. the active label of p should also be the "lowest" one at p). This practically means that if label c at p has not managed to get "above" the active label of p, then we need to assign c as the new label of that vertex. For example, in the case of the updated heights of Fig. 4(b) we would like vertex p to be assigned label c while the rest of the vertices q, r should maintain their current labels. On the other hand, we must also take care maintaining condition (18). It turns out that both of the above criteria can be fulfilled by considering the flows through both interior and exterior edges of $G_c^{x^k,\bar{y}^k}$ and making 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[5] path between the source and node p (otherwise p keeps its current label i.e. $x_p^{k+1} = x_p^k$).*

In Fig. 4(c) you can see the new assignment of labels to vertices that has resulted after applying the above rule to the graph of 4(d). Vertex p gets indeed a new label because edge sp is unsaturated while q, r maintain their active labels since no unsaturated path to them exists in $G_c^{x^k,\bar{y}^k}$. As mentioned above, the new assignment obviously coincides with what we would like to achieve initially. Before proceeding let us state some very useful properties resulting out of the choice of the reassign rule (these properties will hold for all of the considered algorithms):

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

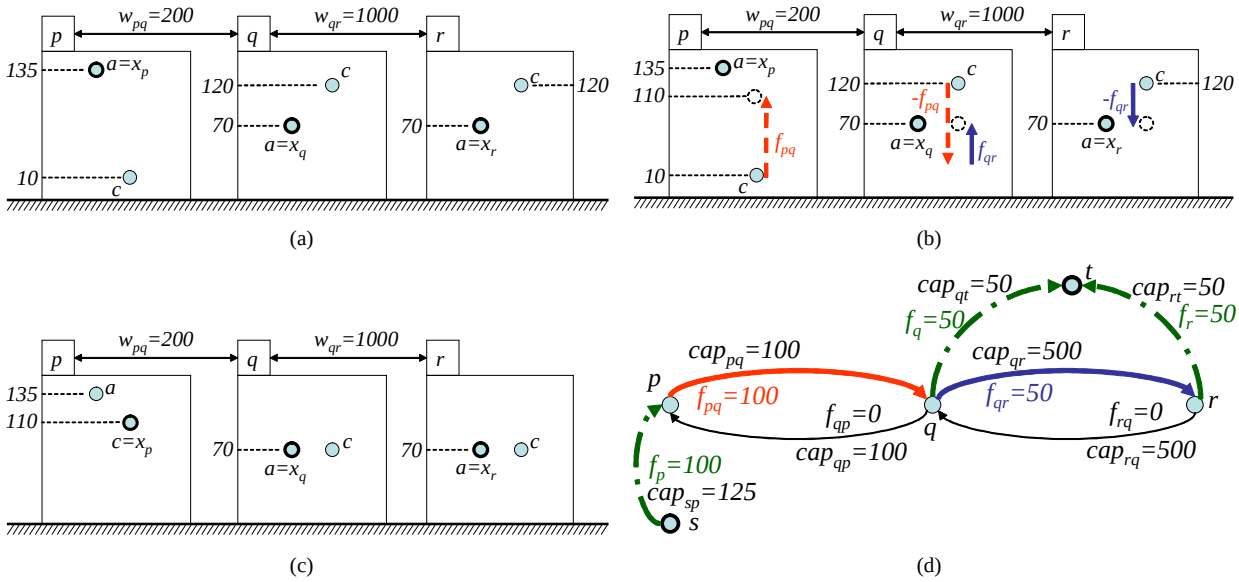

Fig. 4: **(a)** The initial arrangement of the labels' heights at the start of the current c-iteration for a toy example of the Metric Labeling problem. All of the vertices p, q, r are currently assigned label a (as indicated by the circles with the thicker line). **(b)** The red and blue arrows show how the c labels will move due to the update of the balance variables after applying a maximum flow algorithm to the graph in (d). Label movements due to changes in balance variables that are conjugate to each other are drawn with the same line style and color. Furthermore, the dashed circles indicate the final positions of the labels. **(c)** The new active labels (thick circles) that have been selected based on the "reassign rule" are shown here. Only vertex p had to change its label into c since the exterior edge sp of the graph in (d) is unsaturated. **(d)** The associated capacitated graph (assuming that initially all balance variables are zero) and the resulting flows after applying a maximum flow algorithm. The flows f_p, f_q, f_r at the exterior edges are equal to the total change in the height of the c labels at p, q, r respectively. In this example the Potts metric has been used for the distance d_{ab} (i.e. if $a \neq b \Rightarrow d_{ab} = 1$).

Properties 3.1. *Let p, q be two neighboring vertices i.e. $p \sim q$. Then during a c-iteration:*

(a) $a \neq c \Rightarrow \bar{y}^{k+1}_{pq,a} = \bar{y}^{k}_{pq,a},\ ht^{\bar{y}^{k+1}}_{p,a} = ht^{\bar{y}^{k}}_{p,a}$

(b) $x^k_p = c \Rightarrow x^{k+1}_p = c,\ \left(\bar{y}^{k+1}_{pq,c}, \bar{y}^{k+1}_{qp,c}\right) = \left(\bar{y}^{k}_{pq,c}, \bar{y}^{k}_{qp,c}\right),\ ht^{\bar{y}^{k+1}}_{p,x^{k+1}_p} = ht^{\bar{y}^{k}}_{p,x^{k}_p}$

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

(d) $ht^{\bar{y}^{k+1}}_{p,x^{k+1}_p} \leq ht^{\bar{y}^{k+1}}_{p,c}$

(e) *if p is assigned label c but q keeps its current label (i.e. $x^{k+1}_p = c$ and $x^{k+1}_q = x^k_q$), then $\bar{y}^{k+1}_{pq,c} = \bar{y}^{k}_{pq,c} + cap_{pq}$ i.e. the balance variable $\bar{y}^{k+1}_{pq,c}$ attains its maximum value*

(f) *(APF monotonicity) $APF^{x^{k+1},\bar{y}^{k+1}} \leq APF^{x^k,\bar{y}^k}$ Furthermore, if at least one change of label has taken place during the current c-iteration then $APF^{x^{k+1},\bar{y}^{k+1}} < APF^{x^k,\bar{y}^k}$*

Proof:

(a) This property follows directly from the fact that only the balance variables of the c labels are updated during a c-iteration, by definition.

(b) Due to $x^k_p = c$ and (21), the capacities of all interior edges $p\acute{q}, \acute{q}p$ with $\acute{q}$ adjacent to p (i.e. $\acute{q} \sim p$) will be zero and so no flow can pass through them i.e.:

$$f_{p\acute{q}} = f_{\acute{q}p} = 0 \quad \forall\, \acute{q} : \acute{q} \sim p \tag{31}$$

If we then apply the flow conservation at node p (24), we can see that the flow through edge sp will be zero as well (i.e. $f_p = 0$) which in turn implies that the edge sp is unsaturated (since $cap_{sp} = 1$ by (28)). Therefore by the reassign rule it will also be $x_p^{k+1} = c$. Finally, the equality $\left(\bar{y}_{pq,c}^{k+1}, \bar{y}_{qp,c}^{k+1}\right) = \left(\bar{y}_{pq,c}^{k}, \bar{y}_{qp,c}^{k}\right)$ follows directly from applying (31) to $\acute{q} = q$ and then using (29) while the other equality $ht_{p,x_p^{k+1}}^{\bar{y}^{k+1}} = ht_{p,x_p^k}^{\bar{y}^k}$ follows from $f_p = 0$ and (30).

(c) if $x_p^{k+1} \neq c$ then it will necessarily hold $x_p^{k+1} = x_p^k$ since by the reassign rule a vertex is either assigned label c or keeps its current label x_p^k. Therefore it will also be $x_p^k \neq c$. So by setting $x_p^{k+1} = x_p^k = a \neq c$ we may now apply property (a) and easily conclude that $ht_{p,x_p^{k+1}}^{\bar{y}^{k+1}} = ht_{p,x_p^k}^{\bar{y}^k}$ which means that the property holds in this case.

Therefore we may hereafter assume that $x_p^{k+1} = c$. In that case, if p is connected to the source node s then we may easily verify the property as follows:

$$\begin{aligned} ht_{p,x_p^{k+1}}^{\bar{y}^{k+1}} = ht_{p,c}^{\bar{y}^{k+1}} &= ht_{p,c}^{\bar{y}^k} + f_p && \text{by (30)} \\ &\leq ht_{p,c}^{\bar{y}^k} + cap_{sp} \\ &= ht_{p,c}^{\bar{y}^k} + \left(ht_{p,x_p^k}^{\bar{y}^k} - ht_{p,c}^{\bar{y}^k}\right) && \text{by (25)} \\ &= ht_{p,x_p^k}^{\bar{y}^k} \end{aligned}$$

Let us now consider the case where p is connected to the sink t: since we assume $x_p^{k+1} = c$ the reassign rule implies that there must be an unsaturated path, say $s \rightsquigarrow p$, from s to p. But it must then hold $f_p = cap_{pt}$ or else there will also be an unsaturated path $s \rightsquigarrow p \rightarrow t$ between the source and the sink which is impossible due to the max-flow min-cut theorem [15]. Combining this fact (i.e. $f_p = cap_{pt}$) with (30) and the definition of cap_{pt} in (27) it then follows that:

$$\begin{aligned} ht_{p,x_p^{k+1}}^{\bar{y}^{k+1}} = ht_{p,c}^{\bar{y}^{k+1}} &= ht_{p,c}^{\bar{y}^k} - f_p \\ &= ht_{p,c}^{\bar{y}^k} - cap_{pt} \\ &= ht_{p,c}^{\bar{y}^k} - \left(ht_{p,c}^{\bar{y}^k} - ht_{p,x_p^k}^{\bar{y}^k}\right) = ht_{p,x_p^k}^{\bar{y}^k} \end{aligned}$$

(d) If $x_p^{k+1} = c$ the property obviously holds. So we may assume that $x_p^{k+1} \neq c$. In that case, it will also be $x_p^k \neq c$ as well (due to property (b)). If p is connected to the source node s then the arc sp must be saturated i.e. $f_p = cap_{sp}$ or else it would hold $x_p^{k+1} = c$ according to the reassign rule. Using this fact as well as (30) and the definition of cap_{sp} in (25) the property then follows:

$$\begin{aligned} ht_{p,c}^{\bar{y}^{k+1}} &= ht_{p,c}^{\bar{y}^k} + f_p \\ &= ht_{p,c}^{\bar{y}^k} + cap_{sp} \\ &= ht_{p,c}^{\bar{y}^k} + \left(ht_{p,x_p^k}^{\bar{y}^k} - ht_{p,c}^{\bar{y}^k}\right) = ht_{p,x_p^k}^{\bar{y}^k} = ht_{p,x_p^{k+1}}^{\bar{y}^k} \end{aligned}$$

where the last equality is true due to the fact $x_p^k \neq c$ and property ***(a)***.
On the other hand, if p is connected to the sink t then:

$$\begin{aligned} ht_{p,c}^{\bar{y}^{k+1}} &= ht_{p,c}^{\bar{y}^k} - f_p && \text{by (30)} \\ &\geq ht_{p,c}^{\bar{y}^k} - cap_{pt} \\ &= ht_{p,c}^{\bar{y}^k} - \left(ht_{p,c}^{\bar{y}^k} - ht_{p,x_p^k}^{\bar{y}^k}\right) && \text{by (27)} \\ &= ht_{p,x_p^k}^{\bar{y}^k} = ht_{p,x_p^{k+1}}^{\bar{y}^k} \end{aligned}$$

where again the last equality is true due to the fact $x_p^k \neq c$ and property ***(a)***.

