Performance vs Computational Efficiency for Optimizing Single and Dynamic MRFs: Setting the State of the Art with Primal Dual Strategies

Nikos Komodakis, Georgios Tziritas and Nikos Paragios

Abstract

In this paper we introduce a novel method to address minimization of static and dynamic MRFs. Our approach is based on principles from linear programming and, in particular, on primal dual strategies. It generalizes prior state-of-the-art methods such as α-expansion, while it can also be used for efficiently minimizing NP-hard problems with complex pair-wise potential functions. Furthermore, it offers a substantial speedup - of a magnitude ten - over existing techniques, due to the fact that it exploits information coming not only from the original MRF problem, but also from a dual one. The proposed technique consists of recovering pair of solutions for the primal and the dual such that the gap between them is minimized. Therefore, it can also boost performance of dynamic MRFs, where one should expect that the new new pair of primal-dual solutions is closed to the previous one. Promising results in a number of applications, and theoretical, as well as numerical comparisons with the state of the art demonstrate the extreme potentials of this approach[1].

I. Introduction

A wide variety of tasks in computer vision and pattern recognition can be formulated as discrete labeling problems. Furthermore, many of these problems can be very elegantly expressed in the language of discrete Markov Random Fields (MRFs) and it is exactly for this reason that MRF optimization is considered to be a task of fundamental importance, which has attracted a significant amount of research in computer vision over the last years.

N. Komodakis is with the MAS Laboratory at Ecole Centrale de Paris. Address: Grande Voie des Vignes 92 295 Chatenay-Malabry, FRANCE. Phone/Fax: : +33 (0)1 4113 1785/35. E-mail: komod@csd.uoc.gr

N. Paragios is with the MAS Laboratory at Ecole Centrale de Paris

G. Tziritas is with the University of Crete, Computer Science Department

[1] A version of our MRF optimization code will become publicly available from: http://www.csd.uoc.gr/~komod/FastPD

In very simple terms, the MRF optimization problem can be stated as follows: we are given a discrete set of objects $\mathcal{V}$, all of which are vertices in a graph $\mathcal{G}$. The edges of this graph (denoted by $\mathcal{E}$) encode the objects' relationships. We are also given as input a discrete set of labels $\mathcal{L}$. We must then assign one label from $\mathcal{L}$ to each object in $\mathcal{V}$. However, each time we choose to assign a label, say, x_p to an object p, we are forced to pay a price according to the so called *singleton* potential function $\mathfrak{c}_p(x_p)$, while each time we choose to assign a pair of labels, say, x_p and x_q to two interrelated objects p and q (*i.e*, two objects that are connected to each other by an edge in the graph $\mathcal{G}$), we are also forced to pay another price, which is now determined by the so called *pairwise* potential function $d(x_p, x_q)$ (both the singleton and pairwise potential functions are problem specific and are thus assumed to be provided as input)[2].

Our goal is then to choose a labeling which will allow us to pay the smallest total price. In other words, based on what we have mentioned above, we want to choose a labeling that minimizes the sum of all the MRF potentials, or equivalently the MRF energy. This amounts to solving the following optimization problem:

$$\arg\min_{\{x_p\}} \sum_{p \in \mathcal{V}} \mathfrak{c}_p(x_p) + \sum_{(p,q) \in \mathcal{E}} w_{pq} d(x_p, x_q). \tag{1}$$

Note that, in the above formula, we have also included a weight w_{pq} per edge, which can be used for scaling (either up or down) the pairwise potential function associated with that edge.

Despite this seemingly simple formulation, MRFs have great descriptive power and offer an extremely flexible framework, which is capable of modeling a wide range of problems. For this reason, they have been extremely popular in computer vision up to now. For instance, they have been used in problems such as image segmentation [31], 3D reconstruction [27], optical flow estimation [43], texture synthesis [32], image completion [33], image denoising [34], object recognition [35], visual correspondence [29], just to mention a few of the applications of discrete MRFs in computer vision. However, it should be noted that they have been used with great success in many other disciplines as well, including medical imaging [42], [25], [26], computer graphics, machine learning and artificial intelligence [37], digital communications, error-correcting coding theory, and statistical physics [19].

Hence, given the great popularity of MRFs, it becomes immediately obvious that MRF optimization is a task of fundamental importance with far reaching applications in many areas. Yet, this task is highly non-trivial since almost all interesting MRFs exhibit a highly non-convex energy function (with many local

[2] Throughout this document we will assume that all edges of the MRF share a common pairwise potential function $d(\cdot, \cdot)$, but note that everything that we will mention here also applies to the general case where there exists a unique pairwise potential function $d_{pq}(\cdot, \cdot)$ for each edge pq.

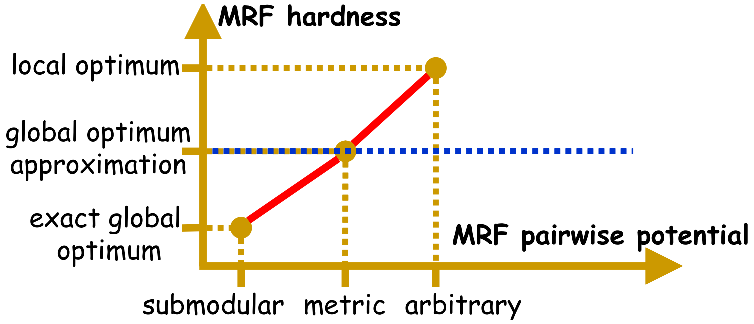


Fig. 1: The difficulty of optimizing an MRF (vertical axis) depends crucially on the type of the its pairwise potential function (horizontal axis). Here, we show how this difficulty varies for a few typical pairwise potential functions. Ideally, we would like to be able to handle NP-hard MRFs whose pairwise potential function is as general as possible (this means going as far as possible towards the right direction in the horizontal axis), but still without losing the ability to provide approximately optimal solutions (this means that we would like to remain at a height as low as possible in the vertical axis, *i.e*, below the blue dashed line). Furthermore, we would like to be able to do that efficiently, *i.e*, as fast as possible.

minima), which is thus NP-hard to optimize. We should, of course, note that not all MRFs encountered in practice are equally hard to optimize. In fact, it turns out that the difficulty of any given MRF optimization problem depends not so much on the form of its singleton potential functions (which can be arbitrary), but mostly on the form of its pairwise potential functions. For instance, if each pairwise potential function is submodular (which can be roughly considered as a discrete counterpart of convexity), MRFs can be optimized in polynomial time, *i.e* the exact global optimum can be computed [22]. If, on the other hand, each pairwise potential is assumed to be simply a metric distance function, then, unfortunately, the resulting MRF optimization problem proves to be NP-hard. Nevertheless, not everything is lost in this case, since there still exist efficient algorithms for obtaining an approximately optimal solution (which is the best one can hope to achieve given the NP-hardness of the problem). However, if absolutely no assumptions are imposed on the structure of the pairwise potential functions, then no approximation guarantees can be provided at all and only a local minimum can be returned as a result. All these cases are illustrated in Fig. 1.

Ideally, we would like an MRF optimization algorithm that is able to handle as general MRF energies as possible, but who is also capable of providing solutions that are approximately optimal as well. By referring back to the plot shown in Fig. 1, this statement could be interpreted as follows: we would like the projection of our algorithm on the horizontal axis to be as large as possible (this corresponds to being able to handle MRFs with more general pairwise potentials), but at the same time we would like the vertical projection of our algorithm to be as small as possible, e.g., below the blue dashed horizontal line in Fig. 1 (this corresponds to being able to generate solutions that, although possibly not optimal due to the problems's NP-hardness, yet they approximate well the optimal one).

However, besides the above mentioned issue of the quality of the MRF solutions, another very important issue that comes up in practice is that of the algorithm's computational efficiency. This means that, ideally, besides being able to compute a solution which is as accurate as possible, our algorithm should also be able to do that as fast as possible, *i.e* at a low computational cost. However, these two goals (*i.e* accuracy and speed) are contradicting each other and so, typically, one has to make a trade-off between them. In fact, this issue of computational efficiency has been recently looked at for the important case of dynamic MRFs. These are MRFs whose potential functions are assumed to vary over time and it should be noted that this type of MRFs has been recently introduced into computer vision by the work of Kohli and Torr [4].

Motivated by all these observations mentioned above, we thus raise the following questions here:

- could there be a discrete MRF optimization algorithm, which would be not only sufficiently accurate for the case of single MRFs, but also sufficiently efficient?
- Furthermore, could that method offer a computational advantage regarding efficiency for the case of dynamic MRFs?

With respect to the two questions raised above, this work makes the following contributions:

Computational efficiency for single MRFs: Graph-cut based optimization algorithms, such as the α-expansion method, try to opimize an MRF by solving a series of max-flow problems. Their efficiency is thus largely determined from the efficiency of these max-flow problems, which, in turn, depends on the number of augmenting paths per max-flow. Here, we build upon the framework proposed in [5] in order to derive a new primal-dual MRF optimization method, called Fast-PD [9]. This method, like [5] or α-expansion, also ends up solving a max-flow problem for a series of graphs. However, unlike these techniques, the graphs constructed by Fast-PD ensure that the number of augmentations per max-flow decreases dramatically over time, thus boosting the efficiency of MRF inference. To show this, we prove a generalized relationship between the number of augmentations and the so-called *primal-dual gap* associated with the original MRF problem and its dual. Furthermore, to fully exploit the above property, 2 new extensions are also proposed: an *adapted max-flow algorithm*, as well as an *incremental graph construction* method.

Accuracy of solutions: Despite its efficiency, our method also makes no compromise regarding either the quality of the solutions it generates or the generality of the MRFs it can handle. So, for instance, if the pairwise potentials $V_{pq}(\cdot,\cdot)$ are assumed to be metric functions, then it can be proved that Fast-PD is as powerful as α-expansion, in the sense that it computes exactly the same solution, but with a substantial speedup. Moreover, it applies to a much wider class of MRFs[3], *i.e*, it can even handle MRFs for which the pairwise potentials $V_{pq}(\cdot,\cdot)$ are non-metric functions. In fact, in all these cases, the proposed method can provide theoretical (*i.e*, worst-case) upper bounds about how far the energy of the generated solution can be from the unknown optimal MRF energy. But besides these theoretical upper bounds, our method is also capable of providing per-instance upper bounds, *i.e* bounds that are specific to each particular problem the algorithm has been tested on. In practice, these bounds prove, of course, to be much tighter (*i.e*, much closer to 1) than the worst-case upper bounds and hence can be very useful for assessing how well the algorithm performed in each particular case.

Efficiency for dynamic MRFs: Furthermore, besides being able to significantly speed up the optimization of static MRFs, our method can also be used for boosting the efficiency of dynamic MRFs. Two works have been proposed in this regard recently [4], [3]. These methods can be applied to dynamic MRFs that are binary or have convex priors. On the contrary, Fast-PD naturally handles a much wider class of dynamic MRFs, and, as we shall see, it manages to achieve that by also exploiting information coming from a problem, which is dual to the original MRF problem. Fast-PD can thus be thought of as a generalization of previous techniques.

The rest of this paper is going to be structured as follows. In section II, we review related work, while, in section III, we briefly review the work of [5] upon which we build in order to derive our algorithm. As already mentioned in the introduction, our algorithm relies heavily on the duality theory of Linear Programming (LP) and, in particular, on the primal-dual schema. We thus start that section by explaining the underlying ideas behind that very general technique, while we then continue by explaining how the primal-dual schema can be applied to the case of MRF optimization. The Fast-PD algorithm is then described in section IV. Its efficiency for optimizing single MRFs is further analyzed in section V, where related results and some important extensions of Fast-PD are presented as well. Section VI focuses on the issue of how Fast-PD can be used for boosting the performance of dynamic MRFs, while it also contains some related experimental results. Finally, we conclude in section VII. We also note that, for reasons of clarity of the presentation, the technical proofs for some of the theorems in this paper have been postponed to appendices A and B.

[3] Fast-PD requires only $d(a,b)\geq 0,\ d(a,b)=0 \Rightarrow a=b$

II. Related work

MRF optimization has motivated a great deal of research in computer vision over the last years and, as a result, many algorithms have been proposed on this topic. Algorithms such as ICM (Iterated Conditional Modes), HCF (Highest Confidence First), relaxation labeling or simulated annealing were among the first ones to be used in this regard. These early algorithms, however, have proven not to be particularly effective in handling difficult MRF optimization tasks such as those encountered in the areas mentioned above. A significant progress in MRF optimization has been achieved over the last years, primarily due to the introduction of two classes of methods, those based on graph-cuts [2], [22], [23], [24] and those based on belief propagation [16], [17], [18], [19], [20], [21], [33] (which is a generalization of dynamic programming to the case of tree-structured graphs). Methods such as the α-expansion or the $\alpha\beta$-swap are two characteristic examples from the former class, whereas loopy Belief Propagation and Tree Reweighted message-passing (with its variants) are some important examples that belong to the latter class. We note here that both types of techniques have been applied with great success to many problems in computer vision or image analysis [7], [15].

Another reason for the significant progress made in MRF optimization over the last years has also been due to exploiting the fact that there exists a very close relationship between linear programming and MRFs. This holds true because MRF optimization can actually be formulated as a problem of linear integer programming (see problem (12) below) and, so, by relaxing the integrality constraints of that integer program, a standard LP then results[4]. The resulting LP relaxation was known since 1970's and has been first used by Schlesinger [40] for the purpose of deriving MRF optimization techniques. Two such techniques have been the max-sum diffusion and the augmenting DAG algorithms. We note here that a very nice and thorough review of Schlesinger's work, including the 2 algorithms mentioned above, is provided by Werner in [39]. This LP relaxation has also been exploited by other authors as well (e.g., see [11]). Furthermore, it has recently been used for showing that there exists a very tight connection between various message-passing techniques for MRF optimization and linear programming. In fact, it is nowadays well-known that reparameterizations (also called equivalent-transformations in [39]), which are essentially one way of making use of the dual LP, are also virtually equivalent to passing messages, as the seminal papers by Wainwright et al. [17] and Kolmorogov [21] show.