(e) If $x_q^k = c$ then $cap_{pq} = 0$ (due to (21)) while also $\bar{y}_{pq,c}^{k+1} = \bar{y}_{pq,c}^k$ (by property (b)) and so the property obviously holds. Therefore we may assume that $x_q^k \neq c$ which implies that $x_q^{k+1} \neq c$ as well (since $x_q^{k+1} = x_q^k$). Since p has been assigned the label c there must exist an unsaturated path $s \rightsquigarrow p$ from s to p. But then the forward arc pq as well as the backward arc qp of the path $s \rightsquigarrow p \rightarrow q$ must be saturated i.e.:

$$f_{pq} = cap_{pq} \text{ and } f_{qp} = 0 \tag{32}$$

or else that path would also be unsaturated (which would in turn imply that $x_q^{k+1} = c$ contrary to our assumption above). Due to (32) and (29) the property then follows.

(f) The first inequality follows directly from *(c)* and the definition of the APF function. Furthermore, if at least one change of label has taken place then according to the reassign rule there must be at least one unsaturated arc, say sp, between the source and some node p. This implies that $f_p < cap_{sp}$ and so by also using (30) and the definition of cap_{sp} in (25) it is then trivial to show that $ht_{p,x_p^{k+1}}^{\bar{y}^{k+1}} < ht_{p,x_p^k}^{\bar{y}^k}$. Due to this fact and by applying property (c) to all other vertices the desired strict inequality follows.

□

Property 3.1(a) simply expresses the fact that only dual variables related to the c-labels may change during a c-iteration while property 3.1(b) simply says that if label c is the active label during a c-iteration then its height is kept fixed during that iteration.

Note also that due to property 3.1(c) the height of a vertex (i.e. the height of its active label) decreases after each iteration of the algorithm. Furthermore, due to property 3.1(d), the new active label at the end of a c-iteration (i.e. the label assigned to a vertex by x^{k+1}) is always located "below" that vertex's label c (as was intended). Based on these observations the new label assigned to a vertex by x^{k+1} is always taken closer to having the minimum height at that vertex after the end of each iteration. We may therefore hope that by keep repeating this procedure we will finally make sure that (17) is satisfied after a sufficient number of iterations has elapsed.

Besides that, however, we also need to ensure that x^{k+1}, y^{k+1} satisfy conditions (18). To this end, the routine POSTEDIT_DUALS$(c, x^{k+1}, \bar{y}^{k+1})$ still needs to apply an additional correction to the resulting dual solution $\bar{y}^{k+1}$. In particular, the role of that routine will be to change $\bar{y}^{k+1}$ into y^{k+1} so that all of the active balance variables of solution y^{k+1} become nonnegative (this should come as no surprise since conditions (18) involve only sums of active balance variables). It turns out that in the case of algorithm PD1, only neighbors p, q with $x_p^{k+1} = x_q^{k+1} = c$ may have one of the active balance variables $\bar{y}_{pq,x_p^{k+1}}^{k+1}$, $\bar{y}_{qp,x_q^{k+1}}^{k+1}$ being negative during a c-iteration. In that case POSTEDIT_DUALS simply sets $y_{pq,c}^{k+1} = y_{qp,c}^{k+1} = 0$. No other differences between $\bar{y}^{k+1}, y^{k+1}$ exist. It is then obvious that for any neighboring vertices p, q the following equation holds: $y_{pq,x_p^{k+1}}^{k+1} + y_{qp,x_q^{k+1}}^{k+1} = \bar{y}_{pq,x_p^{k+1}}^{k+1} + \bar{y}_{qp,x_q^{k+1}}^{k+1}$. This means that all the loads are preserved during the transition from solution $\bar{y}^{k+1}$ to y^{k+1} (i.e. $load^{x^{k+1},y^{k+1}} = load^{x^{k+1},\bar{y}^{k+1}}$) which in turn implies (due to (15)) that POSTEDIT_DUALS does not alter the value of the APF function as well i.e. $APF^{x^{k+1},y^{k+1}} = APF^{x^{k+1},\bar{y}^{k+1}}$

This, seemingly minor, modification of the dual solution by POSTEDIT_DUALS plays, nevertheless, a very crucial role in the success of the PD1 algorithm. In particular, the considered modification along with property 3.1(e) will be absolutely necessary for ensuring that conditions (18) are satisfied by all dual solutions generated throughout PD1. This will become clear during the proof of the theorem 3.2 ahead, which is the main result of this section and proves that PD1 always leads to an f_{app}-approximate solution. The pseudocode for the PD1 algorithm is shown in Fig. 5.

Theorem 3.2. *The final primal and dual solutions generated by PD1 satisfy all conditions (16) - (19). Therefore, (as explained in section 3) these solutions are feasible and satisfy the relaxed complementary slackness conditions with* $f_1 = 1$, $f_2 = f_{app}$.

Proof: Due to the integrality assumption of the quantities $c_{p,a}, w_{pq}, d_{ab}$, both the initial dual solution as well as the capacities of the graph $G^{x^k,\bar{y}^k}$ are always of the form $\frac{n_0}{2}$ with $n_0 \in \mathbb{N}$. It can then be easily verified that any balance variable, and therefore the APF function too, can take values only of that form. So after every c-iteration any decrease of APF will always have magnitude $\geq 1/2$. Based on this observation and the fact that, as mentioned above, POSTEDIT_DUALS does not alter the value of the APF function, the algorithm termination (i.e. no change of label taking place for $|L|$ consecutive inner iterations) is guaranteed by the APF monotonicity property 3.1(f).

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**
$y^k_{pq,x_p^k} = -y^k_{qp,x_p^k} = w_{pq} d_{min}/2 = y^k_{qp,x_q^k} = -y^k_{pq,x_q^k}$
$y_p^k = \min_a ht^{y^k}_{p,a} \;\; \forall p \in V$ {***It imposes conditions (16)***}

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

UPDATE_DUALS_PRIMALS$(c, x^k, \bar{y}^k)$
$x^{k+1} = x^k$, $\bar{y}^{k+1} = \bar{y}^k$
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$(c, x^{k+1}, \bar{y}^{k+1})$
$y^{k+1} = \bar{y}^{k+1}$
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** $y^{k+1}_{pq,c} = y^{k+1}_{qp,c} = 0$
$y_p^{k+1} = \min_a ht^{y^{k+1}}_{p,a} \;\; \forall p \in V$ {***It imposes conditions (16)***}

Fig. 5: Pseudocode for the PD1 algorithm.

Feasibility conditions (16) are enforced by the definition of the PD1 algorithm (see Fig. 5). In addition, due to the specific assignment of capacities to interior edges (see (22)), the balance variables of edges pq, qp are not allowed to grow larger than $w_{pq} d_{min}/2$ and so constraints (19) are also enforced.

Furthermore, we can prove by induction that solutions x^k, y^k (for any k) satisfy slackness conditions (18) and have all of their active balance variables nonnegative i.e.

$$y^k_{pq,x_p^k} \geq 0 \tag{33}$$

These conditions are obviously true at initialization (by the definition of INIT_DUALS), so let us assume that they hold for x^k, y^k and let the current iteration be a c-iteration. We will then show that these conditions hold for x^{k+1}, y^{k+1} as well. To this end, we will consider 3 cases:

Case 1: let us first consider the case where $x_p^{k+1} = x_q^{k+1} = c$. Then $load_{pq}^{x^{k+1}, y^{k+1}} = 0$ (due to (14)) and so (18) obviously holds while (33) is guaranteed to be restored by the definition of POSTEDIT_DUALS.

Case 2: Next, let us examine the case where neither p nor q is assigned a new label (i.e. $x_p^{k+1} = x_p^k, x_q^{k+1} = x_q^k$). We can then show that:

$$y^{k+1}_{pq,x_p^{k+1}} = y^k_{pq,x_p^k} \quad y^{k+1}_{qp,x_q^{k+1}} = y^k_{qp,x_q^k} \tag{34}$$

and so both conditions (18), (33) follow directly from the induction hypothesis. Indeed, by applying either property 3.1(a) or 3.1(b), depending on whether $x_p^{k+1} = x_p^k = a \neq c$ or $x_p^{k+1} = x_p^k = c$, we conclude that $\bar{y}^{k+1}_{pq,x_p^{k+1}} = \bar{y}^k_{pq,x_p^k}$. In addition, $\bar{y}^k_{pq,x_p^k} = y^k_{pq,x_p^k} \geq 0$ with the equality being true due to the definition of the PREEDIT_DUALS function and the inequality following by the induction hypothesis. Combining the above relations we get:

$$\bar{y}^{k+1}_{pq,x_p^{k+1}} = y^k_{pq,x_p^k} \geq 0 \tag{35}$$

while with similar reasoning we can also show that:

$$\bar{y}^{k+1}_{qp,x_q^{k+1}} = y^k_{qp,x_q^k} \geq 0 \tag{36}$$

Therefore, both $\bar{y}^{k+1}_{pq,x_p^{k+1}}, \bar{y}^{k+1}_{qp,x_q^{k+1}}$ are nonnegative and so their values will not be altered by POSTEDIT_DUALS:

$$y^{k+1}_{pq,x_p^{k+1}} = \bar{y}^{k+1}_{pq,x_p^{k+1}} \quad y^{k+1}_{qp,x_q^{k+1}} = \bar{y}^{k+1}_{qp,x_q^{k+1}} \tag{37}$$

The above equation, in conjunction with (35), (36), implies that (34) holds true, as claimed.