Our work also makes use of the same LP relaxation mentioned above. In order to make use of it, however, it relies on the primal-dual schema. This is a well known technique in combinatorial

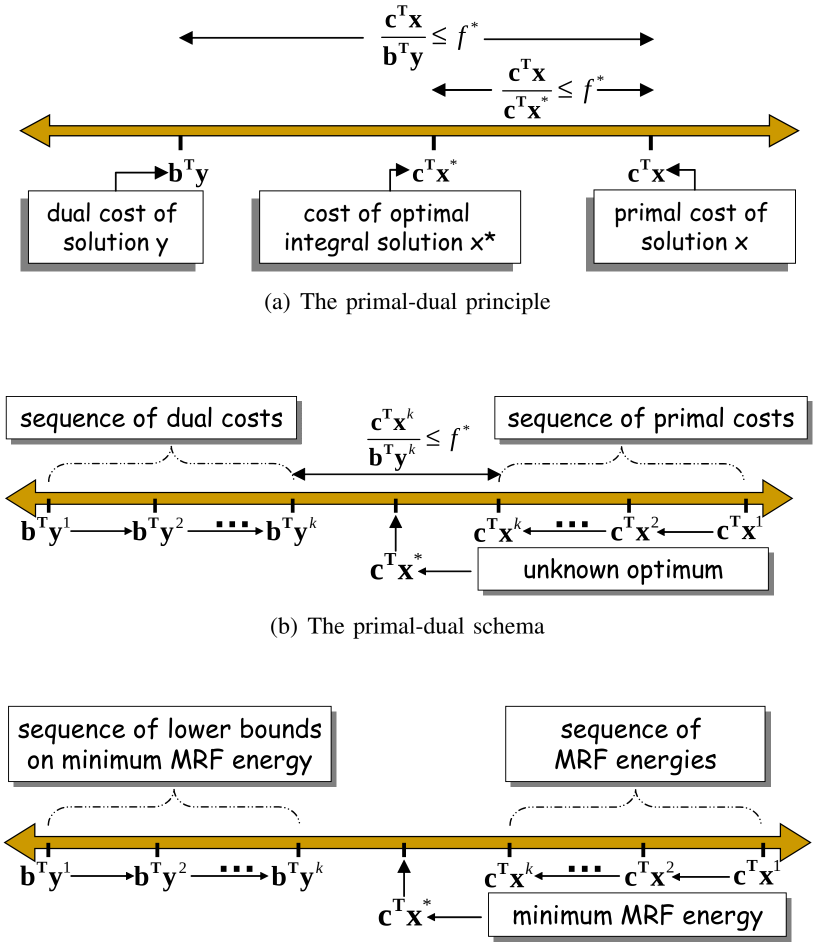
[4] We also note that other LP-relaxations (which are tighter, i.e., better) exist as well [44].

optimization, which was first introduced into computer vision by Komodakis et al. in [5], [10]. Our work thus builds upon the framework provided in [10], but, compared to that work, it provides many new important extensions and contributions. In particular, here we derive primal-dual algorithms which are an order of magnitude faster than those presented in [10]. Given the very important role that computational efficiency plays in vision applications (due to the large nature of the MRFs usually encountered in practice), we believe this is an important advance. Furthermore, despite their efficiency, we prove that the derived algorithms retain significant optimality guarantees and thus provide solutions that are almost of minimum MRF energy. We also present an adapted max-flow algorithm as well as an incremental method for constructing the max-flow graphs, which are two techniques that can also contribute to increasing the efficiency of MRF algorithms. From a theoretical point of view, we show how the primal-dual gap (associated with the LP relaxation and its dual) affects the computational efficiency of MRF optimization, and, to this end, we prove that there exists a close relationship between this primal-dual gap and the number of path augmentations that need to be computed during the course of optimizing an MRF. Another point is that we extend the primal-dual framework to the case of dynamic MRFs, which is another very important application area that is starting to gain a lot of attention from the computer vision community. Last, but not least, we provide experimental validation using a different set of applications where computational efficiency might be more critical than the actual recovery of the optimal solution. One example is the case of medical image and volume registration where we are able to produce results of better quality when compared with the state of the art with a speed gain of a factor of 100. Therefore we do believe that the important gain in terms of computational complexity make the generic formulation of MRFs far more interesting in the field of computer vision and medical imaging. Before proceeding we also note that a much less detailed description of our work has previously appeared in [9].

III. Primal-dual MRF optimization algorithms

A. *The primal-dual schema*

The primal-dual schema is a technique which has been very well known to people in combinatorial optimization for many years now. It started as a very general technique for solving linear programming (LP) problems. As a result, it has been initially used for mainly deriving exact polynomial-time algorithms to many cornerstone problems in combinatorial optimization, including max-flow, matching, shortest path, minimum branching and minimum spanning tree [14]. However, soon, it was realized that it can be a very powerful tool for deriving approximation algorithms to problems of linear integer programming as well. It has thus been used for providing good approximation algorithms to many NP-hard combinatorial



(a) The primal-dual principle

(b) The primal-dual schema

(c) The primal-dual schema for MRFs

Fig. 2: **(a)** By weak duality, we know that the optimal cost $\mathbf{c}^T\mathbf{x}^*$ will always lie between the costs $\mathbf{b}^T\mathbf{y}$ and $\mathbf{c}^T\mathbf{x}$ of any pair $(\mathbf{x}, \mathbf{y})$ of integral-primal and dual feasible solutions. Therefore, if $\mathbf{b}^T\mathbf{y}$ and $\mathbf{c}^T\mathbf{x}$ are close enough (e.g. their ratio is $\leq f^*$), so are $\mathbf{c}^T\mathbf{x}^*$ and $\mathbf{c}^T\mathbf{x}$ (e.g. their ratio is $\leq f^*$ as well), thus proving that $\mathbf{x}$ is an f^*-approximation to $\mathbf{x}^*$. **(b)** According to the primal-dual schema, dual and integral-primal feasible solutions make local improvements to each other, thus generating sequences of primal and dual costs. This is repeated until the final costs $\mathbf{b}^T\mathbf{y}^t$, $\mathbf{c}^T\mathbf{x}^t$ are close enough (e.g., their ratio is $\leq f^*$), in which case we can apply the primal-dual principle (as in Fig. (a)) and thus conclude that $\mathbf{x}^t$ is an f^*-approximation to the optimal solution $\mathbf{x}^*$. **(c)** When the primal-dual schema is applied to the case of MRF optimization, one eventually computes a sequence of MRF energies (corresponding to the primal costs), as well as a sequence of lower bounds on the minimum MRF energy (corresponding to the dual costs).

problems such as those of set-cover, steiner-network, scheduling, steiner tree, feedback vertex set, just to mention a few of them [12], [13]. In the latter case, the basic idea behind using that technique is quite simple and can be explained as follows:

Suppose we want to solve the following linear integer program:

$$\min \mathbf{c}^T\mathbf{x} \tag{2}$$

$$\text{s.t. } \mathbf{A}\mathbf{x} = \mathbf{b} \tag{3}$$

$$\mathbf{x} \in \mathbb{N} \tag{4}$$

Because of the fact that constraints (4) require the components of the solution $\mathbf{x}$ to be natural numbers, the resulting optimization problem becomes of course NP-hard. This means that we can only hope for getting an approximate solution to that problem. In order to achieve this goal, we thus relax the complicating integrality constrains (4) as follows:

$$\min \mathbf{c}^T\mathbf{x} \tag{5}$$

$$\text{s.t. } \mathbf{A}\mathbf{x} = \mathbf{b} \tag{6}$$

$$\mathbf{x} \geq 0 \tag{7}$$

We thus obtain a linear program, which is, of course, a much easier problem. However, the reason that we have tried to do that is not because we actually want to solve this LP, but simply because we want to take its dual problem, which forms, of course, another LP:

$$\max \mathbf{b}^T\mathbf{y} \tag{8}$$

$$\text{s.t. } \mathbf{A}^T\mathbf{y} \leq \mathbf{c} \tag{9}$$

As a result of this process, we have ended up with 2 problems: our original linear integer program (this will be referred to as the primal problem), as well as a dual linear program. According to the primal-dual schema, we must then find a pair $(\mathbf{x}, \mathbf{y})$ of primal and dual solutions such that the corresponding primal-dual gap (*i.e*, the distance between the cost $\mathbf{c}^T\mathbf{x}$ of the primal solution and the cost $\mathbf{b}^T\mathbf{y}$ of the dual solution) is small enough, *e.g*, one way for measuring this is by requiring that the ratio between these two costs is smaller than, say, f^*. If we manage to find such a primal-dual pair, then it is guaranteed that the primal solution $\mathbf{x}$ we have estimated is actually an f^*-approximation to the unknown optimal solution of our original problem, as the following theorem states:

Theorem 1 (The primal-dual principle)**.** *If* $\mathbf{x}$ *and* $\mathbf{y}$ *are integral-primal and dual feasible solutions* *satisfying:*

$$\mathbf{c}^T\mathbf{x} \leq f^* \cdot \mathbf{b}^T\mathbf{y} \tag{10}$$

then $\mathbf{x}$ *is an* f^**-approximation to the optimal integral solution* $\mathbf{x}^*$*, i.e. it holds* $\mathbf{c}^T\mathbf{x}^* \leq \mathbf{c}^T\mathbf{x} \leq f^* \cdot \mathbf{c}^T\mathbf{x}^*$

The reason that this principle holds true is rather simple and is illustrated graphically in Figure 2(a). In particular, in this case, due to weak duality it will hold that the cost $\mathbf{c}^T\mathbf{x}^*$ of the optimal integral solution will always lie between the dual cost $\mathbf{b}^T\mathbf{y}$ and the primal cost $\mathbf{c}^T\mathbf{x}$, i.e. it will hold $\mathbf{b}^T\mathbf{y} \leq \mathbf{c}^T\mathbf{x}^* \leq \mathbf{c}^T\mathbf{x}$. Therefore, if the ratio between the two costs $\mathbf{c}^T\mathbf{x}$ and $\mathbf{b}^T\mathbf{y}$ is smaller than f^* then the same thing will necessarily apply to the ratio between the costs $\mathbf{c}^T\mathbf{x}$ and $\mathbf{c}^T\mathbf{x}^*$, which thus proves the above theorem.

Of course, one cannot expect to come up with such a good primal-dual pair $(\mathbf{x}, \mathbf{y})$ right from the beginning. So what typically happens is that the primal-dual algorithm proceeds iteratively, where each iteration consists of one update of both the primal and dual variables. Hence, given a current pair of solutions, say, $(\mathbf{x}^k, \mathbf{y}^k)$, we act as follows: first, based on our current dual solution $\mathbf{y}^k$, we try to improve our primal solution, thus generating a new solution $\mathbf{x}^{k+1}$, so that its cost $\mathbf{c}^T\mathbf{x}^{k+1}$ comes closer to the dual cost $\mathbf{b}^T\mathbf{y}^k$. Similarly, based on our new primal solution $\mathbf{x}^{k+1}$, we try to improve our dual solution, thus generating a new solution $\mathbf{y}^{k+1}$, so that its cost $\mathbf{b}^T\mathbf{y}^{k+1}$ also comes closer to the primal cost $\mathbf{c}^T\mathbf{x}^{k+1}$. In this manner a new pair $(\mathbf{x}^{k+1}, \mathbf{y}^{k+1})$ is generated, which means that one iteration of the algorithm has just been completed. This is, of course, repeated (thus producing sequences of primal and dual costs, *e.g*, see Fig. 2(b)) until the final primal-dual gap becomes small enough, in which case the algorithm terminates. The iterative procedure that has been just described lies at the heart of any primal-dual algorithm.

One remarkable thing to note here is that despite the fact that we apply only local improvements to both the primal and dual solutions during our algorithm, yet we manage to extract an almost globally optimal solution at the end. Also, another important thing to note here is that, instead of working directly with the primal and dual costs, typically one works with the complementary slackness conditions associated with the linear program. These conditions are thus relaxed by a certain factor f^* and then one has to find a primal-dual pair $(\mathbf{x}, \mathbf{y})$ that satisfies these relaxed conditions. This can be shown to be the same as trying to make the primal-dual gap smaller than f^*, as the following theorem certifies:

Theorem 2 (Relaxed Complementary Slackness)**.** *If the pair* $(\mathbf{x}, \mathbf{y})$ *of integral-primal and dual feasible solutions satisfies the so-called relaxed primal complementary slackness conditions:*

$$\forall\ x_j > 0 \Rightarrow \sum_{i=1}^{m} a_{ij} y_i \geq c_j / f_j\ , \tag{11}$$

where we also assume that $f^* = \max_j f_j$, then the primal-dual gap for the pair $(\mathbf{x}, \mathbf{y})$ is smaller than f^* and therefore $\mathbf{x}$ is an f^*-approximation to the optimal integral solution.

Based on this theorem, as well as on all of the previous observations, the following iterative schema can thus be applied during a primal-dual based approximation algorithm:

Theorem 3 (The primal-dual schema)**.** *Keep generating pairs of integral-primal, dual solutions $\{(\mathbf{x}^k, \mathbf{y}^k)\}_{k=1}^t$, until the elements $\mathbf{x}^t$, $\mathbf{y}^t$ of the last pair are both feasible and satisfy the relaxed primal complementary slackness conditions.*

One last thing, that is worthy of mentioning, is that one can derive different approximation algorithms simply by using different relaxations of the complementary slackness conditions (*e.g*, by using different f_j in (11)).

B. Applying the primal-dual schema to MRF optimization

The above schema has been used in [5] for deriving approximation algorithms that can be be applied to a very wide class of MRFs. For this purpose, the MRF optimization problem was first cast as a problem of integer programming as follows [40], [39], [11]:

$$\min \sum_{p\in\mathcal{V}} \Big(\sum_{a\in\mathcal{L}} \mathfrak{c}_p(a) x_p(a) \Big) + \sum_{(p,q)\in\mathcal{E}} \Big(w_{pq} \sum_{a,b\in\mathcal{L}} d(a,b) x_{pq}(a,b) \Big) \tag{12}$$

$$\text{s.t.} \sum\nolimits_a x_p(a) = 1 \qquad \forall\ p \in V \tag{13}$$

$$\sum\nolimits_a x_{pq}(a,b) = x_q(b) \qquad \forall\ b \in L,\ (p,q) \in E \tag{14}$$

$$\sum\nolimits_b x_{pq}(a,b) = x_p(a) \qquad \forall\ a \in L,\ (p,q) \in E \tag{15}$$

$$x_p(\cdot),\ x_{pq}(\cdot,\cdot) \in \{0,1\} \tag{16}$$

As can be seen, the MRF energy function has essentially been linearized by introducing two types of extra indicator (*i.e*, binary) variables: the unary variables $\{x_p(\cdot)\}$, that are used for indicating which label is assigned to each MRF node (*i.e*, $x_p(a) = 1 \Leftrightarrow$ label a has been assigned to p), as well as the pairwise variables $\{x_{pq}(\cdot,\cdot)\}$, that are used for indicating which pair of labels is assigned to each pair of neighboring nodes (*i.e*, $x_{pq}(a,b) = 1 \Leftrightarrow$ labels a, b have been assigned to nodes p, q). Of course, in order to obtain a equivalent formulation to the MRF problem, we also need to impose the additional linear constraints (13)-(15) on these binary variables. Constraints (13) simply encode the fact that only

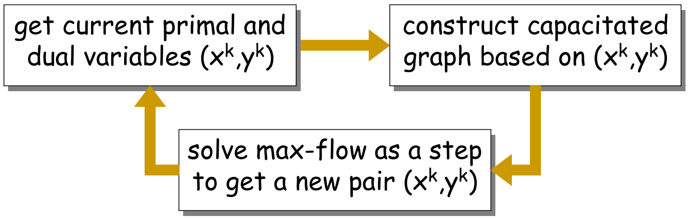

get current primal and dual variables (x^k,y^k)
construct capacitated graph based on (x^k,y^k)
solve max-flow as a step to get a new pair (x^k,y^k)

Fig. 3: In the case of MRF optimization, each iteration of the primal-dual schema essentially involves solving a max-flow problem for a capacitated graph that is constructed based on the current pair $(\mathbf{x}^k, \mathbf{y}^k)$ of primal and dual variables. The resulting flows guide us on how to update the primal, as well as the dual variables in order to get a new pair $(\mathbf{x}^k, \mathbf{y}^k)$ for the next iteration.

one label can be assigned to each node, while constraints (14)-(15) enforce consistency between the unary and the pairwise variables in the sense that they ensure that if $x_p(a) = x_q(b) = 1$, then it should hold $x_{pq}(a,b) = 1$ as well. If the indicator variables are assumed to satisfy all of the above constraints, then it can be verified very easily that the above linear integer program is indeed equivalent to our original problem of minimizing the MRF energy. Given this fact, we can therefore proceed and apply the primal-dual schema to it. This means that we must relax the integrality constraints (16) to the constraints $x_p(\cdot) \geq 0, x_p(\cdot,\cdot) \geq 0$, take the dual to the resulting linear program and choose a proper relaxation of the complementary slackness conditions.

When applying the primal-dual schema to this case (see Fig. 2(c)), essentially, one ends up with computing a sequence of MRF energies (these correspond to the primal costs), as well as a sequence of lower bounds on the unknown minimum MRF energy (these bounds corresponds to the dual costs). After performing all the related analysis (we refer the interested reader to [5] for further details), it turns out that, in this case, each iteration of the primal-dual schema (*i.e*, each update of the primal and the dual variables) reduces to solving a max-flow problem for a certain capacitated graph (see Fig. 3). This means that the flows resulting from solving this max-flow problem tell us how to update both the dual variables, as well as the primal variables associated with our problem. Also, note that this max-flow graph is not static, but changes per iteration. In fact, its construction depends on the current primal-dual pair of solutions, say $(\mathbf{x}^k, \mathbf{y}^k)$. This implies that both the connectivity of that graph, as well as the capacities of its edges are defined based on this pair $(\mathbf{x}^k, \mathbf{y}^k)$. Furthermore, by choosing to use different relaxations of the resulting complementary slackness conditions, the authors in [5] showed that different approximation algorithms can be derived in this manner, all of which are capable of handling a wide variety of MRFs with metric, as well as non-metric pairwise potential functions. In addition, since in every case the generated solutions provably satisfy a chosen set of relaxed complementary slackness conditions, worst-case (*i.e*, theoretical) guarantees about the optimality of these solutions can always be provided.

For the sake of simplicity, in this work we will concentrate on one particular choice of relaxed complementary conditions, but it should be noted that the techniques described here apply equally well to all relaxed complementary slackness conditions mentioned in [5]. In particular, we will focus here on the following conditions:

$$h_p(x_p) = \min_{a\in\mathcal{L}} h_p(a), \quad \forall p\in\mathcal{V} \tag{17}$$

$$y_{pq}(x_p) + y_{qp}(x_q) = w_{pq} d(x_p, x_q), \quad \forall pq\in\mathcal{E} \tag{18}$$

$$y_{pq}(a) + y_{qp}(b) \leq 2 w_{pq} d_{\max}, \qquad \forall pq\in\mathcal{E}, a\in\mathcal{L}, b\in\mathcal{L} \tag{19}$$

As shown in [5], if the above conditions hold true then the solution $\mathbf{x}$ defines a labeling which is an f^*-approximation to the optimal labeling of the MRF, where $f^*{=}2\frac{d_{\max}}{d_{\min}}$.[5]

In these formulas, $\mathbf{x}{=}\{x_p\}_{p\in\mathcal{V}}$ denotes the primal variables, which are used for specifying the *active labels*, i.e., the labels assigned to the MRF nodes (*e.g*, x_p denotes the label assigned to node p), while $y_{pq}(\cdot)$ and $h_p(\cdot)$ are dual variables (called *balance* and *height* variables respectively) that are coming from the following LP, which is dual to the LP relaxation of integer program (12):

$$\max \sum_p z_p \tag{20}$$

$$\text{s.t. } z_p \leq \min_{a\in\mathcal{L}} h_p(a) \tag{21}$$

$$y_{pq}(a) + y_{qp}(b) \leq w_{pq} d(a,b), \quad \forall (p,q) \in \mathcal{E},\ (a,b) \in \mathcal{L}\times\mathcal{L} \tag{22}$$

As can be seen, there exist 2 balance variables $y_{pq}(a), y_{qp}(a)$ per edge (p,q) and label a, as well as 1 height variable $h_p(a)$ per node p and label a. Variables $y_{pq}(a), y_{qp}(a)$ are also called *conjugate* and, for the dual solution to be feasible, these must be set opposite to each other, *i.e*: $y_{qp}(\cdot) \equiv -y_{pq}(\cdot)$. Furthermore, the height variables are always defined in terms of the balance variables as follows:

$$h_p(\cdot) \equiv \mathfrak{c}_p(\cdot) + \sum\nolimits_{q:qp\in\mathcal{E}} y_{pq}(\cdot). \tag{23}$$

Note that, due to (23), only the vector $\mathbf{y}$ (of all balance variables) is needed for specifying a dual solution. In addition, for simplifying conditions (18),(19), one can also define:

[5] $d_{\max} {\equiv} \max_{a,b} d(a,b), \quad d_{\min} {\equiv} \min_{a\neq b} d(a,b)$

```
 1: $[\mathbf{x},\mathbf{y}]\leftarrow$INIT_DUALS_PRIMALS( ); $\mathbf{x}_{\mathrm{old}}\leftarrow\mathbf{x}$
 2: for each label $c$ in $\mathcal{L}$ do
 3:     $\mathbf{y}\leftarrow$ PREEDIT_DUALS$(c,\mathbf{x},\mathbf{y})$;
 4:     $[\mathbf{x}',\mathbf{y}']\leftarrow$UPDATE_DUALS_PRIMALS$(c,\mathbf{x},\mathbf{y})$;
 5:     $\mathbf{y}'\leftarrow$POSTEDIT_DUALS$(c,\mathbf{x}',\mathbf{y}')$;
 6:     $\mathbf{x}\leftarrow\mathbf{x}'$; $\mathbf{y}\leftarrow\mathbf{y}'$;
 7: end for
 8: if $\mathbf{x}\neq\mathbf{x}_{\mathrm{old}}$ then
 9:     $\mathbf{x}_{\mathrm{old}}\leftarrow\mathbf{x}$; goto 2;
10: end if
```

Fig. 4: The primal dual schema for MRF optimization.

$$\mathrm{load}_{pq}(a,b)\equiv y_{pq}(a)+y_{qp}(b). \tag{24}$$

For instance, by using the above definition, conditions (18), (19) can then be simplified as follows:

$$\mathrm{load}_{pq}(x_p,x_q)=w_{pq}d(x_p,x_q), \qquad \forall pq\in\mathcal{E} \tag{25}$$

$$\mathrm{load}_{pq}(a,b)\leq 2w_{pq}d_{\max}, \qquad \forall pq\in\mathcal{E}, a\in\mathcal{L}, b\in\mathcal{L} \tag{26}$$

and so, whenever we refer to conditions (18), (19) hereafter, we will implicitly refer to conditions (25),(26) as well.

The primal and the dual variables are, of course, iteratively updated until all conditions (17)-(19) hold true, *i.e*, until the chosen relaxed complementary slackness conditions are satisfied, which forms the main goal of the primal-dual algorithm for MRF optimization. The basic structure of such an algorithm can be seen in Fig. 4. During an inner c-iteration (lines 3-6 in Fig. 4), a label c is selected and a new primal-dual pair of solutions $(\mathbf{x}',\mathbf{y}')$ is generated based on the current pair $(\mathbf{x},\mathbf{y})$. To this end, among all balance variables $y_{pq}(.)$, only the balance variables of c-labels (*i.e* $y_{pq}(c)$) are updated during a c-iteration. $|\mathcal{L}|$ such iterations (*i.e* one c-iteration per label c in $\mathcal{L}$) make up an outer iteration (lines 2-7 in Fig. 4), and the algorithm terminates if no change of label takes place at the current outer iteration.

During an inner iteration, the main update of the primal and dual variables takes place inside UPDATE_DUALS_PRIMALS, and (as already mentioned) this update reduces to solving a max-flow problem in a capacitated graph, which will be denoted by $\mathcal{G}^c$ hereafter. Furthermore, the routines PREEDIT_DUALS and POSTEDIT_DUALS simply apply corrections to the dual variables before and after this main update, *i.e*

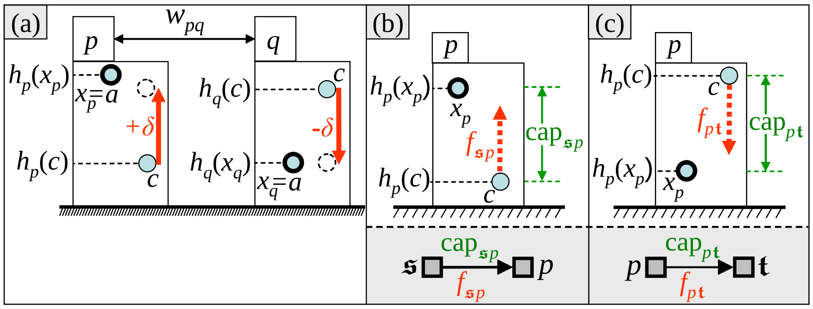


Fig. 5: **(a)** Dual variables' visualization for a simple MRF with 2 nodes $\{p, q\}$ and 2 labels $\{a, c\}$. A copy of labels $\{a, c\}$ exists for every node, and all these labels are represented by balls floating at certain heights. The role of the *height variables* $h_p(\cdot)$ is to specify exactly these heights. Furthermore, balls are not static, but may move (*i.e* change their heights) in pairs by updating conjugate *balance variables*. E.g., here, ball c at p is pulled up by $+\delta$ (due to increasing $y_{pq}(c)$ by $+\delta$) and so ball c at q moves down by $-\delta$ (due to decreasing $y_{qp}(c)$ by $-\delta$). Active labels are drawn with a thicker circle. **(b)** If label c at p is below x_p during a c-iteration, then (due to (17)) we want label c to raise and reach x_p. We thus connect node p to the source $\mathfrak{s}$ with an edge $\mathfrak{s}p$ (*i.e* p is an $\mathfrak{s}$-linked node), and flow $f_{\mathfrak{s}p}$ represents the total raise of c (we also set $\text{cap}_{\mathfrak{s}p} = h_p(x_p) - h_p(c)$). **(c)** If label c at p is above x_p during a c-iteration, then (due to (17)) we want label c not to go below x_p. We thus connect node p to the sink $\mathfrak{t}$ with edge $p\mathfrak{t}$ (*i.e* p is a $\mathfrak{t}$-linked node), and flow $f_{p\mathfrak{t}}$ represents the total decrease in the height of c (we also set $\text{cap}_{p\mathfrak{t}} = h_p(c) - h_p(x_p)$ so that label c will still remain above x_p).

to variables $\mathbf{y}$ and $\mathbf{y}'$ respectively.[6] A family of primal-dual optimization algorithms has been derived in [10], all of which follow the above mentioned scheme. One of them was named PD3$_a$, and we will often refer to it throughout this document in order to illustrate its differences with respect to the primal-dual algorithms presented here (see also [41] for an alternative derivation related to the PD3$_a$ algorithm).

IV. Fast primal-dual MRF optimization

The complexity of the PD3$_a$ primal-dual method largely depends on the complexity of all max-flow instances (one instance per inner-iteration), which, in turn, depends on the number of augmentations per max-flow. So, for designing faster primal-dual algorithms, we first need to understand how the graph $\mathcal{G}^c$, associated with the max-flow problem at a c-iteration of PD3$_a$, is constructed.

To this end, we also have to recall the following intuitive interpretation of the dual variables [5]: for each node p, a separate copy of all labels in $\mathcal{L}$ is considered, and all these labels are represented as balls, which float at certain heights relative to a reference plane. The role of the height variables $h_p(\cdot)$ is then to determine the balls' height (see Figure 5(a)). *E.g* the height of label a at node p is given by $h_p(a)$.

[6] Throughout this paper, we use the following convention for the notation of the dual variables during an inner-iteration: before the UPDATE_DUALS_PRIMALS routine, all dual variables are denoted without an accent, e.g. $y_{pq}(\cdot)$, $h_p(\cdot)$. After UPDATE_DUALS_PRIMALS has updated the dual variables, we always use an accent for denoting these variables, e.g. we write $y'_{pq}(\cdot)$, $h'_p(\cdot)$ in this case.

Also, expressions like “label a at p is below/above label b” imply $h_p(a) \lesseqgtr h_p(b)$. Furthermore, balls are not static, but may move in pairs through updating pairs of conjugate balance variables. *E.g*, in Figure 5(a), label c at p is raised by $+\delta$ (due to adding $+\delta$ to $y_{pq}(c)$), and so label c at q has to move down by $-\delta$ (due to adding $-\delta$ to $y_{qp}(c)$ so that condition $y_{pq}(c) = -y_{qp}(c)$ still holds). Therefore, the role of balance variables is to raise or lower labels. In particular, the value of balance variable $y_{pq}(a)$ represents the partial raise of label a at p due to edge pq, while (by (23)) the total raise of a at p equals the sum of partial raises from all edges of $\mathcal{G}$ incident to p.

Hence, PD3$_a$ tries to iteratively move labels up or down, until all conditions (17)-(19) hold true. To this end, it uses the following strategy: it ensures that conditions (18)-(19) hold at each iteration (which is always easy to do) and is just left with the main task of making the labels’ heights satisfy condition (17) as well in the end (which is the most difficult part, requiring each active label x_p to be the lowest label for p).

For this purpose, labels are moved in groups. In particular, during a c-iteration, only the c-labels are allowed to move (see Fig. 6). Furthermore, the main movement of all c-labels (*i.e* the main update of dual variables $y_{pq}(c)$ and $h_p(c)$ for all p, q) takes place in UPDATE_DUALS_PRIMALS, and this movement has been shown that it can be simulated by pushing the maximum flow through a directed graph $\mathcal{G}^c$ (which is constructed based on the current primal-dual pair $(\mathbf{x}, \mathbf{y})$ at a c-iteration). The nodes of $\mathcal{G}^c$ consist of all nodes of graph $\mathcal{G}$ (the *internal* nodes), plus 2 *external* nodes, the source $\mathfrak{s}$ and the sink $\mathfrak{t}$. In addition, all nodes of $\mathcal{G}^c$ are connected by two types of edges: *interior* and *exterior* edges. Interior

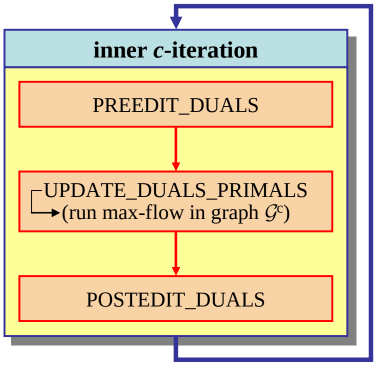
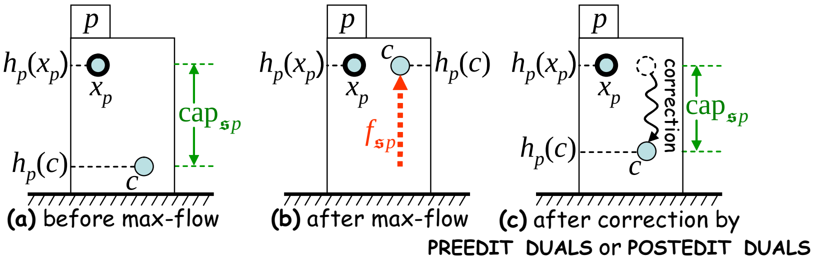


Fig. 6: The basic structure of an inner c-iteration is shown here. During such an iteration, only the c-labels are allowed to move (*i.e* only them can change their heights). The main movement of the c-labels takes place inside the UPDATE_DUALS_PRIMALS routine, and this movement is simulated by pushing the maximum-flow through an appropriate directed graph $\mathcal{G}^c$. However, besides the movement during UPDATE_DUALS_PRIMALS, c-labels also move before and after that routine as well. This happens because routines PREEDIT_DUALS and POSTEDIT_DUALS also apply corrections to the dual variables, and these corrections take place before and after max-flow respectively.