Case 3: Finally, let us consider the only remaining case according to which only one of p, q (say p) is assigned a new label c i.e. $x_p^{k+1} = c \neq x_p^k$ while the other one (say q) keeps its current label i.e. $x_q^{k+1} = x_q^k = a$ with $a \neq c$. In this case due to property 3.1(e) and (22) it follows that $\bar{y}_{pq,c}^{k+1} = \bar{y}_{pq,c}^k + cap_{pq} = w_{pq} \cdot d_{min}/2$. In addition, it holds that $\bar{y}_{qp,a}^{k+1} = \bar{y}_{qp,a}^k = y_{qp,a}^k \geq 0$ where the 1^{st} equality is true due to $a \neq c$ and property 3.1(a), the 2^{nd} equality is true due to the definition of PREEDIT_DUALS and the inequality follows from the induction hypothesis. Since $a \neq c$ (or equivalently $x_q^{k+1} \neq c$), POSTEDIT_DUALS (by definition) will alter none of the active balance variables $\bar{y}_{pq,c}^{k+1}, \bar{y}_{qp,a}^{k+1}$ and so $y_{pq,c}^{k+1} = \bar{y}_{pq,c}^{k+1}, y_{qp,a}^{k+1} = \bar{y}_{qp,a}^{k+1}$. By combining all of the above equalities it is now trivial to verify that (18), (33) hold for x^{k+1}, y^{k+1} as well.

Finally, to conclude the proof of this theorem we need to show that the last primal-dual pair of solutions satisfies condition (17). According to the termination criterion of the PD1 algorithm, during its last $|L|$ inner iterations there should be no label change. Let c be any label and consider the c-iteration out of these last $|L|$ iterations. During that iteration it will hold that:

$$ht_{p,x_p^{k+1}}^{\bar{y}^{k+1}} \leq ht_{p,c}^{\bar{y}^{k+1}} \tag{38}$$

where the above inequality is true due to property 3.1(d). In addition, since no change of label takes place, we can apply the same reasoning as in case 2 above and show again that (37) holds for any neighboring vertices p, q. This implies that all active balance variables are kept constant during the transition from $\bar{y}^{k+1}$ into y^{k+1} which in turn implies that $y^{k+1} = \bar{y}^{k+1}$ since, by definition, POSTEDIT_DUALS cannot touch any non-active balance variables. Therefore, the heights of labels do not change during the transition from $\bar{y}^{k+1}$ into y^{k+1} and so, based on the previous inequality (38), it will also hold that:

$$ht_{p,x_p^{k+1}}^{y^{k+1}} \leq ht_{p,c}^{y^{k+1}} \tag{39}$$

Furthermore, the value of $ht_{p,x_p^{k+1}}^{y^{k+1}}$ is not altered during any of the next iterations. This is true because p keeps its current label (by the termination criterion) and so we may again show that (34) holds for all of the remaining iterations. Similarly, the value of $ht_{p,c}^{y^{k+1}}$ will not change hereafter since by assumption this is the last c-iteration i.e. the last time the balance variables of the c labels are updated. Therefore, inequality (39) will be maintained until the end of the algorithm. Since the same reasoning can be applied to any label c, condition (17) will finally hold true at the end of the last iteration. □

4 PD2 algorithm

The PD2 approximation algorithm can be applied only in the case of d_{ab} being a metric (the necessity of this assumption will become clear later in the section). In fact, PD2 represents not just one algorithm but instead a family of algorithms PD2$_\mu$ which is parameterized by a variable $\mu \in [\frac{1}{f_{app}}\ 1]$. The distinction between the various PD2$_\mu$ algorithms for different values of μ lies in the exact form of slackness conditions that they are trying to achieve. Specifically, the primal-dual solutions generated by PD2$_\mu$ will satisfy slackness conditions (11), (12) with $f_1 = \mu f_{app}$ and $f_2 = f_{app}$. The reason for requiring $\mu \geq \frac{1}{f_{app}}$ is that it can never be $f_1 < 1$.

4.1 Algorithm overview

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, the PD2$_\mu$ algorithm ensures that the (probably) infeasible dual solutions generated are "not too far away" from feasibility. This notion of "not too far away" practically means that if the infeasible dual 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" [14] in the linear programming literature.

More specifically, we will prove that the algorithm is generating a series of intermediate pairs consisting of primal-dual solutions with the following properties: all of them satisfy slackness condition (12) as an equality with $f_2 = \frac{1}{\mu}$:

$$x_p \neq x_q \Rightarrow load_{pq}^{x,y} = \mu w_{pq} d_{x_p x_q} \tag{40}$$

In addition, the last intermediate pair satisfies the exact (i.e. $f_1 = 1$) slackness condition (11) which, as explained in section 3, reduces to:

$$y_p = \min_a ht^y_{p,a} \tag{41}$$

$$ht^y_{p,x_p} = \min_a ht^y_{p,a} \tag{42}$$

However, the dual solution of this last pair is probably infeasible since although it satisfies dual constraints (7) (due to (41)), it can be guaranteed to satisfy only:

$$y_{pq,a} + y_{qp,b} \leq 2\mu w_{pq} d_{max} \quad \forall a, b \in L,\ (p,q) \in E \tag{43}$$

in place of the dual constraints (8).

Nevertheless the above conditions ensures that the last dual solution is not "too far away" from feasibility. This practically means that by replacing this last solution, say y, with $y^{fit} = \frac{y}{\mu f_{app}}$, it is then easy to show that the resulting dual solution y^{fit} satisfies the dual constraints (8) as well (and is therefore feasible):

$$y^{fit}_{pq,a} + y^{fit}_{qp,b} = \frac{y_{pq,a} + y_{qp,b}}{\mu f_{app}} \leq \frac{2\mu w_{pq} d_{max}}{\mu f_{app}} = \frac{2\mu w_{pq} d_{max}}{\mu 2 d_{max}/d_{min}} = w_{pq} d_{min} \leq w_{pq} d_{ab}$$

Furthermore, by making use of (40)-(42) it again takes only elementary algebra to show that the feasible primal-dual pair (x, y^{fit}) (where x is the last primal solution) satisfies the relaxed slackness conditions (11), (12) with $f_1 = \mu f_{app}$ and $f_2 = f_{app}$ and thus comprises an f_{app}-approximate solution. Indeed, this can be verified for the case of conditions (11) by making use of the next equalities:

$$\begin{aligned} y^{fit}_p = \frac{y_p}{\mu f_{app}} = \frac{ht^y_{p,x_p}}{\mu f_{app}} &= \frac{c_{p,x_p} + \sum_{q:q\sim p} y_{pq,x_p}}{\mu f_{app}} \\ &= \frac{c_{p,x_p}}{\mu f_{app}} + \sum_{q:q\sim p} \frac{y_{pq,x_p}}{\mu f_{app}} \\ &= \frac{c_{p,x_p}}{\mu f_{app}} + \sum_{q:q\sim p} y^{fit}_{pq,x_p} \end{aligned}$$

while for the case of conditions (12) one may proceed as follows:

$$y^{fit}_{pq,x_p} + y^{fit}_{qp,x_q} = \frac{y_{pq,x_p} + y_{qp,x_q}}{\mu f_{app}} = \frac{load^{x,y}_{pq}}{\mu f_{app}} = \frac{\mu w_{pq} d_{x_p x_q}}{\mu f_{app}} = \frac{w_{pq} d_{x_p x_q}}{f_{app}}$$

The generation of y^{fit} (given y) is exactly what the DUAL_FIT routine does.

The goal of the $PD2_\mu$ algorithm will therefore be to extract a primal-dual pair (x, y) satisfying conditions (40)-(43). It should be noted that (as in the PD1 case) imposing condition (41) is always a trivial thing to do. We now proceed to describe the bulk of the algorithm i.e. each of the main routines appearing during a c-iteration.

4.2 Update of the primal and dual variables

The update of the primal and dual variables inside UPDATE_DUALS_PRIMALS$(c, x^k, \bar{y}^k)$ takes place in exactly the same way as in the PD1 algorithm (see (29), (30) and the "reassign rule"). In addition, the construction of the graph $G_c^{x^k, \bar{y}^k}$ (as described in section 3.2) can be replicated here as well except for the assignment of capacities to a certain subset of interior edges. This subset will consist of all interior edges pq, qp (corresponding to edge (p,q) of the original graph G) 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 (22), we instead define:

$$cap_{pq} = \mu w_{pq}(d_{ac} + d_{cb} - d_{ab}) \tag{44}$$

$$cap_{qp} = 0 \tag{45}$$

In addition, in this case (i.e. when $x_p^k = a \neq c, x_q^k = b \neq c$) PREEDIT_DUALS changes $y_{qp,c}^k$ (and therefore its conjugate $y_{pq,c}^k$) into $\bar{y}_{qp,c}^k$ (and $\bar{y}_{pq,c}^k$) so as to ensure:

$$\bar{y}_{pq,a}^k + \bar{y}_{qp,c}^k = \mu w_{pq} d_{ac} \tag{46}$$

This is done without modifying $y_{pq,a}^k$ i.e. $\bar{y}_{pq,a}^k = y_{pq,a}^k$. Regarding the rest of the balance variables, PREEDIT_DUALS applies no changes to them, during the transition from y^k to $\bar{y}^k$, just like in the PD1 case. No further differences exist between the routines PREEDIT_DUALS, UPDATE_DUALS_PRIMALS of the PD1 algorithm and the corresponding routines in PD2$_\mu$.

Based on the above observations it is easy to see that PREEDIT_DUALS does not alter any of the active balance variables (indeed, according to its definition, PREEDIT_DUALS modifies a balance variable only if it is of the form $y_{pq,c}^k$ or $y_{qp,c}^k$ with $x_k^p \neq c$ and $x_k^q \neq c$). Therefore the values of all the loads are not altered during the transition from y^k to $\bar{y}^k$:

$$load_{pq}^{x^k,\bar{y}^k} = load_{pq}^{x^k,y^k} \tag{47}$$

In addition, the above equation (47) (along with (15)) implies that the APF function is also not modified:

$$APF^{x^k,\bar{y}^k} = APF^{x^k,y^k} \tag{48}$$

Furthermore, (44) explains why d_{ab} needs to be a metric (or else cap_{pq} would be negative).

The rationale behind the capacity assignments in (44), (45) and the specific definition of PREEDIT_DUALS is to ensure that in the case which only one of p, q is assigned the label c (by the new primal solution x^{k+1}), then condition (40) still remains valid. The basic tool to be used for the proof of this assertion will be property 3.1(e). All of the above can be seen in the following key lemma.

Lemma 4.1. *During a c-iteration, let p, q be two neighbors (i.e. $p \sim q$) with $x_p^k = a, x_q^k = b$ and assume that $x^k, \bar{y}^k$ satisfy condition (40) i.e. $load_{pq}^{x^k,\bar{y}^k} = \mu w_{pq} d_{x_p^k x_q^k}$.*

(a) *if $a \neq c, b \neq c$ then $\bar{y}_{pq,c}^{k+1} \leq \mu w_{pq} d_{cb} - \bar{y}_{qp,b}^k$ and $\bar{y}_{qp,c}^{k+1} \leq \mu w_{pq} d_{ac} - \bar{y}_{pq,a}^k$*

(b) *$x^{k+1}, \bar{y}^{k+1}$ satisfy condition (40) as well i.e. $load_{pq}^{x^{k+1},\bar{y}^{k+1}} = \mu w_{pq} d_{x_p^{k+1} x_q^{k+1}}$*

Proof:

(a) Since both of a, b are $\neq c$ then (44), (45), (46) hold. In addition, by the lemma hypothesis:

$$load_{pq}^{x^k,\bar{y}^k} = \bar{y}_{pq,a}^k + \bar{y}_{qp,b}^k = \mu w_{pq} d_{ab} \tag{49}$$

By property 3.1(e) the maximum value of $\bar{y}_{pq,c}^{k+1}$ will be: $\bar{y}_{pq,c}^k + cap_{pq} = \bar{y}_{pq,c}^k + \mu w_{pq}(d_{ac} + d_{cb} - d_{ab}) = \mu w_{pq} d_{cb} - \bar{y}_{qp,b}^k$ where the first equality is due to (44) and the last equality follows by substituting d_{ab}, d_{ac} from (49), (46). Likewise the maximum value of $\bar{y}_{qp,c}^{k+1}$ will be: $\bar{y}_{qp,c}^k + cap_{qp} = \bar{y}_{qp,c}^k = \mu w_{pq} d_{ac} - \bar{y}_{pq,a}^k$ where the first equality is due to (45) and the last equality follows from (46).

(b) If $x_p^{k+1} = x_q^{k+1}$ then $load_{pq}^{x^{k+1},\bar{y}^{k+1}} = 0$ and part (b) of the lemma obviously holds. Therefore we may hereafter assume that $x_p^{k+1} \neq x_q^{k+1}$. This assumption has as a result that not both of a, b can be equal to c or else it would hold $x_p^{k+1} = x_q^{k+1} = c$ due to property 3.1(b). On the other hand, if either one of a, b is equal to c (say $x_p^k = a = c, x_q^k = b \neq c$), this implies that $\bar{y}_{qp,b}^{k+1} = \bar{y}_{qp,b}^k$ (due to $b \neq c$ and property 3.1(a)) as well as $x_p^{k+1} = c$ and $\bar{y}_{pq,c}^{k+1} = \bar{y}_{pq,c}^k$ (due to $x_k^p = c$ and property 3.1(b)). Then necessarily $x_q^{k+1} = x_q^k = b$ (since we assume $x_q^{k+1} \neq x_p^{k+1} = c$ and by definition of x^{k+1} any vertex q is either assigned label c or else keeps its current label x_q^k). By combining all of the above equalities it follows that $load_{pq}^{x^{k+1},\bar{y}^{k+1}} = \bar{y}_{pq,c}^{k+1} + \bar{y}_{qp,b}^{k+1} = \bar{y}_{pq,c}^k + \bar{y}_{qp,b}^k = load_{pq}^{x^k,\bar{y}^k} = \mu w_{pq} d_{cb}$ (where the last equality is true due to the lemma hypothesis) and part (b) of the lemma therefore holds in this case.

We still need to consider only the case where both of a, b are different than c (i.e. $a \neq c, b \neq c$). Since we assume $x_p^{k+1} \neq x_q^{k+1}$ only one of p, q may be assigned label c by x^{k+1}. If label c is assigned to p but q keeps its current label b (i.e. $x_p^{k+1} = c, x_q^{k+1} = b$) then by property 3.1(e) $\bar{y}_{pq,c}^{k+1}$ attains its maximum value and so by part *(a)* $\bar{y}_{pq,c}^{k+1} = \mu w_{pq} d_{cb} - \bar{y}_{qp,b}^k$. In addition $\bar{y}_{qp,b}^{k+1} = \bar{y}_{qp,b}^k$ (due to $b \neq c$ and property 3.1(a)) and so $load_{pq}^{x^{k+1},\bar{y}^{k+1}} = \bar{y}_{pq,c}^{k+1} + \bar{y}_{qp,b}^{k+1} = \left(\mu w_{pq} d_{cb} - \bar{y}_{qp,b}^k\right) + \bar{y}_{qp,b}^k = \mu w_{pq} d_{cb}$. Likewise we can show that if label c is assigned to q (by x^{k+1}) but p keeps its current label a then $load_{pq}^{x^{k+1},\bar{y}^{k+1}} = \mu w_{pq} d_{ac}$. □

As will become clear during the proof of theorem 4.3 ahead, the above lemma plays a very significant role for the success of the PD2$_\mu$ algorithm. The second part of that lemma can lead, as we shall see, directly to the proof that conditions (40) remain valid throughout PD2$_\mu$. While the first part of the above lemma can be used in showing that the balance variables do not actually increase too much per iteration. This way we will be able later to show that the balance variables are always bounded above by the quantity $\mu w_{pq} d_{max}$, thus proving that conditions (43) are satisfied throughout PD2$_\mu$ as well.

However, for proving this last assertion the first part of lemma 4.1 does not suffice. We still need to apply an additional modification to the dual solution $\bar{y}^{k+1}$. This will be carried out by the POSTEDIT_DUALS routine, which also plays a very crucial role for the success of the PD2$_\mu$ algorithm. More specifically, as in the PD1 case, the role of that routine will be to change solution $\bar{y}^{k+1}$ into y^{k+1} so as to ensure that the active balance variables of the resulting solution y^{k+1} are nonnegative. The difference with the PD1 algorithm is that a greater number of active balance variables may turn out to be negative now. In addition, care must be taken (by POSTEDIT_DUALS) so that the value of the loads and the APF function are not altered during this transition from $\bar{y}^{k+1}$ to y^{k+1}.

To this end, POSTEDIT_DUALS is applying an operator RECTIFY(p,q) to any pair $(p,q) \in E$. This operator is defined as follows: let $x_p^{k+1} = a, x_q^{k+1} = b$ and let us also assume that at least one of the active balance variables $\bar{y}_{pq,a}^{k+1}, \bar{y}_{qp,b}^{k+1}$ is negative, say $\bar{y}_{qp,b}^{k+1} < 0$. Then if $a = b$ the operator RECTIFY(p,q) simply sets $y_{pq,a}^{k+1} = y_{qp,a}^{k+1} = 0$ while if $a \neq b$ it sets $y_{pq,a}^{k+1} = \bar{y}_{pq,a}^{k+1} + \bar{y}_{qp,b}^{k+1},\ y_{qp,b}^{k+1} = 0$.[6] During the transition from $\bar{y}^{k+1}$ to y^{k+1} no other balance variables are modified by the RECTIFY operator.

The resulting dual solution y^{k+1} then satisfies the following properties for any pair of neighboring vertices p, q:

Properties 4.2. ***(a)*** $load_{pq}^{x^{k+1},y^{k+1}} = load_{pq}^{x^{k+1},\bar{y}^{k+1}}$

(b) *The primal-dual solutions x^{k+1}, y^{k+1} satisfy conditions (40) i.e.* $load_{pq}^{x^{k+1},y^{k+1}} = \mu w_{pq} d_{x_p^{k+1} x_q^{k+1}}$

(c) $APF^{x^{k+1},y^{k+1}} = APF^{x^{k+1},\bar{y}^{k+1}}$

(d) $y_{pq,a}^{k+1} \leq |\bar{y}_{pq,a}^{k+1}|,\ \ y_{qp,a}^{k+1} \leq |\bar{y}_{qp,a}^{k+1}|\ \ \forall\, a \in L$

(e) $y_{pq,x_p^{k+1}}^{k+1} \geq 0,\ \ y_{qp,x_q^{k+1}}^{k+1} \geq 0$

Proof:

(a) This equality can be easily verified directly from the definition of the operator RECTIFY.

(b) This property follows easily by induction from lemma 4.1(b). Indeed, assuming that the pair x^k, y^k satisfies conditions (40) then the same thing applies to pair $x^k, \bar{y}^k$ as well due to (47). Therefore the hypothesis of lemma 4.1 holds and by use of 4.1(b) in that lemma the pair $x^{k+1}, \bar{y}^{k+1}$ also satisfies (40). By then applying property (a) the same conclusion can be drawn regarding the pair x^{k+1}, y^{k+1} and this completes the induction.

(c) It follows by combining property (a) above and equation (15).

(d) This can be trivially verified based on the definition of operator RECTIFY(p,q).

(e) Combining properties (a) and (b) we conclude that $load_{pq}^{x^{k+1},\bar{y}^{k+1}} \geq 0$. Using this fact it is then trivial to verify the property based on the definition of the RECTIFY operator.

□

It should be noted that property 4.2(e) above ensures that all active balance variables remain nonnegative throughout $PD2_\mu$ (as was intended) i.e. for any pair of neighboring vertices p, q it holds that:

$$y_{pq,x_p^k}^k \geq 0 \quad \forall\, k \tag{50}$$

The pseudocode for PD2$_\mu$ is shown in Fig. 6. We are now ready to prove the main theorem of this section.

Theorem 4.3. *The final primal-dual solutions generated by PD2$_\mu$ are feasible and satisfy the relaxed complementary slackness conditions with $f_1 = \mu f_{app}$ and $f_2 = f_{app}$.*

[6] Of course we also set their conjugate balance variables as: $y_{qp,a}^{k+1} = -y_{pq,a}^{k+1},\ \ y_{pq,b}^{k+1} = -y_{qp,b}^{k+1}$.

INIT_DUALS
$y^k = 0$
for each pair $(p,q) \in E$ with labels $x_p^k \neq x_q^k$ **do**
$y^k_{pq,x_p^k} = -y^k_{qp,x_p^k} = \mu w_{pq} d_{x_p^k x_q^k}/2 = y^k_{qp,x_q^k} = -y^k_{pq,x_q^k}$
$y_p^k = \min_a ht^{y^k}_{p,a} \ \forall p \in V$ {***It imposes conditions (41)***}

PREEDIT_DUALS(c, x^k, y^k)
$\bar{y}^k = y^k$
for each $(p,q) \in E$ with $x_p^k = a \neq c, x_q^k = b \neq c$ **do**
$\bar{y}^k_{qp,c} = \mu w_{pq} d_{ac} - \bar{y}^k_{pq,a}$; $\bar{y}^k_{pq,c} = -\bar{y}^k_{qp,c}$

POSTEDIT_DUALS$(c, x^{k+1}, \bar{y}^{k+1})$
for each pair $(p,q) \in E$ **do** RECTIFY(p,q)
$y_p^{k+1} = \min_a ht^{y^{k+1}}_{p,a} \ \forall p \in V$ {***It imposes conditions (41)***}

DUAL_FIT$(\ y^k\ )$**:** $y^{fit} = \frac{y^k}{\mu f_{app}}$

Fig. 6: Pseudocode for the PD2$_\mu$ algorithm. Only those routines differing from their counterparts in algorithm PD1 are shown.