(a) before max-flow **(b)** after max-flow **(c)** after correction by PREEDIT DUALS or POSTEDIT DUALS

Fig. 7: **(a)** Label c at p is below x_p, and thus label c is allowed to raise itself in order to reach x_p. This means that p will be an $\mathfrak{s}$-linked node of graph $\mathcal{G}^c$, *i.e* $\text{cap}_{\mathfrak{s}p}\!>\!0$, and thus a non-zero flow $f_{\mathfrak{s}p}$ (representing the total raise of label c in UPDATE_DUALS_PRIMALS) may pass through edge $\mathfrak{s}p$. Therefore, in this case, edge $\mathfrak{s}p$ may become part of an augmenting path during max-flow. **(b)** After UPDATE_DUALS_PRIMALS (*i.e* after max-flow), label c has managed to raise by $f_{\mathfrak{s}p}$ and reach x_p. Since it cannot go higher than that, no flow can pass through edge $\mathfrak{s}p$, *i.e* $\text{cap}_{\mathfrak{s}p}\!=\!0$, and so no augmenting path may traverse that edge thereafter. **(c)** However, due to some correction applied later to c-label's height by PREEDIT_DUALS or POSTEDIT_DUALS, label c has dropped below x_p once more and p has become an $\mathfrak{s}$-linked node again (*i.e* $\text{cap}_{\mathfrak{s}p}\!>\!0$). Edge $\mathfrak{s}p$ can thus be part of an augmenting path again (as in (a)).

edges come in pairs pq, qp (with one such pair for every 2 neighbors p,q in $\mathcal{G}$), and are responsible for updating the balance variables during UPDATE_DUALS_PRIMALS. In particular, the corresponding flows f_{pq}/f_{qp} represent the increase/decrease of balance variable $y_{pq}(c)$, *i.e* $y'_{pq}(c)\!=\!y_{pq}(c)+f_{pq}-f_{qp}$. Also, as we shall see, the capacities of these edges are responsible to ensure (along with PREEDIT_DUALS, POSTEDIT_DUALS) that conditions (18), (19) hold true.

But for now, in order to understand how to make a faster primal-dual method, it is the exterior edges (which are in charge of the update of height variables during UPDATE_DUALS_PRIMALS), as well as their capacities (which are left with ensuring condition (17) on their own), that are of interest to us. The reason is that these edges determine the number of $\mathfrak{s}$-linked nodes, which, in turn, affects the number of augmenting paths per max-flow. In particular, each internal node connects to either the source $\mathfrak{s}$ (*i.e* it is an $\mathfrak{s}$-*linked* node) or to the sink $\mathfrak{t}$ (*i.e* it is a $\mathfrak{t}$-*linked* node) through one of these exterior edges, and this is done (with the goal of ensuring (17)) as follows: if label c at p is above x_p during a c-iteration (*i.e* $h_p(c)\,>\,h_p(x_p)$), then label c should not go below x_p, or else (17) will be violated for p. Node p thus connects to $\mathfrak{t}$ through directed edge $p\mathfrak{t}$ (*i.e* p becomes $\mathfrak{t}$-linked), and flow $f_{p\mathfrak{t}}$ represents the total decrease in the height of c during UPDATE_DUALS_PRIMALS, *i.e* $h'_p(c)\!=\!h_p(c)\!-\!f_{p\mathfrak{t}}$ (see Fig. 5(c)). Furthermore, the capacity of $p\mathfrak{t}$ is set so that label c will still remain above x_p, *i.e* $\text{cap}_{p\mathfrak{t}} = h_p(c) - h_p(x_p)$. On the other hand, if label c at p is below active label x_p (*i.e* $h_p(c) < h_p(x_p)$), then (due to (17)) label c should raise so as to reach x_p, and so p connects to $\mathfrak{s}$ through edge $\mathfrak{s}p$ (*i.e* p becomes $\mathfrak{s}$-linked), while flow $f_{\mathfrak{s}p}$ represents the total raise of ball c, *i.e* $h'_p(c)\!=\!h_p(c)\!+\!f_{\mathfrak{s}p}$ (see Fig. 5(b)). In this case, we also set $\text{cap}_{\mathfrak{s}p}\!=\!h_p(x_p)\!-\!h_p(c)$.

```
$[\mathbf{x},\mathbf{y}] \leftarrow$ INIT_DUALS_PRIMALS( ):   $\mathbf{x} \leftarrow$ random labels; $\mathbf{y} \leftarrow \mathbf{0}$;
    $\forall pq$, adjust $y_{pq}(x_p)$ or $y_{qp}(x_q)$ so that $\text{load}_{pq}(x_p,x_q) = w_{pq}d(x_p,x_q)$

$\mathbf{y} \leftarrow$ PREEDIT_DUALS($c,\mathbf{x},\mathbf{y}$):
    $\forall pq$, if $\text{load}_{pq}(c,x_q) > w_{pq}d(c,x_q)$ or $\text{load}_{pq}(x_p,c) > w_{pq}d(x_p,c)$
        adjust $y_{pq}(c)$ so that $\text{load}_{pq}(c,x_q) = w_{pq}d(c,x_q)$

$[\mathbf{x}',\mathbf{y}'] \leftarrow$ UPDATE_DUALS_PRIMALS($c,\mathbf{x},\mathbf{y}$):   $\mathbf{x}' \leftarrow \mathbf{x}$; $\mathbf{y}' \leftarrow \mathbf{y}$;
    Construct $\mathcal{G}^c$ and apply max-flow to compute all flows $f_{\mathfrak{s}p}/f_{p\mathfrak{t}}$, $f_{pq}$
    $\forall pq$, $y'_{pq}(c) \leftarrow y_{pq}(c) + f_{pq} - f_{qp}$
    $\forall p$ , if an unsaturated path from $\mathfrak{s}$ to $p$ exists, then $x'_p \leftarrow c$

$\mathbf{y}' \leftarrow$ POSTEDIT_DUALS($c,\mathbf{x}',\mathbf{y}'$): {We denote $\text{load}'_{pq}(\cdot,\cdot) = y'_{pq}(\cdot) + y'_{qp}(\cdot)$}
    $\forall pq$, if $\text{load}'_{pq}(x'_p,x'_q) > w_{pq}d(x'_p,x'_q)$ {This implies $x'_p = c$ or $x'_q = c$}
        adjust $y'_{pq}(c)$ so that $\text{load}'_{pq}(x'_p,x'_q) = w_{pq}d(x'_p,x'_q)$
```

Fig. 8: Fast-PD's pseudocode.

This way, by pushing flow through the exterior edges of $\mathcal{G}^c$, all c-labels that are strictly below an active label try to raise and reach that label during UPDATE_DUALS_PRIMALS[7]. Not only that, but the fewer are the c-labels below an active label (*i.e* the fewer are the $\mathfrak{s}$-linked nodes), the fewer will be the edges connected to the source, and thus the less will be the number of possible augmenting paths. In fact, the algorithm terminates when, for any label c, there are no more c-labels strictly below an active label (*i.e* no $\mathfrak{s}$-linked nodes exist and thus no augmenting paths may be found), in which case condition (17) will finally hold true, as desired. Put another way, UPDATE_DUALS_PRIMALS tries to push c-labels (which are at a low height) up, so that the number of $\mathfrak{s}$-linked nodes is reduced and thus fewer augmenting paths may be possible for the next iteration.

However, although UPDATE_DUALS_PRIMALS tries to reduce the number of $\mathfrak{s}$-linked nodes (by pushing the maximum amount of flow), PREEDIT_DUALS or POSTEDIT_DUALS very often spoil that progress. As we shall see later, this occurs because, in order to restore condition (18) (which is their main goal), these routines are forced to apply corrections to the dual variables (*i.e* to the labels' height). This is abstractly illustrated in Figure 7, where, due to UPDATE_DUALS_PRIMALS (*i.e* due to max-flow), a c-label has initially managed to reach an active label x_p, but it has again dropped below x_p, due to some correction by these routines. In fact, as one can show, the only point where a new $\mathfrak{s}$-linked node can be created is during either PREEDIT_DUALS or POSTEDIT_DUALS.

[7] Equivalently, if c-label at p cannot raise high enough to reach x_p, UPDATE_DUALS_PRIMALS then assigns that c-label as the new active label of p (*i.e* $x'_p = c$), thus effectively making the active label go down. This once again helps condition (17) to become true, and forms the main rationale behind the update of the primal variables $\mathbf{x}$ in UPDATE_DUALS_PRIMALS.

To fix this problem, we will redefine PREEDIT_DUALS, POSTEDIT_DUALS so that they can now ensure condition (18) by using just a minimum amount of corrections for the dual variables, (*e.g* by touching these variables only rarely). To this end, however, UPDATE_DUALS_PRIMALS needs to be modified as well. The resulting algorithm, called Fast-PD, carries the following main differences over PD3$_a$ during a c-iteration (its pseudocode appears in Fig. 8):

- the new PREEDIT_DUALS modifies a pair $y_{pq}(c), y_{qp}(c)$ of dual variables only when absolutely necessary. So, whereas the previous version modified these variables (thereby changing the height of a c-label) whenever $c{\neq}x_p$, $c{\neq}x_q$ (which could happen extremely often), a modification is now applied only if $\text{load}_{pq}(c, x_q){>}w_{pq}d(c, x_q)$ or $\text{load}_{pq}(x_p, c){>}w_{pq}d(x_p, c)$ (which, in practice, happens much more rarely). In this case, a modification is needed (see code in Fig. 8), because the above inequalities indicate that condition (18) will be violated if either (c, x_q) or (x_p, c) become the new active labels for p, q. On the contrary, no modification is needed if the following inequalities are true:

$$\text{load}_{pq}(c, x_q){\leq}w_{pq}d(c, x_q), \qquad \text{load}_{pq}(x_p, c){\leq}w_{pq}d(x_p, c),$$

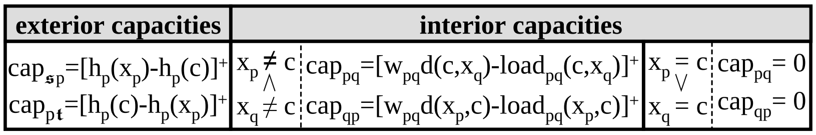
because then, as we shall see below, the new UPDATE_DUALS_PRIMALS can always restore (18) (*i.e* even if (c, x_q) or (x_p, c) are the next active labels - *e.g*, see (31)). In fact, the modification to $y_{pq}(c)$ that is occasionally applied by the new PREEDIT_DUALS can be shown to be the minimal correction that restores exactly the above inequalities (assuming, of course, this restoration is possible)[8].

- Similarly, the balance variables $y'_{pq}(x'_p)$ (with $x'_p = c$) or $y'_{qp}(x'_q)$ (with $x'_q = c$) are modified much more rarely by the new POSTEDIT_DUALS. So, whereas the previous version modified these variables (thereby changing the height of a c-label) whenever they were negative (which, in practice, happened most of the time), the new routine applies a modification only if $\text{load}'_{pq}(x'_p, x'_q){>}w_{pq}d(x'_p, x'_q)$,[9] which may happen only in very seldom occasions (*e.g* if the distance function $d(\cdot, \cdot)$ is a metric, one may then show that this can never happen). If the above inequality does hold true, then POSTEDIT_DUALS simply needs to reduce $\text{load}'_{pq}(x'_p, x'_q)$ so as to just restore (18).

- But, to allow for the above changes, we also need to modify the construction of graph $\mathcal{G}^c$ in UPDATE_DUALS_PRIMALS. In particular, for $c{\neq}x_p$ and $c{\neq}x_q$, the capacities of interior edges pq, qp

[8] This may not be possible if d$(\cdot, \cdot)$ is not a metric distance.

[9] As in (24), we define $\text{load}'_{pq}(a, b){\equiv}y'_{pq}(a){+}y'_{qp}(b)$ for variable $\mathbf{y}'$.



exterior capacities	interior capacities			
$\text{cap}_{sp}=[h_p(x_p)-h_p(c)]^+$	$x_p \neq c$	$\text{cap}_{pq}=[w_{pq}d(c,x_q)-\text{load}_{pq}(c,x_q)]^+$	$x_p = c$	$\text{cap}_{pq}=0$
$\text{cap}_{pt}=[h_p(c)-h_p(x_p)]^+$	$\wedge$ $x_q \neq c$	$\text{cap}_{qp}=[w_{pq}d(x_p,c)-\text{load}_{pq}(x_p,c)]^+$	$\vee$ $x_q = c$	$\text{cap}_{qp}=0$

Fig. 9: Capacities of graph $\mathcal{G}^c$, as set by Fast-PD.

must now be set as follows:[10]

$$\text{cap}_{pq} = \big[w_{pq}d(c,x_q) - \text{load}_{pq}(c,x_q)\big]^+ \ , \tag{27}$$

$$\text{cap}_{qp} = \big[w_{pq}d(x_p,c) - \text{load}_{pq}(x_p,c)\big]^+ \ , \tag{28}$$

where $[x]^+ \equiv \max(x,0)$. Besides ensuring condition (19) (by not letting the balance variables increase too much), the main rationale behind the above definition of interior capacities is to also ensure that (after max-flow) condition (18) will be met by most pairs (p,q), no matter if (c,x_q) or (x_p,c) are the next labels assigned to them (which is good, since we will thus manage to avoid the need for a correction by POSTEDIT_DUALS for all but a few p,q). To see this, the crucial thing to observe is that if, say, (c,x_q) are the next labels for p and q, then capacity cap_{pq} can be shown to represent the increase of $\text{load}_{pq}(c,x_q)$ after max-flow, *i.e*:

$$\text{load}'_{pq}(c,x_q) = \text{load}_{pq}(c,x_q) + \text{cap}_{pq}. \tag{29}$$

Hence, if the following inequality is true as well:

$$\text{load}_{pq}(c,x_q) \le w_{pq}d(c,x_q) \ , \tag{30}$$

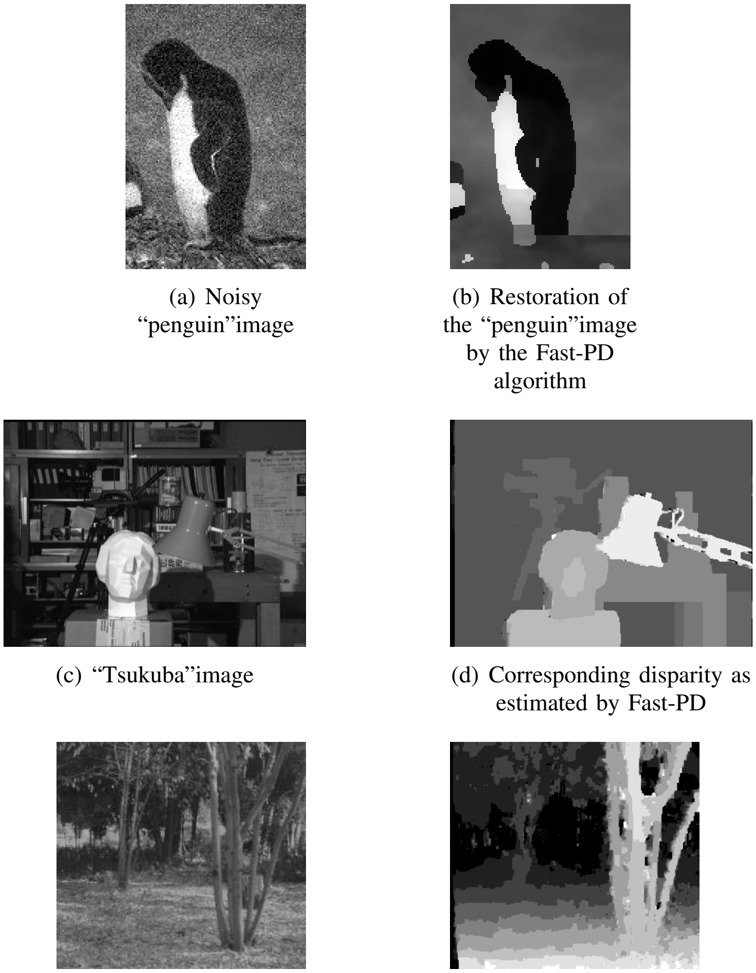
then condition (18) will do remain valid after max-flow, as the following trivial derivation shows:

$$\begin{aligned} \text{load}'_{pq}(c,x_q) &\overset{(29),\,(27)}{=} \text{load}_{pq}(c,x_q) + [w_{pq}d(c,x_q) - \text{load}_{pq}(c,x_q)]^+ \\ &\overset{(30)}{=} \text{load}_{pq}(c,x_q) + [w_{pq}d(c,x_q) - \text{load}_{pq}(c,x_q)] = w_{pq}d(c,x_q) \end{aligned} \tag{31}$$

But this means that a correction may need to be applied by POSTEDIT_DUALS only for pairs p,q violating (30) (before max-flow). However, such pairs tend to be very rare in practice (*e.g*, as one can prove, no such pairs exist when $d(\cdot,\cdot)$ is a metric), and thus very few corrections need to take place.

Fig. 9 summarizes how Fast-PD sets the capacities for all edges of $\mathcal{G}^c$. As already mentioned, based on the interior capacities, one may show that UPDATE_DUALS_PRIMALS (with the help of PREEDIT_DUALS, POSTEDIT_DUALS in a few cases) ensures (18),(19), while, thanks to the exterior capacities, UPDATE_DUALS_PRIMALS

[10] If $c=x_p$ or $c=x_q$, then $\text{cap}_{pq}=\text{cap}_{qp}=0$ as before, *i.e* as in PD3$_a$.



(a) Noisy “penguin”image

(b) Restoration of the “penguin”image by the Fast-PD algorithm

(c) “Tsukuba”image

(d) Corresponding disparity as estimated by Fast-PD

(e) “SRI tree”image

(f) Corresponding disparity as estimated by Fast-PD

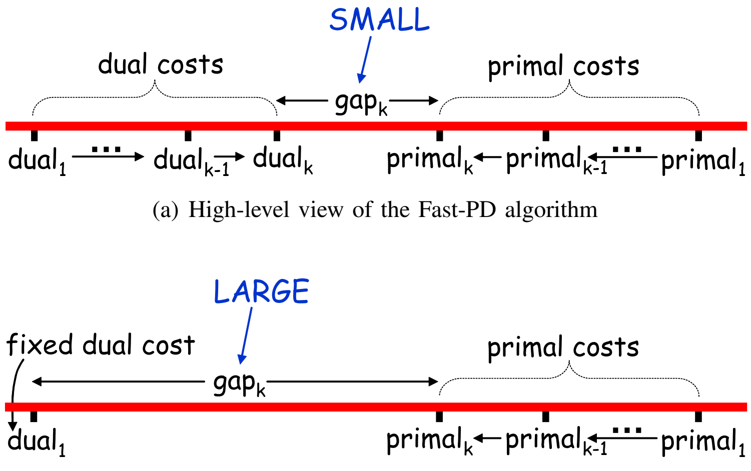
Fig. 10: Image restoration and stereo matching results by the Fast-PD algorithm.

can ensure (17). As a result, the next theorem holds (see appendix A for a complete proof):

Theorem 4. *The last primal-dual pair* $(\mathbf{x}, \mathbf{y})$ *of Fast-PD satisfies conditions* (17)-(19)*, and so* $\mathbf{x}$ *is an* f_{app}*-approximate solution.*

In fact, Fast-PD maintains all good optimality properties of the PD3$_a$ method. *E.g*, for a metric $d(\cdot, \cdot)$, Fast-PD proves to be as powerful as α-expansion (see appendix B for a proof):

Theorem 5. *If* $d(\cdot, \cdot)$ *is a metric, then the Fast-PD algorithm computes the best* c*-expansion after any* c*-iteration.*



(a) High-level view of the Fast-PD algorithm

(b) High-level view of the α-expansion algorithm

Fig. 11: (a) Fast-PD generates pairs of primal-dual solutions iteratively, with the goal of always reducing the primal-dual gap (*i.e* the gap between the resulting primal and dual costs). But, for the case of Fast-PD, this gap can be viewed as a rough estimate for the number of augmentations, and so this number is forced to reduce over time as well. **(b)** On the contrary, α-expansion works only in the primal domain (*i.e* it is as if a fixed dual cost is used at the start of each new iteration) and thus the primal-dual gap can never become small enough. Therefore, no significant reduction in the number of augmentations takes place as the algorithm progresses.

V. Efficiency of Fast-PD for single MRFs

But, besides having all these good optimality properties, a very important advantage of Fast-PD over all previous primal-dual methods, as well as α-expansion, is that it proves to be much more efficient in practice.

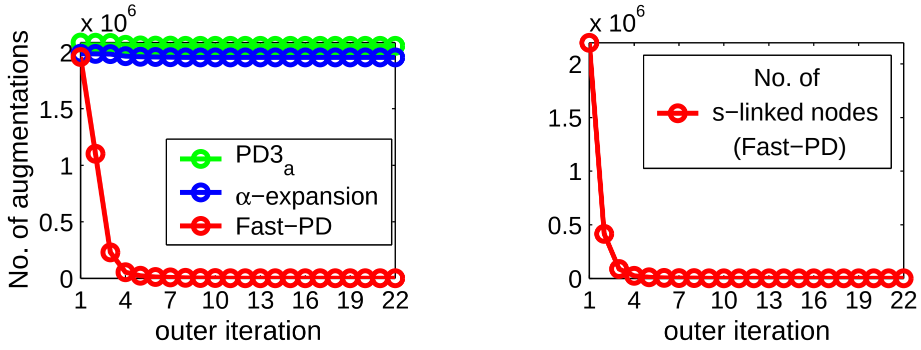
In fact, the computational efficiency for all methods of this kind is largely determined from the time taken by each max-flow problem, which, in turn, depends on the number of augmenting paths that need to be computed. For the case of Fast-PD, the number of augmentations per inner-iteration decreases dramatically, as the algorithm progresses. *E.g* Fast-PD has been applied to the problem of image restoration, where, given a corrupted (by noise) image, one seeks to restore the original (uncorrupted) image back. In this case, labels correspond to intensities, while the singleton potential function $\mathfrak{c}_p(\cdot)$ was defined as a truncated squared difference $\mathfrak{c}_p(a) = \min\{|I_p - a|^2, 10^4\}$ between the label and the observed intensity I_p at pixel p. Fig. 10(b) contains a related result about the denoising of a corrupted (with gaussian noise) "penguin" image (256 labels and a truncated quadratic distance $d(a,b) = \min(|a-b|^2, D)$ - where $D = 200$ - were also used in this case). Fig. 12(a) shows the corresponding number of augmenting paths per outer-iteration (*i.e* per group of $|\mathcal{L}|$ inner-iterations). Notice that, for both α-expansion, as well as PD3$_a$, this number remains very high (*i.e* almost over $2 \cdot 10^6$ paths) throughout all iterations. On the contrary, for the case of Fast-PD, it drops towards zero very quickly, *e.g* only 4905 and 7 paths had to be found during the 8$^{\text{th}}$ and last outer-iteration respectively (obviously, as also shown in Fig. 13(a), this directly affects the total time needed per outer-iteration). In fact, for the case of Fast-PD, it is very typical that, after very few inner-iterations, only a very small number of augmenting paths need to be computed per max-flow, which really boosts the performance in this case.

This property can be explained by the fact that Fast-PD maintains both a primal, as well as a dual solution at each iteration. Fast-PD manages to effectively use this pair of primal and dual solutions from the previous iteration so as to reduce the number of augmenting paths for the next iteration. What essentially happens is illustrated in Fig. 11(a). Intuitively, Fast-PD ultimately wants to close the gap between the primal and the dual cost (see Theorem 1), and, for this, it iteratively generates primal-dual pairs, with the goal of continuously decreasing the size of this gap. But, for Fast-PD, the gap's size can be thought of as, roughly speaking, a rough estimation for the number of augmenting paths per inner-iteration (see Theorem 7 below). Therefore, since Fast-PD manages to reduce this gap throughout its execution, the number of augmenting paths is forced to decrease over time as well, which, in turn, results in improving the efficiency of the max-flow algorithm (recall that a path augmenting max-flow algorithm works by keep finding augmenting paths until there there are no more of them).

On the contrary, a method like α-expansion, that works only in the primal domain, ignores dual solutions completely. It is, roughly speaking, as if α-expansion is resetting the dual solution to zero at the start of each inner-iteration, thus effectively forgetting that solution thereafter (see Fig. 11(b)). For this reason, it fails to substantially reduce the primal-dual gap and thus also fails to achieve a reduction in path augmentations over time, *i.e* across inner-iterations. This, of course, has as a result that more time is needed to be spent per iteration. However, not only the α-expansion, but the PD3$_a$ algorithm as well fails to mimic Fast-PD's behavior (despite being a primal-dual method). As explained in sec. IV, this happens because, in this case, PREEDIT_DUAL and POSTEDIT_DUAL temporarily destroy the gap just before the start of UPDATE_DUALS_PRIMALS, *i.e* just before max-flow is about to begin computing the augmenting paths. (Note, of course, that this destruction is only temporary, and the gap is restored again after the execution of UPDATE_DUALS_PRIMALS).

The above mentioned relationship between the primal-dual gap and the number of augmenting paths is formally described in the next theorem (see appendix C for a proof):

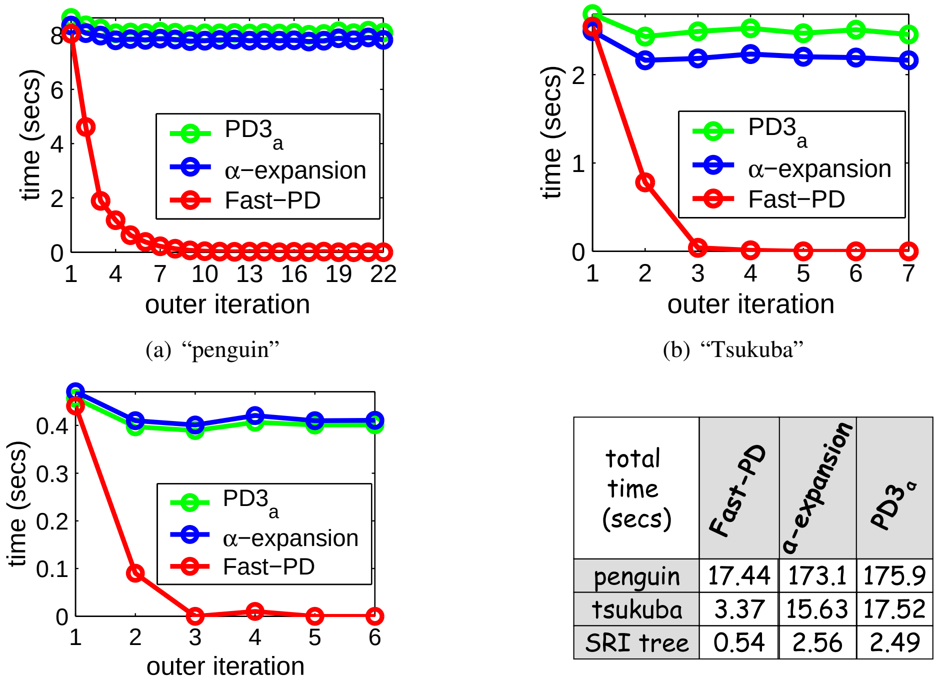
Theorem 6. *For Fast-PD, the primal-dual gap at the current inner-iteration forms an approximate upper bound for the number of augmenting paths at each iteration thereafter.*



(a)

(b)

Fig. 12: **(a)** Number of augmenting paths per outer iteration for the "penguin" example (similar results hold for the other examples as well). Note that only in the case of Fast-PD, this number decreases dramatically over time. **(b)** This property of Fast-PD is directly related to the decreasing number of $\mathfrak{s}$-linked nodes per outer-iteration (this number is shown here for the same example as in (a)).



(a) "penguin"

(b) "Tsukuba"

total time (secs)	Fast-PD	α-expansion	PD3$_a$
penguin	17.44	173.1	175.9
tsukuba	3.37	15.63	17.52
SRI tree	0.54	2.56	2.49

(c) "SRI tree"

(d) Total times

Fig. 13: Total time per outer iteration for the **(a)** "penguin", **(b)** "Tsukuba" and **(c)** "SRI tree" examples. **(d)** Resulting total running times for the same examples. (We note that for all experiments of this paper, a 1.6GHz laptop has been used).

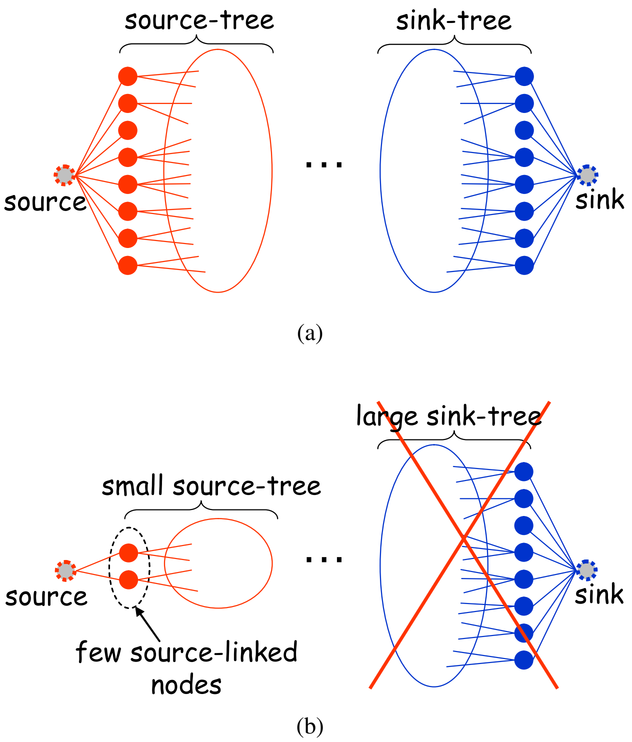
Due to the above mentioned property, the time per outer-iteration decreases dramatically over time. This has been verified experimentally with virtually all problems that Fast-PD has been tested on. *E.g*

Fast-PD has been also applied to the problem of stereo matching. In this case, the conventional measure of SSD (sum of squared differences) or SAD (sum of absolute differences) has been used for the singleton potentials $\mathfrak{c}_p(\cdot)$. Fig. 10(d) contains the resulting disparity (of size 384×288 with 16 labels) for the well-known "Tsukuba" stereo pair, while fig. 10(f) contains the resulting disparity (of size 256×233 with 10 labels) for an image pair from the well-known "SRI tree" sequence (in both cases, a truncated linear distance $d(a,b) = \min(|a-b|, D)$ - with $D = 2$ and $D = 5$ - has been used, while the weights w_{pq} were allowed to vary based on the image gradient at p). Figures 13(b), 13(c) contain the corresponding running times per outer iteration. Notice how much faster the outer-iterations of Fast-PD become as the algorithm progresses, e.g. the last outer-iteration of Fast-PD (for the "SRI-tree" example) lasted less than 1 msec (since, as it turns out, only 4 augmenting paths had to be found during that iteration). Contrast this with the behavior of either the α-expansion or the PD3$_a$ algorithm, which both require an almost constant amount of time per outer-iteration, *e.g* the last outer-iteration of α-expansion needed more than 0.4 secs to finish (*i.e it was more than 400 times slower than Fast-PD's iteration!*). Similarly, for the "Tsukuba" example, α-expansion's last outer-iteration was more than 2000 times slower than Fast-PD's iteration.