Proof: Due to the integrality assumption of the quantities $c_{p,a}, w_{pq}, d_{ab}$, both the initial dual solution as well as the capacities of the graph $G^{x^k,\bar{y}^k}$ are always of the form $\frac{n_0}{2}$ with $n_0 \in \mathbb{N}$. It is then easy to verify that the APF function can take values only of the form $\frac{n_0}{2}$ with $n_0 \in \mathbb{N}$, so any decrease of APF will necessarily be of magnitude $\geq \frac{1}{2}$. The algorithm termination is then guaranteed by the APF monotonicity property 3.1(f) and the observation that neither PREEDIT_DUALS (due to (48)) nor POSTEDIT_DUALS (due to property 4.2(c)) alters the value of APF.

Also, conditions (41) are enforced by the definition of the $PD2_\mu$ algorithm (see Fig. 6) while conditions (40) follows directly from property 4.2(b).

In order to prove that conditions (43) hold as well it is enough to show by induction that $y^k_{pq,c}, y^k_{qp,c} \leq \mu w_{pq} d_{max} \ \forall c \in L$. This is obviously true at initialization so let's assume it holds for $y^k_{pq,c}, y^k_{qp,c}$. We will show that during a c-iteration this holds for $y^{k+1}_{pq,c}, y^{k+1}_{qp,c}$ as well. Due to property 4.2(d) it is enough to show $\bar{y}^{k+1}_{pq,c}, \bar{y}^{k+1}_{qp,c} \leq \mu w_{pq} d_{max}$. If either one of $x_p^k = a, x_q^k = b$ equals c then by property 3.1(b) $\bar{y}^{k+1}_{pq,c} = \bar{y}^k_{pq,c}$, $\bar{y}^{k+1}_{qp,c} = \bar{y}^k_{qp,c}$. Also, $\bar{y}^k_{pq,c} = y^k_{pq,c}$, $\bar{y}^k_{qp,c} = y^k_{qp,c}$ (since PREEDIT_DUALS may alter $y^k_{pq,c}$ only if $x_p^k \neq c$ and $x_q^k \neq c$). The assertion then follows from the induction hypothesis.

If both of a, b are $\neq c$ then by lemma 4.1(a) $\bar{y}^{k+1}_{pq,c} \leq \mu w_{pq} d_{cb} - \bar{y}^k_{qp,b}$ while also $\bar{y}^k_{qp,b} = y^k_{qp,b}$ (since $b \neq c$ and PREEDIT_DUALS, by definition, may alter only balance variables of the form $y^k_{pq,c}$ during a c-iteration). But $y^k_{qp,b} \geq 0$ since $x_q^k = b$ i.e. this is an active balance variable (see (50)). Therefore $\bar{y}^{k+1}_{pq,c} \leq \mu w_{pq} d_{cb} \leq \mu w_{pq} d_{max}$. Likewise, using again lemma 4.1(a), we can prove that $\bar{y}^{k+1}_{qp,c} \leq \mu w_{pq} d_{ac} \leq \mu w_{pq} d_{max}$ and the assertion follows.

Finally, we may show that the last primal-dual pair of solutions (say x, y) also satisfies conditions (42), by following the same reasoning that has been used in the proof of theorem 3.2 to show the satisfiability of the equivalent conditions (17). Therefore all conditions (40)-(43) hold true and so (as explained in section 4.1) the pair (x, y^{fit}) generated by DUAL_FIT will be feasible and will also satisfy all required slackness conditions, thus concluding the proof of the theorem. □

All PD2$_\mu$ algorithms with $\mu < 1$ (as well as PD1) are non-greedy algorithms. That means that neither the primal objective function (nor the dual objective function) are necessarily decreasing (increasing) during each iteration. Instead, it is the value of the APF function which is constantly decreasing but since APF is always kept close to the true primal function the decrease in APF is finally reflected to the values of the primal function as well. However, a notable thing happens when $\mu = 1$. In that case, due to (40), the load between any neighboring vertices p, q represents exactly their separation cost (i.e. $load^{x^k,y^k}_{pq} = w_{pq} d_{x_p^k x_q^k}$) and so it can be proved that APF coincides with the primal function (see lemma 4.4). In addition it can be shown that the resulting PD2$_{\mu=1}$ algorithm is actually equivalent to the c-expansion algorithm introduced by Boykov et.al. in [4] (which has been interpreted only as a greedy local search technique up to now). The proof of this fact comes in theorem 4.6 ahead. Before proceeding, we recall that a label assignment x' is called a c-expansion of (another label assignment) x^k, if it holds that either $x'_p = x_p^k$ or $x'_p = c$ for any $p \in V$ (i.e. only label c may be assigned as a new label by x').

Lemma 4.4. *Let x be a label assignment and y a dual solution (not necessarily feasible) satisfying the following conditions:*

$$load_{pq}^{x,y} \leq w_{pq} d_{x_p x_q} \; \forall (p,q) \in E \tag{51}$$

Let us also denote by $PRIMAL^x$ the value of the primal objective function at x. Under these assumptions it is always true that $APF^{x,y} \leq PRIMAL^x$ while if, in addition, conditions (51) *hold as equalities then $APF^{x,y} = PRIMAL^x$.*

Proof: Equation (15) and the assumptions of the lemma imply:

$$\begin{aligned} APF^{x,y} &= \sum_p c_{p,x_p} + \sum_{(p,q)\in E} load_{pq}^{x,y} \\ &\leq \sum_p c_{p,x_p} + \sum_{(p,q)\in E} w_{pq} d_{x_p x_q} = PRIMAL^x \end{aligned}$$

□

Lemma 4.5. *Let x^k, y^k be a primal dual pair of solutions at the start of a c-iteration of the PD2$_{\mu=1}$ algorithm. Let x' be any label assignment due to a c-expansion of x^k.*

(a) $APF^{x^{k+1},\bar{y}^{k+1}} \leq APF^{x',\bar{y}^{k+1}}$

(b) $APF^{x',\bar{y}^{k+1}} \leq PRIMAL^{x'}$

Proof:

(a) Since x' is a label assignment due to a c-expansion, this means that x' may either keep the current label x_p^k of a vertex p or assign label c to it. If $x'_p = x_p^k \neq c$ (i.e. x' keeps the current label of p) then by property 3.1(c) $ht_{p,x_p^{k+1}}^{\bar{y}^{k+1}} \leq ht_{p,x_p^k}^{\bar{y}^k} = ht_{p,x'_p}^{\bar{y}^k} = ht_{p,x'_p}^{\bar{y}^{k+1}}$ where the last equality is true due to $x'_p \neq c$ and property 3.1(a). On the other hand if $x'_p = c$ then by property 3.1(d) $ht_{p,x_p^{k+1}}^{\bar{y}^{k+1}} \leq ht_{p,c}^{\bar{y}^{k+1}} = ht_{p,x'_p}^{\bar{y}^{k+1}}$. So in any case $ht_{p,x_p^{k+1}}^{\bar{y}^{k+1}} \leq ht_{p,x'_p}^{\bar{y}^{k+1}}$ and therefore $APF^{x^{k+1},\bar{y}^{k+1}} \leq APF^{x',\bar{y}^{k+1}}$.

(b) Let us first recall that the following equation holds:

$$load_{pq}^{x^k,\bar{y}^k} = load_{pq}^{x^k,y^k} = w_{pq} d_{x_p^k x_q^k} \tag{52}$$

where the 1^{st} equality is due to the preservation of the load by PREEDIT_DUALS (see (47)) while the 2^{nd} one follows from the fact that all x^k, y^k generated by PD2$_{\mu=1}$ satisfy slackness conditions (40).

According to lemma 4.4, to prove the assertion it is enough to show that $x', \bar{y}^{k+1}$ satisfy (51). If $x'_p = x'_q$ this is obviously true since in that case $load_{pq}^{x',\bar{y}^{k+1}} = 0$ due to (14). If $x'_p = x_p^k, x'_q = x_q^k$ then by applying either property 3.1(a) or 3.1(b) (depending on whether $x_p^k = a \neq c$ or $x_p^k = c$) we can conclude that $\bar{y}_{pq,x_p^k}^{k+1} = \bar{y}_{pq,x_p^k}^k$. Similarly we can show that $\bar{y}_{qp,x_q^k}^{k+1} = \bar{y}_{qp,x_q^k}^k$ and so:

$$load_{pq}^{x^k,\bar{y}^{k+1}} = load_{pq}^{x^k,\bar{y}^k} \tag{53}$$

The property then follows since: $load_{pq}^{x',\bar{y}^{k+1}} = load_{pq}^{x^k,\bar{y}^{k+1}} = load_{pq}^{x^k,\bar{y}^k} = w_{pq} d_{x_p^k x_q^k} = w_{pq} d_{x'_p x'_q}$ where the the first and last equalities are true due to our assumption that $x'_p = x_p^k, x'_q = x_q^k$ while the 2^{nd} and 3^{rd} equalities are true due to equations (53) and (52) respectively. Also, if $x'_p \neq c, x'_q \neq c$ then necessarily $x'_p = x_p^k, x'_q = x_q^k$ (since x' is a c-expansion) and so we fall back into the previous case.