A. Max-flow algorithm adaptation

However, for fully exploiting the decreasing number of path augmentations and reduce the running time, one also has to properly adapt the max-flow algorithm. To this end, the crucial thing to observe is that the decreasing number of augmentations is directly related to the decreasing number of $\mathfrak{s}$-linked nodes, as already explained in sec. IV. *E.g* fig. 12(b) shows how the number of $\mathfrak{s}$-linked nodes varies per outer-iteration for the "penguin" example (with a similar behavior being observed for the other examples as well). As can be seen, this number decreases drastically over time. In fact, as implied by condition (17), no $\mathfrak{s}$-linked nodes will finally exist upon the algorithm's termination. Any augmentation-based max-flow algorithm striving for computational efficiency, should certainly exploit this property when trying to extract its augmenting paths.

The most efficient of these algorithms [1] maintains 2 search trees for the fast extraction of these paths, a *source* and a *sink* tree (see Fig. 14(a)). Here, the source tree will start growing by exploring non-saturated edges that are adjacent to $\mathfrak{s}$-linked nodes, whereas the sink tree will grow by exploring non-saturated edges that connect to the $\mathfrak{t}$-linked nodes (whenever these two trees touch each other, this means that a new augmenting path has been found). Of course, the max-flow algorithm will terminate when no unsaturated edges that connect these two trees can be found any more. However, in the case of the Fast-PD algorithm, the source tree turns out to be of much smaller size than the sink tree. This is due



(a)

(b)

Fig. 14: **(a)** Typically, path augmenting max-flow algorithms maintain and expand 2 trees (*i.e*, a source tree and a sink tree) in order to find new augmenting paths. **(b)** However, during the execution of the Fast-PD algorithm, the source tree has a much smaller size than the sink tree (as explained in the text, this is due to the existence of a small number of source-linked nodes). Hence, in this case, it is much more efficient to find new augmenting paths by simply maintaining only the source tree.

to the fact that, as explained above, there can exist only a small number of $\mathfrak{s}$-linked nodes (see Fig. 14(b)). Hence, maintaining the sink tree will be completely inefficient in this case and will certainly not take advantage of the above mentioned property. Therefore, instead of doing that, we propose maintaining only the source tree during max-flow, as this will be a much cheaper operation to perform (*e.g*, in many inner iterations, there can be fewer than 10 $\mathfrak{s}$-linked nodes, but many thousands of $\mathfrak{t}$-linked nodes). Moreover, due to the small size of the source tree, detecting the termination of the max-flow procedure can now be done a lot faster, *i.e* without having to fully expand the large sink tree (which is a very costly operation), thus giving a substantial speedup. In addition to that, for efficiently building the source tree, we keep track of all $\mathfrak{s}$-linked nodes and don't recompute them from scratch each time. In our case, this tracking can be done without cost, since, as explained in sec. IV, an $\mathfrak{s}$-linked node can be created only inside the PREEDIT_DUALS or the POSTEDIT_DUALS routine, and thus can be easily detected. The above simple strategy has been extremely effective for boosting the performance of max-flow, especially when a small number of augmentations were needed.

B. Incremental graph construction

But besides the max-flow algorithm adaptation, one may also modify the way the max-flow graph $\mathcal{G}^c$ is constructed. That is, instead of constructing this capacitated graph $\mathcal{G}^c$ from scratch each time, we also propose an incremental way of setting its capacities (recall that the max-flow graph is not static, but changes per iteration). In fact, our framework provides a principled way of doing this incremental graph construction for the case of general MRFs. What essentially allows us to achieve that is the fact that the Fast-PD algorithm maintains both primal and dual information throughout its execution. The following lemma turns out to be crucial in this regard:

Lemma 1. *Let $\mathcal{G}^c$, $\bar{\mathcal{G}}^c$ be the graphs for the current and previous c-iteration. Let also p, q be 2 neighboring MRF nodes. If, during the interval from the previous to the current c-iteration, no change of label took place for p and q, then the capacities of the interior edges pq, qp in $\mathcal{G}^c$ and of the exterior edges $\mathfrak{s}p, p\mathfrak{t}, \mathfrak{s}q, q\mathfrak{t}$ in $\mathcal{G}^c$ equal the residual capacities of the corresponding edges in $\bar{\mathcal{G}}^c$.*

The proof follows directly from the fact that if no change of label took place for p, q, then none of the height variables $h_p(x_p), h_q(x_q)$ or the balance variables $y_{pq}(x_p), y_{qp}(x_q)$ could have changed. Due to the above lemma, for building graph $\mathcal{G}^c$, we can simply reuse the residual graph of $\bar{\mathcal{G}}^c$ and only recompute those capacities of $\mathcal{G}^c$ for which the above lemma does not hold. This way, an additional speedup can be obtained in some cases.

C. Combining speed with optimality

Fig. 13(d) contains the running times of Fast-PD for various MRF problems. As can be seen from that figure, Fast-PD proves to be much faster than either the α-expansion[11] or the PD3$_a$ method, *e.g* Fast-PD has been more than 9 times faster than α-expansion for the case of the "penguin" image (17.44 secs vs 173.1 secs). In fact, this behavior is a typical one, since Fast-PD has consistently provided at least a 3-9 times speedup for all the problems it has been tested on. However, besides its efficiency, Fast-PD does not make any compromise regarding the optimality of its solutions. On the one hand, this is ensured by theorems 4, 5, which essentially provide worst-case (*i.e*, theoretical) suboptimality bounds. On the other hand, Fast-PD, like any other primal-dual method, can also tell for free how well

[11] We note that the publicly available implementation of [7] has been used for the α-expansion algorithm. Furthermore, since α-expansion cannot be applied when $d(\cdot,\cdot)$ is not a metric, the extension proposed in [6] has been used for the cases where a non-metric distance function $d(\cdot,\cdot)$ was needed.

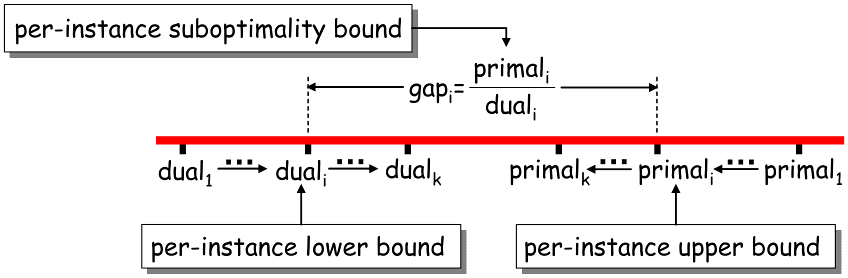
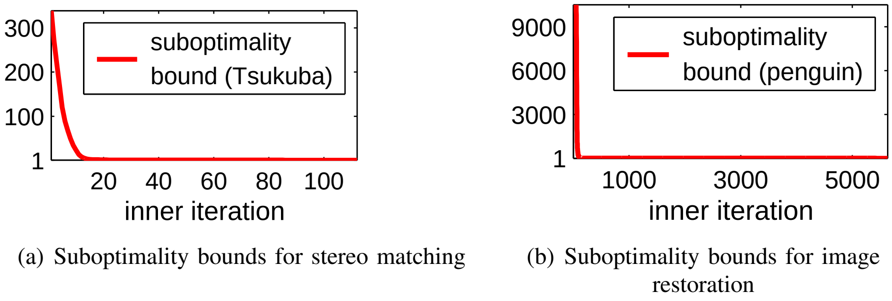


Fig. 15: In the case of a primal-dual MRF optimization method, such as Fast-PD, the cost dual_i of any dual solution generated during the course of the algorithm forms a per-instance lower bound on the optimal MRF energy, say, primal^*, whereas the cost primal_i of any primal solution is obviously a per-instance upper bound on the minimum energy primal^*. Hence, any primal-dual ratio $\text{gap}_i = \text{primal}_i/\text{dual}_i$ forms a per-instance suboptimality bound telling at most how far the current energy is from the unknown optimal energy primal^*. Of course, this suboptimality bound gets refined (*i.e*, becomes smaller) during the execution of the algorithm.



(a) Suboptimality bounds for stereo matching

(b) Suboptimality bounds for image restoration

Fig. 16: Suboptimality bounds (per inner iteration) for a stereo matching problem ("Tsukuba" example), as well as for an image restoration problem ("penguin" example). As can be seen in both cases, the bounds drop very close to 1, which means that the corresponding solutions are almost optimal.

it performed for each particular problem instance it has been tested on. And it can do that at no extra cost by providing per-instance suboptimality bounds, which, in practice, prove to be much tighter than their theoretical counterparts. To understand how this can be done, let $\{\text{primal}_i\}_{i=1}^k$, $\{\text{dual}_i\}_{i=1}^k$ be the sequence of primal and dual costs generated during the course of the algorithm. As already explained in section III, any of the dual costs dual_i forms a per-instance lower bound on the optimal MRF energy, say, primal^*. Furthermore, any of the primal costs primal_i is obviously a per-instance upper bound on this minimum energy primal^*. As a result, any primal-dual ratio $\text{gap}_i = \text{primal}_i/\text{dual}_i$ forms a per-instance suboptimality bound telling at most how far the current energy can be from the unknown optimal energy primal^*. Of course, this suboptimality bound is updated and becomes tighter as the algorithm progresses. *E.g* fig. 16 shows how these ratios vary per inner-iteration for the "tsukuba" and "penguin" problems (with similar results holding for the other problems as well). As one can notice, these ratios finally drop very close to 1, meaning that an almost optimal solution is estimated at the end of the algorithm (and despite the problem being NP-hard). Before proceeding, we should also note that no special tuning of either the singleton or the pairwise potential functions took place for deriving the results in Figure 10. Therefore, by properly adjusting these functions with more care, even better results may be obtained by the Fast-PD algorithm. *E.g* Figure 17 displays the resulting disparity (for the "Tsukuba" image pair), when a Potts function (instead of a truncated linear function) has been used as the distance function $d(\cdot,\cdot)$.

We have also applied the Fast-PD algorithm to the problem of inter or intra modal deformable registration, which is one of the most challenging problems in medical imaging. Towards satisfying smoothness of the deformation field and reducing the dimensionality of the problem, we represent deformation through Free Form Deformations. Our method reformulates registration as an MRF optimization problem, where a set of labels is associated with a set of deformations, and one seeks to attribute a label to each control point such that once the corresponding deformation has been applied, the similarity metric between the source and the target is maximal for all voxels [25]. Schnabel et al. [38] have also proposed a non-rigid image registration method based on B-Spline Free Form Deformation (FFD) together with a gradient-descent optimization. Their approach can be seen as the state-of-the-art in FFD based registration. In order to demonstrate the performance of our deformable registration framework using FastPD, we run both algorithms on the same data with similar registration parameters in terms of equal FFD mesh size and same dissimilarity measure. The test data are two CT volumes showing the heart of pig (see Fig. 18). The volume resolution is 128x128x88. Due to the heart beat a deformation of the shape before registration is clearly visible. While the gradient-descent approach takes more than two hours to converge, the running time of our method is less than 2 minutes (AMD Athlon64 2.21 GHz). Also, visually the results of the

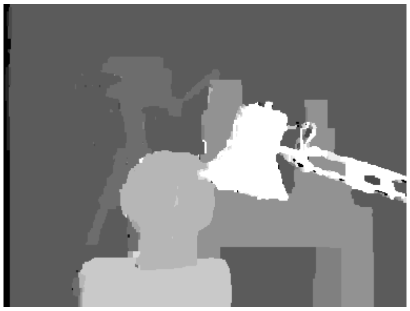


Fig. 17: Disparity for the "Tsukuba" image as estimated by the Fast-PD algorithm in the case where a Potts function has been used for the distance $d(\cdot,\cdot)$.

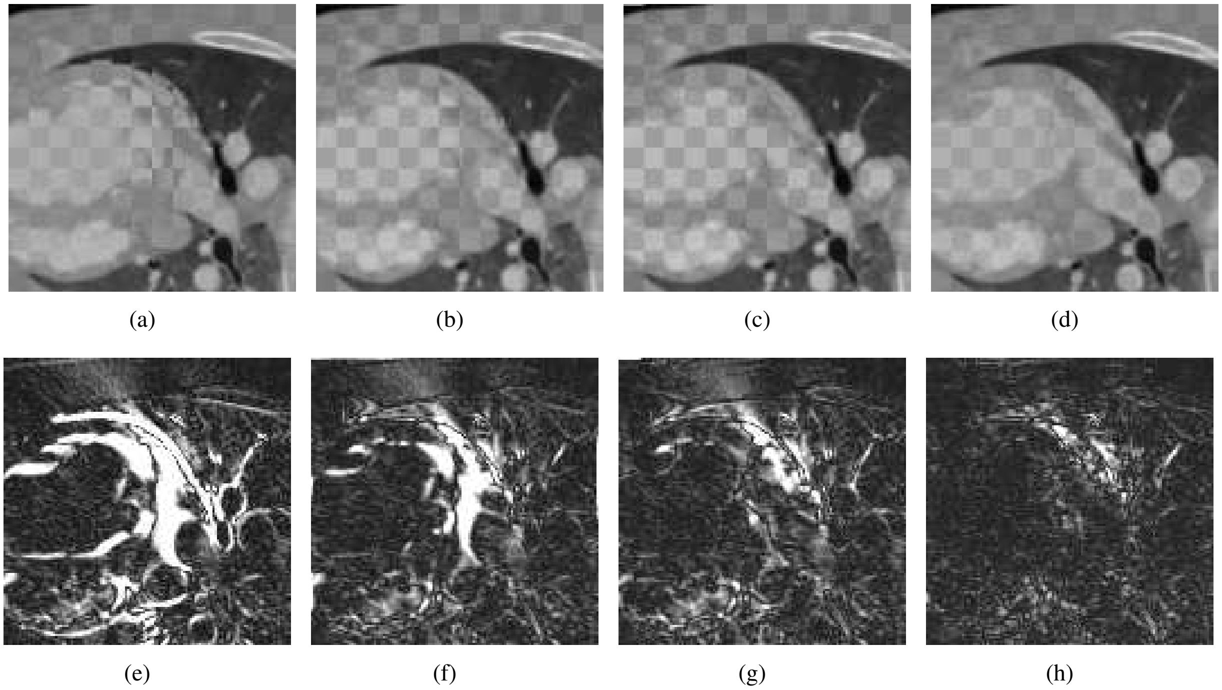
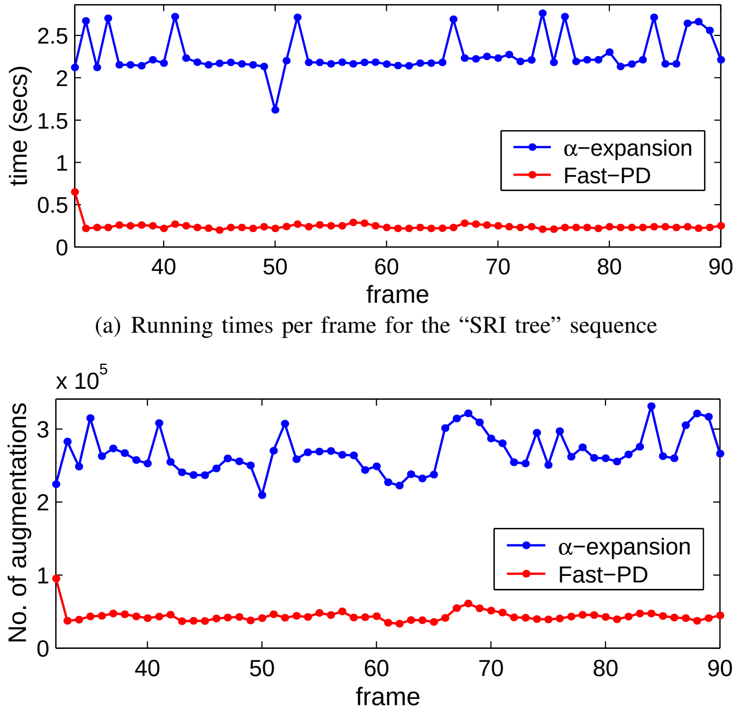


Fig. 18: **(a)** Checkerboard visualization before registration, **(b)** after registration using the method in [38], and **(c)** after registration using our method **(d)** After registration using our approach with pyramidal settings. Same order for the difference images in **(e)**-**(h)**.

registration are slightly better. Within a region of interest enclosing the heart we measure an average SSD error of 12278 before registration. The gradient-descent method achieves an error of 3402, while our method minimizes the error to 3180. Last, but not least, we should mention, that this experiment was not performed to obtain the best registration of the two data sets, but rather to compare the two algorithms. With our standard pyramidal approach we obtain a SSD error of 1233 by same running time of about 2 minutes.