Therefore we still need to consider only the case where $x'_p \neq x'_q$, $(x'_p, x'_q) \neq (x_p^k, x_q^k)$ and one of x'_p, x'_q is equal to c. Assume that $x'_p = c$ and let us also set $x_p^k = a, x_q^k = b$. Based on all of the above assumptions and the fact that x' is a c-expansion of x^k, one may then easily prove that: $x'_q = b, a \neq c, b \neq c$. This together with (52) implies that the hypothesis of lemma 4.1(a) holds and so $\bar{y}_{pq,c}^{k+1} \leq w_{pq} d_{cb} - \bar{y}_{qp,b}^k$ while also $\bar{y}_{qp,b}^{k+1} = \bar{y}_{qp,b}^k$ (due to $b \neq c$ and property 3.1(a)). Therefore $load_{pq}^{x',\bar{y}^{k+1}} = \bar{y}_{pq,c}^{k+1} + \bar{y}_{qp,b}^{k+1} \leq \left(w_{pq} d_{cb} - \bar{y}_{qp,b}^k\right) + \bar{y}_{qp,b}^k = w_{pq} d_{cb}$ and the lemma follows. □

Theorem 4.6. *The label assignment x^{k+1} selected during a c-iteration of the PD2$_{\mu=1}$ algorithm, has the minimum primal cost among all label assignments that can result after a c-expansion of x^k.*

Proof: Assignment x^{k+1} is indeed a c-expansion since no c label may be replaced during a c-iteration (see property 3.1(b)). In addition $load_{pq}^{x^{k+1},\bar{y}^{k+1}} = load_{pq}^{x^{k+1},y^{k+1}} = w_{pq}d_{x_p^{k+1}x_q^{k+1}}$ due to properties 4.2(a) and 4.2(b). So, solutions $x^{k+1}, \bar{y}^{k+1}$ satisfy conditions (51) of lemma 4.4 as equalities and therefore $PRIMAL^{x^{k+1}} = APF^{x^{k+1},\bar{y}^{k+1}}$ by that lemma. Let now x' be any other label assignment due to a c-expansion of x^k. Combining the above equality with the (a) and (b) inequalities of lemma 4.5 we get: $PRIMAL^{x^{k+1}} = APF^{x^{k+1},\bar{y}^{k+1}} \leq APF^{x',\bar{y}^{k+1}} \leq PRIMAL^{x'}$ and the theorem therefore follows. □

5 PD3 algorithms: extending PD2 to the semimetric case

By modifications to the PD2$_\mu$ algorithm, three different variations (PD3$_a$, PD3$_b$, PD3$_c$) of that algorithm may result which are applicable even in the case of d_{ab} being a semimetric. Only two of these variations lead to approximations with guaranteed optimality properties. The proof of this fact makes again use of the dual fitting technique and follows along the same lines as those of proving the PD2$_\mu$ optimality properties. For this reason that proof will be omitted. For simplicity we will consider only the $\mu = 1$ case i.e. only the variations of the PD2$_{\mu=1}$ algorithm. An additional reason for that is because in that case, the resulting algorithms have nice, intuitive interpretations in the primal domain. Before proceeding, we are recalling a fact that proves to be very useful for providing these intuitive interpretations as well as for explaining the rationale lying behind the algorithms' definition: if exact slackness conditions hold and the approximation factor is therefore equal to 1 (i.e. the current solution is optimal) then the load between any two neighbors should represent exactly their separation cost.

The main difficulty of extending PD2$_{\mu=1}$ to the case of a semimetric relates to how the capacities of certain interior edges of $G_c^{x^k,\bar{y}^k}$ are defined during a c-iteration. In particular, these are all edges whose capacity is defined by equation (44). (This should come as no surprise since (44) has been the only place where the metric hypothesis has been used.) Equivalently, these are all interior edges pq, qp whose endpoints p, q are currently assigned labels $\neq c$ (i.e. $x_p^k = a \neq c,\ x_q^k = b \neq c$) and in addition the following inequality holds:

$$d_{ab} > d_{ac} + d_{cb} \tag{54}$$

Hereafter we will call any such pair a "conflicting pair" while the corresponding triplet of labels (a, b, c) will be called a "conflicting label-triplet". The only differences between a PD3 algorithm and PD2$_{\mu=1}$ originate from the way the former deals with any "conflicting pairs" that might occur. Also, as we shall see, these differences will concern only the definition of capacity of the above mentioned edges and/or certain modifications to the behavior of routines PREEDIT_DUALS and POSTEDIT_DUALS. The above observations imply that if the distance d_{ab} is a metric then the PD3 algorithms always coincide with PD2$_{\mu=1}$.

5.1 Algorithms PD3$_a$ and PD3$_b$

The 2 algorithms of the current section differ from the PD2$_{\mu=1}$ algorithm only when a "conflicting pair" is met during their execution. In all other cases, i.e. at non-conflicting pairs, these algorithms completely coincide with PD2$_{\mu=1}$, meaning that their routines PREEDIT_DUALS, UPDATE_DUALS_PRIMALS and POSTEDIT_DUALS work in exactly the same way with the corresponding routines from PD2$_{\mu=1}$.

Upon meeting a "conflicting pair" p, q (with $x_p^k = a \neq c, x_q^k = b \neq c$) during a c-iteration, both algorithms proceed then as follows in order to define the capacities of the edges pq, qp:

They first make use of a rule RESOLVE$([a, b], c)$ in order to do what we call "resolving the conflicting pair". RESOLVE$([a, b], c)$ (which can be freely specified by the user on a per-application basis) takes as input a "conflicting label-triplet" (a, b, c) (with $d_{ab} > d_{ac} + d_{cb}$) and selects one of the 2 pairs $(a, c),\ (c, b)$ while excluding the other one. The physical meaning of the resolve rule will become clear later in the section.

Then the value for one of the 2 capacities cap_{pq}, cap_{qp} is defined based on the output of this RESOLVE routine. In particular, if RESOLVE$([a, b], c)$ selects (a, c), then $cap_{qp} = 0$ and PREEDIT_DUALS sets $\bar{y}_{qp,c}^k$ so that $\bar{y}_{pq,a}^k + \bar{y}_{qp,c}^k = w_{pq}d_{ac}$. While if (c, b) is selected, then $cap_{pq} = 0$ and PREEDIT_DUALS assigns a value to $\bar{y}_{pq,c}^k$ so that $\bar{y}_{pq,c}^k + \bar{y}_{qp,b}^k = w_{pq}d_{cb}$. In both cases, no other change of balance variables takes place during PREEDIT_DUALS i.e. $\bar{y}_{pq,a}^k = y_{pq,a}^k, \bar{y}_{qp,b}^k = y_{qp,b}^k$. It should be noted that in the original PD2$_{\mu=1}$ algorithm it has been always the first case that was taking place (see (45), (46)). Let us now assume w.l.o.g. that RESOLVE$([a, b], c)$ has selected pair (a, c). Then the only thing that we still need to define,

before being able to apply UPDATE_DUALS_PRIMALS, is the capacity cap_{pq}. Two options will be considered, giving rise to 2 different algorithms:

PD3$_a$ algorithm: We choose to set $cap_{pq} = 0$ as well[7]. In this case, if the pair of labels a, c (the pair selected by RESOLVE) is assigned to p, q (i.e. $x_p^{k+1} = a, x_q^{k+1} = c$), then (as in lemma 4.1(b)) we may easily show that $load_{pq}^{x^{k+1},\bar{y}^{k+1}} = w_{pq}d_{ac}$. In addition, due to property 4.2(a), it is always the case that $load_{pq}^{x^{k+1},y^{k+1}} = load_{pq}^{x^{k+1},\bar{y}^{k+1}}$. Therefore, at the start of the next iteration the load between vertices p, q (i.e. $load_{pq}^{x^{k+1},y^{k+1}}$) will represent exactly the actual separation cost of p, q (i.e. $w_{pq}d_{ac}$) as it should in the optimal case.

However, if the pair of labels c, b (the pair excluded by RESOLVE) is assigned to p, q (i.e. $x_p^{k+1} = c, x_q^{k+1} = b$) then due to our choice of setting $cap_{pq} = 0$ it is again easy to show that $load_{pq}^{x^{k+1},\bar{y}^{k+1}} = w_{pq}(d_{ab} - d_{ac})$ which is $> w_{pq}d_{cb}$ since p, q is a "conflicting pair" (see (54)). So in this case, the load overestimates the actual separation cost. In order that this overestimation is not maintained during the next iteration of the algorithm, POSTEDIT_DUALS changes the active balance variables $\bar{y}_{pq,c}^{k+1}, \bar{y}_{qp,b}^{k+1}$[8] into $y_{pq,c}^{k+1}, y_{qp,b}^{k+1}$ so that the value of the sum $y_{pq,c}^{k+1} + y_{qp,b}^{k+1}$ decreases to $w_{pq}d_{cb}$ and the equality between $load_{pq}^{x^{k+1},y^{k+1}}$ and separation cost is therefore restored. The rest of the POSTEDIT_DUALS routine is exactly the same as in the PD2$_{\mu=1}$ algorithm.

For an intuitive understanding of what is really happening, one can think of the situation as follows: since $d_{ab} > d_{ac} + d_{cb}$ no matter how the capacities cap_{pq}, cap_{qp} are defined there will always exist one pair among (a, c), (c, b) which, if assigned to p, q by x^{k+1}, will lead to an overestimation of the separation cost in the sense that (for the next primal-dual pair x^{k+1}, y^{k+1}) the load between p, q will be greater than the actual separation cost of these vertices. RESOLVE selects which one of the two label pairs will cause an overestimated cost while at the same time the assignment of zero capacity to both edges $pq,\ qp$ by the algorithm ensures that this overestimation error will be as small as possible. In addition, POSTEDIT_DUALS is trying so that this overestimation is canceled before the beginning of the algorithm's next iteration.

One may also view this cost overestimation as an equivalent overestimation of the distance between labels. In the above case, for example, we saw that if labels c, b are assigned to p, q by x^{k+1} then instead of the actual separation cost $w_{pq}d_{cb}$ the resulting overestimated cost has been $w_{pq}\bar{d}_{cb}$ with $\bar{d}_{cb} = d_{ab} - d_{ac}$. This is equivalent to saying that the algorithm has assigned the distance $\bar{d}_{cb} > d_{cb}$ to labels c, b instead of their actual distance d_{cb}. In all other cases (i.e. when (a, b) or (a, c) are assigned to p, q by x^{k+1}) no cost overestimation takes place and so the distances assigned to the corresponding labels by the algorithm are equal to the actual distances i.e. $\bar{d}_{ab} = d_{ab}, \bar{d}_{ac} = d_{ac}$. Since $\bar{d}_{ac} + \bar{d}_{cb} = \bar{d}_{ab}$ one could then argue that the $PD3_a$ algorithm chose to overestimate the distance between labels c, b in order to restore the triangle inequality for the current label-triplet (a, b, c). Put otherwise, it is as if a "dynamic approximation" of the d semimetric by a varying "metric" $\bar{d}$ is taking place. The distance $\bar{d}_{cb}$ assigned to any pair of labels (c, b) by this metric is not kept fixed throughout PD3$_a$. Instead, the PD3$_a$ algorithm constantly adapts $\bar{d}$ according to the "conflicting label triplets" occurring during its execution, always trying to restore the triangle inequality for the current "conflicting label triplet" while introducing the least amount of overestimation error to the $\bar{d}$ semimetric at the same time. Furthermore, an advantage of this "metric approximation" to d is that it can be explicitly controlled through the RESOLVE scheme. That scheme is specified by the user and can therefore be chosen on a per-application basis.