VI. DYNAMIC MRFS

But, besides single MRFs, Fast-PD can be easily adapted to also boost the efficiency for dynamic MRFs [4], *i.e* MRFs varying over time, thus showing the generality and power of the proposed method. In fact, Fast-PD fits perfectly to this task. The implicit assumption here is that the change between successive MRFs is small, and so, by initializing the current MRF with the final (primal) solution of the previous MRF, one expects to speed up inference. A significant advantage of Fast-PD in this regard, however, is that it can exploit not only the previous MRF's primal solution (say $\bar{\mathbf{x}}$), but also its dual solution (say $\bar{\mathbf{y}}$). And this, for initializing current MRF's both primal and dual solutions (say $\mathbf{x}, \mathbf{y}$).



(a) Running times per frame for the “SRI tree” sequence

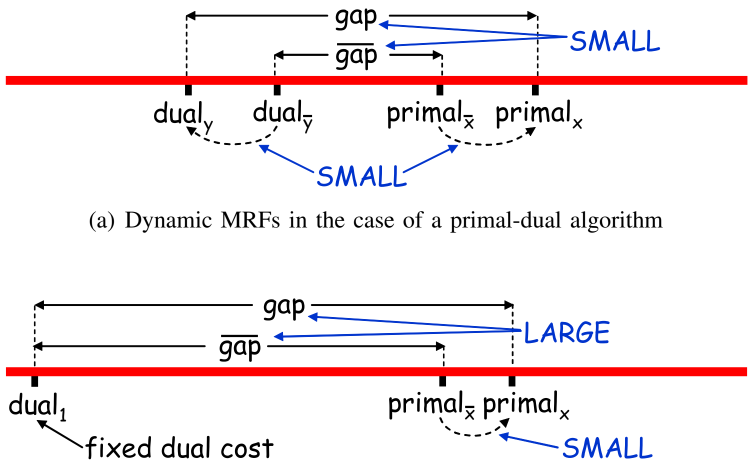
(b) Augmenting paths per frame for the “SRI tree” sequence

Fig. 19: Statistics for the “SRI tree” sequence.

Obviously, for initializing $\mathbf{x}$, one can simply set $\mathbf{x}=\bar{\mathbf{x}}$. Regarding the initialization of $\mathbf{y}$, however, things are slightly more complicated. For maintaining Fast-PD's optimality properties, it turns out that, after setting $\mathbf{y}=\bar{\mathbf{y}}$, a slight correction still needs to be applied to $\mathbf{y}$. In particular, Fast-PD requires its initial solution $\mathbf{y}$ to satisfy condition (18), *i.e* $y_{pq}(x_p)+y_{qp}(x_q)=w_{pq}d(x_p,x_q)$, whereas $\bar{\mathbf{y}}$ satisfies $\bar{y}_{pq}(x_p)+\bar{y}_{qp}(x_q)=\bar{w}_{pq}\bar{d}(x_p,x_q)$, *i.e* condition (18) with $w_{pq}d(\cdot,\cdot)$ replaced by the pairwise potential $\bar{w}_{pq}\bar{d}(\cdot,\cdot)$ of the previous MRF. The solution for fixing that is very simple: *e.g* we can simply set $y_{pq}(x_p){+}{=}w_{pq}d(x_p,x_q)-\bar{w}_{pq}\bar{d}(x_p,x_q)$. Finally, for taking into account the possibly different singleton potentials between successive MRFs, the new heights will obviously need to be updated as $h_p(\cdot){+}{=}\mathfrak{c}_p(\cdot)-\bar{\mathfrak{c}}_p(\cdot)$, where $\bar{\mathfrak{c}}_p(\cdot)$ are the singleton potentials of the previous MRF. These are the only changes

```
$[\mathbf{x},\mathbf{y}]\leftarrow$ INIT_DUALS_PRIMALS$(\bar{\mathbf{x}},\bar{\mathbf{y}})$:
    $\mathbf{x}\leftarrow\bar{\mathbf{x}};\ \ \mathbf{y}\leftarrow\bar{\mathbf{y}};$
    $\forall pq, y_{pq}(x_p) {+}{=} w_{pq}d(x_p,x_q)-\bar{w}_{pq}\bar{d}(x_p,x_q);$
    $\forall p,\ \ h_p(\cdot) {+}{=} \mathfrak{c}_p(\cdot)-\bar{\mathfrak{c}}_p(\cdot);$
```

Fig. 20: Fast-PD's new pseudocode for dynamic MRFs.



(a) Dynamic MRFs in the case of a primal-dual algorithm

(b) Dynamic MRFs in the case of an algorithm that works in the primal domain only

Fig. 21: **(a)** The final costs primal$_{\bar{\mathbf{x}}}$, dual$_{\bar{\mathbf{y}}}$ of the previous MRF are slightly perturbed to give the initial costs primal$_{\mathbf{x}}$, dual$_{\mathbf{y}}$ of the current MRF. Therefore, "gap" (*i.e*, the initial primal-dual gap of the current MRF) will be close to "$\overline{\text{gap}}$" (*i.e*, the final primal-dual gap of the previous MRF). Since, in the case of Fast-PD, the latter is small, so will be the former, and thus few augmenting paths will need to be computed right from the start for the current MRF, thus boosting its inference. **(b)** However, in the case of an algorithm that works only in the primal domain, "$\overline{\text{gap}}$" will be large and thus the initial primal-dual of the current MRF will remain large as well (despite that only a small perturbation exists between primal$_{\bar{\mathbf{x}}}$ and primal$_{\mathbf{x}}$). Therefore, there will be no significant reduction in the number of augmenting paths for the current MRF.

needed for the case of dynamic MRFs, and thus the new pseudocode appears in Fig. 20.

As expected, the speedup provided by Fast-PD for dynamic MRFs is even greater than single MRFs. *E.g* Fig. 19(a) shows the running times per frame for the "SRI tree" image sequence. Fast-PD proves to be be more than 10 times faster than α-expansion in this case (requiring on average 0.22 secs per frame, whereas α-expansion required 2.28 secs on average). Fast-PD can thus run on about 5 frames/sec, *i.e* it can do stereo matching almost in real time for this example (in fact, if successive MRFs bear greater similarity, even much bigger speedups can be achieved). Furthermore, fig. 19(b) shows the corresponding number of augmenting paths per frame for the "SRI tree" image sequence (for both α-expansion and Fast-PD). As can be seen from that figure, a substantial reduction in the number of augmenting paths is achieved by Fast-PD, which helps that algorithm to reduce its running time.

This same behavior has been observed in all other dynamic problems that Fast-PD has been tested on as well. Intuitively, what happens is illustrated in Fig. 21(a). Fast-PD has already managed to close the gap between the costs primal$_{\bar{\mathbf{x}}}$, dual$_{\bar{\mathbf{y}}}$ of the final primal-dual solutions $\bar{\mathbf{x}}, \bar{\mathbf{y}}$ of the previous MRF. However, due to the possibly different singleton $(i.e\ \mathfrak{c}_p(\cdot))$ or pairwise $(i.e\ w_{pq}d(\cdot,\cdot))$ potentials of the current MRF,

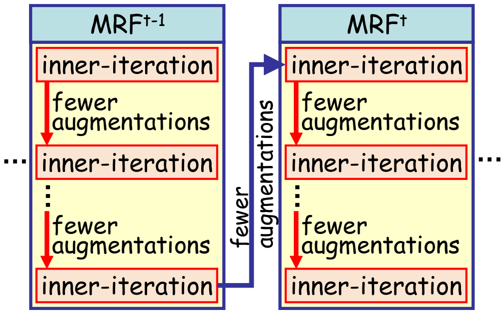


Fig. 22: Fast-PD reduces the number of augmenting paths in 2 ways: internally, *i.e* across iterations of the same MRF (see red arrows), as well as externally, *i.e* across different MRFs (see blue arrow).

these costs need to be perturbed to generate the costs $\text{primal}_{\mathbf{x}}$, $\text{dual}_{\mathbf{y}}$ for the initial solutions $\mathbf{x}, \mathbf{y}$ of the current MRF. Nevertheless, as only slight perturbations take place, the new primal-dual gap (*i.e* between $\text{primal}_{\mathbf{x}}$, $\text{dual}_{\mathbf{y}}$) will still be close to the previous gap (*i.e* between $\text{primal}_{\bar{\mathbf{x}}}$, $\text{dual}_{\bar{\mathbf{y}}}$). This means that the new initial gap will be small and, so, few augmenting paths will have to be found even from the first iteration of the current MRF, which helps a lot in improving the efficiency for doing inference on that MRF.

On the other hand, for the case of an algorithm that works only in the primal domain (*e.g*, α-expansion), the final primal-dual gap for the previous MRF will be large. Hence, despite that only a small perturbation takes place between the costs of the previous and the current primal solution (*i.e*, between $\text{primal}_{\bar{\mathbf{x}}}$ and $\text{primal}_{\mathbf{x}}$), the initial primal-dual gap for the current MRF will necessarily be large, which means that a lot of augmenting paths will have to be computed right from the first iteration of the current MRF, thus making inference slower.

Put otherwise, for the case of dynamic MRFs, Fast-PD manages to boost performance, *i.e* reduce number of augmenting paths, across two different “axes”. The first axis lies along the different inner-iterations of the same MRF (*e.g* see red arrows in Fig. 22), whereas the second axis extends across time, *i.e* across different MRFs (*e.g* see blue arrow in Fig. 22, connecting last iteration of MRF^{t-1} to first iteration of MRF^{t}).

VII. Conclusions

In conclusion, a new graph-cut based method for MRF optimization has been proposed, which is capable of handling a wide class of MRFs that are frequently encountered in computer vision. It builds upon the recently proposed primal-dual framework for MRF optimization in order to generate solutions that are approximately optimal. To this end, it can provide both theoretical (*i.e*, worst-case) guarantees, but also per-instance guarantees, with the latter being, in practice, much tighter than the former. Furthermore, despite the quality of the generated solutions, it still makes no compromise regarding its efficiency. It thus can be used for providing a significant speedup on the optimization of static MRFs, but, in addition, it can be used for boosting the performance of dynamic MRFs as well (*i.e*, MRFs that vary over time). In both cases, its efficiency comes from the fact that it exploits information related not only to the "primal" problem (*i.e* the MRF optimization problem), but also to an "LP-dual" problem. Due to all of the above, and given the ubiquity of MRFs, we strongly believe that the Fast-PD algorithm can prove to be an extremely valuable tool for many problems in computer vision in the years to come.

References

[1] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *In IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 26(9), 2004.

[2] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *In IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 23(11), 2001.

[3] O. Juan and Y. Boykov. Active graph cuts. *In Proc. Computer Vision and Pattern Recognition (CVPR)*, 2006.

[4] P. Kohli and P. H. Torr. Efficiently solving dynamic markov random fields using graph cuts. *In Proc. International Conference on Computer Vision (ICCV)*, 2005.

[5] N. Komodakis and G. Tziritas. A new framework for approximate labeling via graph-cuts. *In Proc. International Conference on Computer Vision (ICCV)*, 2005.

[6] C. Rother, S. Kumar, V. Kolmogorov, and A. Blake. Digital tapestry. *In Proc. Computer Vision and Pattern Recognition (CVPR)*, 2005.

[7] R. Szeliski, R. Zabih, D. Scharstein, O. Veksler, V. Kolmogorov, A. Agarwala, M. Tappen, and C. Rother. A comparative study of energy minimization methods for markov random fields. *In Proc. European Conference on Computer Vision (ECCV)*, 2006.

[8] V. Kolmogorov and R. Zabih. What Energy Functions can be Minimized via Graph Cuts? *In IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 26(2), 2004.

[9] N. Komodakis, G. Tziritas and N. Paragios, "Fast, Approximately Optimal Solutions to Single and Dynamic MRFs." in *CVPR*, 2007.

[10] N. Komodakis and G. Tziritas. Approximate Labeling via Graph Cuts Based on Linear Programming *In IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 29(8), 2007.

[11] 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.

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

[13] D. S. Hochbaum, *Approximation Algorithms for NP-hard problems*. PWS Publishing Company, Boston, MA, 1997.

[14] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, 1982.

[15] M. F. Tappen and W. T. Freeman, "Comparison of graph cuts with belief propagation for stereo, using identical MRF parameters." in *ICCV*, 2003, pp. 900–907.

[16] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1988.

[17] M. Wainwright, T. Jaakkola, and A. Willsky. Map estimation via agreement on (hyper)trees: message-passing and linear programming. *IEEE Trans. on Information Theory*, 2005.

[18] N. Komodakis, N. Paragios and G. Tziritas, “MRF Optimization via Dual Decomposition: Message-Passing Revisited.” in *ICCV*, 2007.

[19] J. Yedidia, W. Freeman, and Y. Weiss. Constructing free energy approximations and generalized belief propagation algorithms. *IEEE Transactions on Information Theory*, 2005.

[20] T. Meltzer, C. Yanover, and Y. Weiss, “Globally optimal solutions for energy minimization in stereo vision using reweighted belief propagation.” in *ICCV*, 2005.

[21] V. Kolmogorov. Convergent tree-reweighted message passing for energy minimization. *In IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 2006.

[22] H. Ishikawa. Exact Optimization for Markov Random Fields with Convex Priors *In IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 25(10), 2003.

[23] O. Veksler, “Efficient graph-based energy minimization methods in computer vision,” Ph.D. dissertation, Cornell, 1999.

[24] S. Roy and I. Cox, “A maximum-flow formulation of the n-camera stereo correspondence problem,” in *ICCV*, 1998.

[25] B. Glocker, N. Komodakis, N. Paragios, G. Tziritas and N. Navab, “Inter and Intra-modal Deformable Registration: Continuous Deformations Meet Efficient Optimal Linear Programming,” in *IPMI*, 2007.

[26] A. Raj, G. Singha and R. Zabih, “MRF’s forMRI’s: Bayesian Reconstruction of MR Images via Graph Cuts,” in *CVPR*, 2006.