It can be shown (using similar reasoning with that in theorem 4.3) that the intermediate primal-dual solutions generated by both algorithms PD3$_a$ and PD2$_{\mu=1}$ satisfy exactly the same conditions and therefore it can be guaranteed that PD3$_a$ always leads to an f_{app}-approximate solution as well.

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 (i.e. the pair excluded by* RESOLVE*) to the vertices p, q respectively.* To prove this, it is enough to recall the "reassign rule" and also observe that there will always be a possibility of increasing the flow through directed pq without that edge ever becoming saturated (since $cap_{pq} = +\infty$). Indeed, if label c is assigned to p by x^{k+1} (which, according to the "reassign rule", means that there is an unsaturated path $s \rightsquigarrow p$) then label b can never be assigned to q since in that case the path $s \rightsquigarrow p \rightarrow q$ would also be unsaturated and so, by the "reassign rule" again, q would have to be assigned label c as well. Put otherwise, if the labels excluded by RESOLVE are assigned to p, q an infinite overestimation of the separation cost takes place and so we implicitly prevent those labels from being assigned to the "conflicting pair".

[7] instead of using the capacity definition (44) which would now be invalid.

[8] and also their conjugate variables

Unfortunately the price we pay for this infinite overestimation 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 primal-dual solutions satisfy a "not too far away from feasibility" condition like (43). This in turn implies that no dual-fitting technique can be applied in this case.

However, the PD3$_b$ algorithm has a nice interpretation in the primal domain. This can be seen in the following theorem which is analogous to theorem 4.6 and can be proved using similar reasoning with the proof of that theorem.

Theorem 5.1. *The label assignment x^{k+1} selected during a c-iteration of the PD3$_b$ algorithm, has the minimum primal cost among all label assignments that may result after a c-expansion of x^k, disregarding the ones that assign labels excluded by* RESOLVE *to "conflicting pairs".*

This theorem designates the price we pay for d_{ab} being a semimetric: in the metric case we can choose the best assignment among all c-expansion moves while in the semimetric case we are only able to choose the best one among a certain subset of these c-expansion moves. Despite this fact, the considered subset contains a very large number of c-expansion moves which makes the algorithm a good candidate as a local minimizer. Another interesting thing to note is that the the choice of the c-expansion moves to be included in this subset can be controlled by the RESOLVE scheme that will be selected. This scheme can be application specific and each time could reflect a priori knowledge about the considered problem (excluding for example configurations which are a priori highly unlikely to appear).

5.2 Algorithm PD3$_c$

Contrary to the previous two algorithms PD3$_a$ and PD3$_b$, the algorithm of this section may differ with PD2$_{\mu=1}$ even at "non-conflicting pairs". In addition, PD3$_c$ does not make use of any RESOLVE scheme at all. Instead, it applies the following modifications to the PD2$_{\mu=1}$ algorithm. It first adjusts (if needed) the dual solution y^k so that the following inequality holds for any neighboring vertices p, q:

$$load_{pq}^{x^k,y^k} \leq w_{pq}(d_{ac} + d_{cb}) \tag{55}$$

If this is not the case (i.e. if $load_{pq}^{x^k,y^k} = y_{pq,a}^k + y_{qp,b}^k$ is greater than $w_{pq}(d_{ac} + d_{cb})$) then in order to restore (55) one can simply decrease $y_{pq,a}^k, y_{qp,b}^k$ so that $load_{pq}^{x^k,y^k} = w_{pq}(d_{ac} + d_{cb})$. After this initial adjustment of the dual solution, the algorithm then continues in exactly the same way as the PD2$_{\mu=1}$ algorithm with the only difference being that instead of using equation (44) to define capacity cap_{pq}, that capacity is set by the algorithm as follows:

$$cap_{pq} = w_{pq}(d_{ac} + d_{cb} - \bar{d}_{ab})$$

In other words, the algorithm has simply replaced the distance d_{ab} in equation (44) with a new distance $\bar{d}_{ab}$ which is defined as:

$$\bar{d}_{ab} = \frac{load_{pq}^{x^k,y^k}}{w_{pq}} \tag{56}$$

Due to (55) it is obvious that $\bar{d}_{ab} \leq d_{ac} + d_{cb}$ and so the above definition of capacity cap_{pq} is valid i.e. $cap_{pq} \geq 0$. No other differences between PD3$_c$ and PD2$_{\mu=1}$ exist. Based on this definition of PD3$_c$, it can then be shown that if d_{ab} is a metric then the distances d_{ab}, $\bar{d}_{ab}$ coincide (i.e. $d_{ab} = \bar{d}_{ab}$) while, in addition, condition (55) always holds true and so no initial adjustment of y^k is needed. These two facts imply that PD3$_c$ is completely equivalent to PD2$_{\mu=1}$ in this case.

It is now interesting to examine what happens if p, q is a "conflicting pair" (with $x_p^k = a \neq c, x_q^k = b \neq c$). In that case it holds that $d_{ac} + d_{cb} < d_{ab}$ and so by combining this inequality with (56) and (55) one can conclude that $\bar{d}_{ab} < d_{ab}$ as follows:

$$\bar{d}_{ab} = \frac{load_{pq}^{x^k,y^k}}{w_{pq}} \leq \frac{w_{pq}(d_{ac} + d_{cb})}{w_{pq}} < \frac{w_{pq}d_{ab}}{w_{pq}} = d_{ab}$$

Furthermore, it can be proved that if either the pair (a, c) or (c, b) is assigned to p, q by x^{k+1} during the current iteration, then the resulting load (i.e. $load_{pq}^{x^{k+1},y^{k+1}}$) will represent exactly the actual separation cost of p, q (i.e. either $w_{pq}d_{ac}$ or $w_{pq}d_{cb}$). However, if none of p, q is assigned a new label by x^{k+1} (i.e. they both retain their current labels a, b) then it can also be shown that $load_{pq}^{x^{k+1},y^{k+1}} = w_{pq}\bar{d}_{ab}$ and so the load constitutes an underestimation of the actual separation cost $w_{pq}d_{ab}$ since $\bar{d}_{ab} < d_{ab}$ as was shown above.

Based on these observations, one can see that the PD3$_c$ algorithm works in a complementary way to the PD3$_a$ algorithm: in order to restore the triangle inequality for the "conflicting label-triplet" (a, b, c), instead of overestimating the distance between either the labels (a, c) or (c, b) (like PD3$_a$ did), it chooses to underestimate the distance between (a, b). Again one may view this as a "dynamic approximation" of the d semimetric by a constantly adapting metric $\bar{d}$, however this time we set $\bar{d}_{ab} = load_{pq}^{x^k,y^k}/w_{pq} < d_{ab}$, $\bar{d}_{ac} = d_{ac}$ and $\bar{d}_{cb} = d_{cb}$.

It can be shown that the intermediate primal-dual solutions generated by both algorithms PD3$_c$ and PD2$_{\mu=1}$ satisfy exactly the same conditions except for condition (40). In place of that condition, the intermediate solutions of PD3$_c$ can be shown to satisfy:

$$load_{pq}^{x^k,y^k} \geq w_{pq}\hat{d}_{x_p^k x_q^k} \tag{57}$$

where $\hat{d}_{ab} = \min_{c \in L}\big(d_{ac} + d_{cb}\big)$. By applying then the same (as in PD2$_{\mu=1}$) dual fitting factor to the last dual solution of PD3$_c$, one can easily prove that PD3$_c$ leads to an f'_{app}-approximate solution where:

$$f'_{app} = f_{app} \cdot c_0 \quad \text{with} \quad c_0 = \max_{a \neq b} \frac{d_{ab}}{\hat{d}_{ab}} \tag{58}$$

Since it is always true that $\hat{d}_{ab} \leq d_{ab}$ (due to $d_{ab} = d_{ab} + d_{bb}$), this has as a result that $c_0 \geq 1$ with equality holding only if d_{ab} is a metric. Therefore it will always hold that $f'_{app} \geq f_{app}$ and so PD3$_c$ cannot guarantee a better approximation factor.

It should be noted at this point that in the case of d_{ab} being a semimetric the choice (between PD3$_a$, PD3$_b$, PD3$_c$) of the algorithm that will be applied can be decided on an iteration by iteration basis.

6 Algorithmic properties of the presented primal-dual algorithms

As implied by the Primal-Dual Principle in section 2, each ratio $r = c^T x / b^T y$ (where x, y is any pair of integral-primal, dual feasible solutions) provides a suboptimality bound for the cost of the current primal solution, in the sense that x is guaranteed to be an r-approximation to the optimal integral solution. This property (which is a very significant advantage of any primal-dual algorithm) leads to an important consequence that proves to be very useful in practice:
By considering the sequence of primal-dual solutions $\{x^k, y^k\}_{k=1}^t$ generated throughout the primal-dual schema, a series of suboptimality bounds $\{r^k = c^T x^k / b^T y^k\}_{k=1}^t$ can be obtained. The minimum of these bounds is much tighter (i.e. much closer to 1) than the corresponding worst-case bound and so this allows one to have a much clearer view about the goodness of a solution.

This has been verified experimentally by applying the presented algorithms to the stereo matching problem. In this case, the labels represent image pixel disparities and they can be chosen from a set $L = \{0, 1, \ldots, K\}$ of discretized disparities where K denotes the maximum allowed disparity. The vertices of the graph G are the image pixels and the edges of G connect each pixel to its 4 immediate neighbors in the image. During our tests, the label cost for assigning disparity a to the image pixel p has been set equal to:

$$c_{p,a} = |I_{right}(p + a) - I_{left}(p)| \tag{59}$$

where I_{left}, I_{right} represent the intensities of the left and right images respectively.

We have applied our algorithms to the well-known Tsukuba stereo data set [16] setting the maximum disparity value equal to $K = 14$ based on the provided ground truth data. A sample from the results produced when using the algorithms PD2$_{\mu=1}$ (which is also equivalent to the c-expansion algorithm) and PD1 are shown in Fig. 7. It should be noted that no attempt has been made to model occlusions during the stereo matching procedure while, in addition, all edge weights w_{pq} have been set equal to each other instead of properly adjusting their values based on the intensity differences $|I_{left}(p) - I_{left}(q)|$ (something which would improve the quality of the estimated disparity since, in practice, disparity discontinuities usually coincide with intensity edges). The reason for that as well as for using the very simple label cost presented in (59) is because our main goal was not to produce the best possible disparity estimation but only to test the tightness of the suboptimality bounds that are provided by the considered algorithms i.e. to test the ability of these algorithms to efficiently minimize the objective function of a Metric Labeling problem.

To this end, 3 different distances d_{ab} have been used during our experiments. These are the Potts distance d^p (a metric),

(a) (b) (c) (d)

Fig. 7: (a) The left and **(b)** right images for one stereo pair from the Tsukuba data set. **(c)** The disparity estimated by the PD1 algorithm. **(d)** The corresponding disparity of the PD2$_{\mu=1}$ algorithm. This algorithm was shown to be equivalent to the expansion Graph Cuts algorithm. In both of the above cases the Potts metric has been used as the distance between labels. Since this distance is a metric the algorithms PD3$_a$, PD3$_b$, PD3$_c$ coincide with PD2$_{\mu=1}$ in this case and therefore produce the same result as that in figure (d).

the truncated linear distance d^l (also a metric) and the truncated quadratic distance d^q (a semimetric), defined as follows:

$$d^p_{ab} = 1 \qquad \forall a \neq b \tag{60}$$

$$d^l_{ab} = \min(M, |a-b|) \qquad \forall a, b \tag{61}$$

$$d^q_{ab} = \min(M, |a-b|^2) \qquad \forall a, b \tag{62}$$

In the above equations the constant M denotes the maximum allowed distance.

Each experiment consisted of selecting an approximation algorithm and a distance function and then using them for computing disparities for each one of the Tsukuba stereo pairs. The average values of the obtained suboptimality bounds are displayed in table 1. The columns f_{app}^{PD1}, $f_{app}^{PD2_{\mu=1}}$, $f_{app}^{PD3_a}$, $f_{app}^{PD3_b}$, $f_{app}^{PD3_c}$ of that table list these averages for the algorithms $PD1, PD2_{\mu=1}, PD3_a, PD3_b$ and $PD3_c$ respectively. In addition, the last column lists the value of the corresponding approximation factor f_{app} which, as already proved, constitutes a worst-case suboptimality bound for most of the above algorithms. By observing table 1 one can conclude that the per-instance suboptimality bounds are much tighter (i.e. much closer to 1) than the worst-case bounds predicted in theory. This holds for all cases i.e. for all combinations of algorithms and distances and so this fact indicates that the presented algorithms are always able to extract a nearly optimal solution (with this being true even for the more difficult case where d_{ab} is merely a semimetric). *Since the c-expansion algorithm has proved to be equivalent to one of the presented algorithms this gives yet another explanation for the great success that graph-cut techniques exhibit in practice.*

Besides the tightness of the per instance suboptimality bounds, another important issue is their accuracy i.e. how well these bounds describe the true suboptimality of the generated solutions. To investigate this issue we modified our experiments in the following way: 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 easily computed using dynamic programming. This in turn implies that we are able to compute the true suboptimality of a solution. If this fact is applied to our case, it leads to the construction of table 2. Its columns f_{true}^{PD1}, $f_{true}^{PD2_{\mu=1}}$, $f_{true}^{PD3_a}$, $f_{true}^{PD3_b}$, $f_{true}^{PD3_c}$ contain the true average suboptimality of the solutions of $PD1, PD2_{\mu=1}, PD3_a, PD3_b$ and $PD3_c$ respectively where the average is taken over all image scanlines. Furthermore, by examining table 2 one can see that the true suboptimality of a solution is always close to the corresponding suboptimality bound. This implies that these bounds are relatively accurate and therefore reliable for judging the goodness of an algorithms's solution.

More generally speaking, any primal-dual approximation algorithm places an upper bound on the so-called integrality gap $IGAP_{primal}$ [14] of the LP relaxation of the primal problem. In particular, an f-approximation primal-dual algorithm places the following bound on the integrality gap:

$$IGAP_{primal} \leq f$$

where the integrality gap $IGAP_{primal}$ is defined as the supremum (over all instances of the problem) of the ratio of the optimal integral and fractional solutions and is always ≥ 1.

Obviously, the integrality gap is the best approximation factor we can hope to prove. In fact, for special cases of the Metric Labeling Problem it can be shown that the presented primal-dual algorithms yield approximation factors that are essentially equal to the resulting integrality gap i.e. these are the best possible approximation factors.

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 $(M = 5)$	1.0226	1.0104	1.0104	1.0104	1.0104	10
Trunc. quad. $(M = 5)$	1.0280	-	1.0143	1.0158	1.0183	10

Table 1: The average suboptimality bounds obtained when using various combinations of primal-dual algorithms and distances to compute disparities for the Tsukuba stereo data set. As expected these are much closer to 1 than the corresponding worst-case suboptimaly bounds listed in the last column f_{app} of the table. This in turn implies that the generated solutions are much closer to the optimal solution. It should be noted that when using as distance either the Potts or the Truncated Linear metric the suboptimality bounds of the algorithms PD2$_{\mu=1}$, PD3$_a$, PD3$_b$ and PD3$_c$ always coincide (as the algorithms themselves coincide since the distance d_{ab} is a metric in both cases). Furthermore, $PD2_{\mu=1}$ cannot be applied when the truncated quadratic distance is being used since that distance is not a metric.

Distance	$\mathbf{f_{app}^{PD1}}$	$\mathbf{f_{true}^{PD1}}$	$\mathbf{f_{app}^{PD2_{\mu=1}}}$	$\mathbf{f_{true}^{PD2_{\mu=1}}}$	$\mathbf{f_{app}^{PD3_a}}$	$\mathbf{f_{true}^{PD3_a}}$	$\mathbf{f_{app}^{PD3_b}}$	$\mathbf{f_{true}^{PD3_b}}$	$\mathbf{f_{app}^{PD3_c}}$	$\mathbf{f_{true}^{PD3_c}}$
Potts	1.0098	1.0036	1.0066	1.0004	1.0066	1.0004	1.0066	1.0004	1.0066	1.0004
Trunc. Linear	1.0202	1.0107	1.0115	1.0021	1.0115	1.0021	1.0115	1.0021	1.0115	1.0021
Trunc. quad.	1.0255	1.0130	-	-	1.0135	1.0011	1.0144	1.0020	1.0160	1.0036

Table 2: The average suboptimality bounds (columns 2-4-6-8-10) obtained when applying our stereo matching algorithms to one scanline at a time (instead of the whole image). In this case, we are also able to compute the true average suboptimality (columns 3-5-7-9-11) of the generated solutions using dynamic programming. As can be seen by inspecting the table the suboptimality bounds always approximate the true suboptimality relatively well, meaning that they can be safely used as a measure for judging the goodness of a generated solution.

Such a special case is the Generalized Potts model [17]. *(This explains in yet another way why Graph-Cut techniques are so good in dealing with problems that are related to minimizing the Potts energy).* The Generalized Potts model can be derived simply by using the Potts metric (defined in (60)) as the distance d_{ab} between labels. In this case the provided approximation factor can be easily seen to be $f_{app} = 2$ and so $IGAP_{potts} \leq f_{app} = 2$. This is essentially the best possible bound for the integrality gap of the Potts model since one may easily construct a sequence of instances $\{I_k\}_{k=3}^{\infty}$ of the Metric Labeling problem where each I_k makes use of the Potts metric while, in addition, the sequence of the integrality gaps of all I_k converges to two [1]. For example, one could consider as I_k the instance where G is a complete graph on k nodes $\{p_1, p_2, \ldots, p_k\}$, the edge weights $w_{p_i q_j}$ are all equal to 1 and the label set L consists of k labels $\{a_1, a_2, \ldots, a_k\}$ with all label costs equal to zero except for the costs $\{c_{p_i,a_i}\}_{i=1}^{k}$ which are set equal to infinity. It is then not difficult for one to check that the resulting sequence of integrality gaps indeed converges to 2.

References

[1] J. Kleinberg and E. Tardos, "Approximation algorithms for classification problems with pairwise relaionships: metric labeling and markov random fields," *Journal of the ACM*, vol. 49, pp. 616–630, 2002.

[2] C. Chekuri, S. Khanna, J. Naor, and L. Zosin, "Approximation algorithms for the metric labeling problem via a new linear programming formulation," in *12^{th} Annual ACM-SIAM Symposium on Discrete Algorithms*, 2001, pp. 109–118.

[3] A. Archer, J. Fakcharoenphol, C. Harrelson, R. Krauthgamer, K. Talvar, and E. Tardos, "Approximate classification via earthmover metrics," in *Proceedings of the 15^{th} Annual ACM-SIAM Symposium on Discrete Algorithms*, 2004.

[4] Y. Boykov, O. Veksler, and R. Zabih, "Fast approximate energy minimization via graph cuts," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, no. 11, pp. 1222–1239, Nov. 2001.

[5] O. Veksler, "Efficient graph-based energy minimization methods in computer vision," Ph.D. dissertation, Department of Computer Science, Cornell University, 1999.

[6] S. Roy and I. Cox, "A maximum-flow formulation of the n-camera stereo correspondence problem," in *Proceedings of the International Conference on Computer Vision*, 1998, pp. 492–499.

[7] A. Gupta and E. Tardos, "Constant factor approximation algorithms for a class of classification problems," in *Proceedings of the 32^{nd} Annual ACM Symposium on the theory of Computing*, 2000, pp. 652–658.

[8] H. Ishikawa and D. Geiger, "Segmentation by grouping junctions," in *IEEE conference on Computer Vision and Pattern Recognition*, 1998.

[9] Y. Boykov and M.-P. Jolly, "Interactive graph cuts for optimal boundary and region segmentation of objects in n-d images." in *IEEE International Conference on Computer Vision*, 2001, pp. 105–112.

[10] Y. Boykov and V. Kolmogorov, "Computing geodesics and minimal surfaces via graph cuts." in *IEEE International Conference on Computer Vision*, 2003, pp. 26–33.

[11] V. Kolmogorov and R. Zabih, "Multi-camera scene reconstruction via graph cuts." in *European Conference on Computer Vision*, 2002, pp. 82–96.

[12] R. Zabih and V. Kolmogorov, "Spatially coherent clustering using graph cuts." in *IEEE conference on Computer Vision and Pattern Recognition*, 2004, pp. 437–444.

[13] V. Kolmogorov and R. Zabih, "What energy functions can be minimized via graph cuts?" in *European Conference on Computer Vision*, 2002, pp. 65–81.

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

[15] A. Gibbons, *Algorithmic Graph Theory*. Cambridge University Press, 1985.

[16] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *International Journal of Computer Vision*, vol. 47, no. 1/2/3, pp. 7–42, April-June 2002.

[17] Y. Boykov, O. Veksler, and R. Zabih, "Markov random fields with efficient approximations," in *IEEE conference on Computer Vision and Pattern Recognition*, 1998.