[27] V. Kolmogorov and R. Zabih, “Multi-camera scene reconstruction via graph cuts.” in *ECCV*, 2002, pp. 82–96.

[28] R. Zabih and V. Kolmogorov, “Spatially coherent clustering using graph cuts.” in *CVPR*, 2004, pp. 437–444.

[29] J. Kim, V. Kolmogorov and R. Zabih, “Visual Correspondence Using Energy Minimization and Mutual Information,” in *ICCV*, 2003.

[30] 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.

[31] M. Kumar, P. Torr and A. Zisserman, “OBJ CUT.” in *CVPR*, 2005.

[32] V. Kwatra, A. Schodl, I. Essa, G. Turk and A. Bobick, “Graphcut Textures: Image and Video Synthesis Using Graph Cuts.” in *SIGGRAPH*, 2003.

[33] N. Komodakis and G. Tziritas, “Image Completion Using Global Optimization.” in *CVPR*, 2006.

[34] S. Roth and M. Black, “Fields of Experts: A Framework for Learning Image Priors.” in *CVPR*, 2005.

[35] P. Felzenszwalb and D. Huttenlocher Pictorial Structures for Object Recognition *In IJCV*, 61(1), 2005.

[36] B. Frey, *Graphical models for machine learning and digital communication*. MIT Press, 1998.

[37] M. Jordan, *Learning in graphical models*. MIT Press, 1999.

[38] J. Schnabel, D. Rueckert, M. Quist, J. Blackall, A. Castellano-Smith, T. Hartkens, G. Penney, W. Hall, H. Liu, C. Truwit, F. Gerritsen, D. Hill and D. Hawkes, “A Generic Framework for Non-rigid Registration Based on Non-uniform Multi-level Free-Form Deformations.” in *MICCAI*, 2001.

[39] T. Werner. A Linear Programming Approach to Max-sum Problem: A Review. *In IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 2007.

[40] M. I. Schlesinger. Syntactic Analysis of Two-dimensional Visual Signals in Noisy Conditions. *In Kybernetica*, 1976.

[41] V. Kolmogorov, “A note on the primal-dual method for the semi-metric labeling problem”, *Technical Report*, 2007.

[42] B. Glocker, N. Komodakis, N. Paragios, G. Tziritas, and N. Navab. Dense Image registration through MRFs and Efficient Linear Programming. *Medical Image Analysis*, 2008.

[43] B. Glocker, N. Paragios, N. Komodakis, G. Tziritas, and N. Navab., “Optical Flow Estimation with Uncertainties through

Dynamic MRFs." in *CVPR*, 2008.

[44] N. Komodakis, and N. Paragios, "Beyond Loose LP-relaxations: Optimizing MRFs by Repairing Cycles." in *ECCV*, 2008.

Appendix A: Proof of Theorem 4 about the optimality of Fast-PD's solutions

The purpose of this section is to provide the proof for Theorem 4, which certifies that the solutions estimated by the Fast-PD algorithm have guaranteed optimality properties. But before that, the following 3 lemmas need to be proved:

Lemma A.1

During a c-iteration, the following inequalities hold true exactly after UPDATE_DUALS_PRIMALS*:*

$$y'_{pq}(c) \leq y_{pq}(c) + \mathrm{cap}_{pq} \tag{32}$$

$$y'_{qp}(c) \leq y_{qp}(c) + \mathrm{cap}_{qp} \tag{33}$$

Proof. An intuitive proof comes from the fact that flows f_{pq} and f_{qp} represent the increase of the balance variables $y_{pq}(c)$ and $y_{qp}(c)$ respectively during UPDATE_DUALS_PRIMALS. Since it is always true that:

$$f_{pq} \leq \mathrm{cap}_{pq} \ ,$$
$$f_{qp} \leq \mathrm{cap}_{qp} \ ,$$

the lemma then follows directly. □

Lemma A.2 *During a c-iteration, the following entailments hold true:*

$$\mathrm{load}_{pq}(c, \bar{x}_q) \leq w_{pq} d(c, \bar{x}_q) \Rightarrow \mathrm{load}'_{pq}(c, \bar{x}_q) \leq w_{pq} d(c, \bar{x}_q) \ , \tag{34}$$

$$\mathrm{load}_{pq}(\bar{x}_p, c) \leq w_{pq} d(\bar{x}_p, c) \Rightarrow \mathrm{load}'_{pq}(\bar{x}_p, c) \leq w_{pq} d(\bar{x}_p, c) \ , \tag{35}$$

where $\bar{\mathbf{x}}$ can be any labeling which is a c-expansion of the primal solution $\mathbf{x}$ at the start of the current c-iteration. (In the above entailments, quantities $\mathrm{load}_{pq}(c, \bar{x}_q)$, $\mathrm{load}_{pq}(\bar{x}_p, c)$ are supposed to have been estimated using the value of the balance variables exactly after PREEDIT_DUALS*).*

Proof. If $\bar{x}_q = c$ then (34) is trivial to prove. We may therefore assume that $\bar{x}_q = x_q \neq c$ (since $\bar{\mathbf{x}}$ is a c-expansion of $\mathbf{x}$). So, in order to prove (34), let us then also assume that:

$$\mathrm{load}_{pq}(c, x_q) \leq w_{pq} d(c, x_q) \tag{36}$$

But then, by combining Lemma A.1 with the definition of capacity cap_{pq} in (27), we get:

$$\begin{aligned} y'_{pq}(c) \overset{(32)}{\leq} y_{pq}(c) + \mathrm{cap}_{pq} &\overset{(27)}{=} y_{pq}(c) + [w_{pq}d(c,x_q) - \mathrm{load}_{pq}(c,x_q)]^+ \\ &\overset{(36)}{=} y_{pq}(c) + w_{pq}d(c,x_q) - \mathrm{load}_{pq}(c,x_q) \\ &= w_{pq}d(c,x_q) - y_{qp}(x_q) \overset{x_q \neq c}{=} w_{pq}d(c,x_q) - y'_{qp}(x_q) \end{aligned}$$

which thus proves (34). The proof for (35) proceeds similarly. □

Lemma A.3 *At the last c-iteration of the Fast-PD algorithm, the following inequalities hold (for any p, q):*

$$\mathrm{load}'_{pq}(c, x'_q) \leq w_{pq}d_{\max} \tag{37}$$

$$\mathrm{load}'_{pq}(x'_p, c) \leq w_{pq}d_{\max} \tag{38}$$

Proof. The lemma is trivial if either $c = x'_p$ or $c = x'_q$, and so we will hereafter assume that $c \neq x'_p$ and $c \neq x'_q$. Furthermore, since this is the last c-iteration, no label change takes place, and so:

$$x'_p = x_p, \quad x'_q = x_q. \tag{39}$$

CASE 1: If the following two inequalities hold true:

$$\mathrm{load}_{pq}(c, x_q) \leq w_{pq}d(c, x_q)\ , \tag{40}$$

$$\mathrm{load}_{pq}(x_p, c) \leq w_{pq}d(x_p, c)\ , \tag{41}$$

then the lemma follows directly from Lemma A.2.

CASE 2: It thus remains to consider the case where at least one of the inequalities (40), (41) is violated. Then (and only then), PREEDIT_DUALS (by definition) will adjust $y_{pq}(c)$ so that:

$$\mathrm{load}_{pq}(c, x_q) = w_{pq}d(c, x_q) \tag{42}$$

Hence, condition (40) will be restored after the adjustment. We may then assume that (41) will remain violated after the adjustment (or else we would fall back to case 1), *i.e* we may assume that:

$$\mathrm{load}_{pq}(x_p, c) > w_{pq}d(x_p, c) \tag{43}$$

Based on (42), (43) and the definition of capacities in (27), (28), it then results that $\text{cap}_{pq} = \text{cap}_{qp} = 0$. This implies that $y'_{pq}(c) = y_{pq}(c)$, and it is then easy to show that:

$$\text{load}'_{pq}(c, x_q) = \text{load}_{pq}(c, x_q) \tag{44}$$

$$\text{load}'_{pq}(x_p, c) = \text{load}_{pq}(x_p, c) \tag{45}$$

But then:

$$\text{load}'_{pq}(c, x_q) \overset{(44)}{=} \text{load}_{pq}(c, x_q) \overset{(42)}{=} w_{pq} d(c, x_q) \leq w_{pq} d_{\max} \tag{46}$$

and also:

$$\text{load}'_{pq}(x_p, c) \overset{(45)}{=} \text{load}_{pq}(x_p, c) = [\text{load}_{pq}(x_p, c) + \text{load}_{pq}(c, x_q)] - \text{load}_{pq}(c, x_q) \tag{47}$$

$$= \text{load}_{pq}(x_p, x_q) - \text{load}_{pq}(c, x_q) \tag{48}$$

$$\overset{(18),(42)}{=} w_{pq} d(x_p, x_q) - w_{pq} d(c, x_q) \leq w_{pq} d_{\max}, \tag{49}$$

with equality (48) being true due to the identity $\text{load}_{pq}(x_p, c) + \text{load}_{pq}(c, x_q) = \text{load}_{pq}(x_p, x_q)$. □

We may now proceed to prove Theorem 4, which (as already mentioned) forms the main goal of this section.

Proof for Theorem 4. To complete the proof of this theorem, we need to show that each one of the complementary slackness conditions (17)-(19) will hold true by the time Fast-PD terminates:

Condition (18)**:** As already explained in section IV, the UPDATE_DUALS_PRIMALS routine can restore condition (18) for most pairs (p, q) during any inner-iteration. However, even if there do exist pairs that violate this condition after UPDATE_DUALS_PRIMALS, then the POSTEDIT_DUALS routine can, by definition, always restore condition (18) for them.

Condition (19)**:** Based on Lemma A.3, it follows that, given any label a, the following inequality will hold true after the last a-iteration:

$$\text{load}_{pq}(a, x_q) \leq w_{pq} d_{\max}. \tag{50}$$

Similarly, given any label b, the following inequality will also hold true after the last b-iteration:

$$\text{load}_{pq}(x_p, b) \leq w_{pq} d_{\max}. \tag{51}$$

Combining these inequalities with the identity:

$$\mathrm{load}_{pq}(a,b)+\mathrm{load}_{pq}(x_p,x_q)=\mathrm{load}_{pq}(a,x_q)+\mathrm{load}_{pq}(x_p,b), \tag{52}$$

we get that:

$$\begin{aligned}\mathrm{load}_{pq}(a,b) &= [\mathrm{load}_{pq}(a,b)+\mathrm{load}_{pq}(x_p,x_q)]-\mathrm{load}_{pq}(x_p,x_q)\\ &\overset{(52)}{=} [\mathrm{load}_{pq}(a,x_q)+\mathrm{load}_{pq}(x_p,b)]-\mathrm{load}_{pq}(x_p,x_q)\\ &\overset{(50),\ (51)}{\leq} 2w_{pq}d_{\max}-\mathrm{load}_{pq}(x_p,x_q),\end{aligned}$$

and then condition (19) follows trivially, since $\mathrm{load}_{pq}(x_p,x_q)=d(x_p,x_q)\geq 0$ by (18).

Condition (17)**:** It turns out that the UPDATE_DUALS_PRIMALS routine can finally ensure condition (17) due to the way that the exterior capacities of graph $\mathcal{G}^c$ are defined. Since Fast-PD uses the same definition as PD3$_a$ for these capacities, the corresponding proof (that has been used for the case of the PD3$_a$ algorithm) in [5] applies here as well. □

Appendix B: Proof of Theorem 5 about the equivalence of Fast-PD and α-expansion in the case of a metric distance function $d(\cdot,\cdot)$

In this section, we will provide the proof for Theorem 5, which shows that when distance $d(\cdot,\cdot)$ is a metric, then Fast-PD can compute exactly the same solution as the α-expansion algorithm. To this end, we will make use of the following two lemmas:

Lemma B.1 *Let us define:*

$$\text{primal}(\mathbf{x}) \equiv \text{MRF energy of labeling } \mathbf{x}\ ,$$

and let also $\mathbf{x}$ *be any primal solution generated during an inner-iteration of the Fast-PD algorithm. It then holds that:*

$$\text{primal}(\mathbf{x}) = \sum_p h_p(x_p) \tag{53}$$

Proof.

$$\begin{aligned}
\text{primal}(\mathbf{x}) &\overset{(??)}{=} \sum_p \mathfrak{c}_p(x_p) + \sum_{pq\in\ \mathcal{E}} w_{pq} d(x_p,x_q) \\
&\overset{(18)}{=} \sum_p \mathfrak{c}_p(x_p) + \sum_{pq\in\ \mathcal{E}} \text{load}(x_p,x_q) \overset{(24)}{=} \sum_p \mathfrak{c}_p(x_p) + \sum_{pq\in\ \mathcal{E}} \big(y_{pq}(x_p) + y_{qp}(x_q)\big) \\
&= \sum_p \mathfrak{c}_p(x_p) + \sum_p \sum_{q:pq\in\ \mathcal{E}} y_{pq}(x_p) = \sum_p \big(\mathfrak{c}_p(x_p) + \sum_{q:pq\in\ \mathcal{E}} y_{pq}(x_p)\big) \overset{(23)}{=} \sum_p h_p(x_p)
\end{aligned}$$

□

Lemma B.2 *Let the distance function* $d(\cdot,\cdot)$ *be a metric. Let* $\mathbf{x}$ *be the primal solution at the start of the current* c*-iteration, and let also* $\bar{\mathbf{x}}$ *be any solution which coincides with a* c*-expansion of solution* $\mathbf{x}$*. It will then hold that:*

$$\text{load}'_{pq}(\bar{x}_p,\bar{x}_q) \le w_{pq} d(\bar{x}_p,\bar{x}_q) \tag{54}$$

Proof. If either $\bar{x}_p = \bar{x}_q = c$ or $\bar{x}_p = x_p$, $\bar{x}_q = x_q$, the lemma is trivial to prove. So let us assume that $\bar{x}_p = x_p$, $\bar{x}_q = c$ (the case $\bar{x}_p = c$, $\bar{x}_q = x_q$ can be handled similarly). In this case, we need to show that:

$$\text{load}'_{pq}(x_p,c) \le w_{pq} d(x_p,c) \tag{55}$$

Due to entailment (35) in Lemma A.2, it then suffices to show that the following condition will hold true after PREEDIT_DUALS:

$$\text{load}_{pq}(x_p, c) \le w_{pq} d(x_p, c). \tag{56}$$

Regarding inequality (56), this will always hold if PREEDIT_DUALS has to apply no adjustment to $y_{pq}(c)$ (this results from the definition of PREEDIT_DUALS). However, even if PREEDIT_DUALS must adjust the value of $y_{pq}(c)$, inequality (56) will still hold true, provided that $d(\cdot,\cdot)$ is a metric.

To see that, it suffices to observe that after the adjustment made by PREEDIT_DUALS, it will then hold:

$$\text{load}_{pq}(c, x_q) = w_{pq} d(c, x_q) \tag{57}$$

and so:

$$\begin{aligned}\text{load}_{pq}(x_p, c) &= [\text{load}_{pq}(x_p, c) + \text{load}_{pq}(c, x_q)] - \text{load}_{pq}(c, x_q)\\ &= \text{load}_{pq}(x_p, x_q) - \text{load}_{pq}(c, x_q) \overset{(18),(57)}{=} w_{pq} d(x_p, x_q) - w_{pq} d(c, x_q) \le w_{pq} d(x_p, c)\ ,\end{aligned}$$

where the last inequality holds due to that $d(\cdot,\cdot)$ is a metric and thus has to satisfy the triangle inequality.

□

We may now proceed to the main goal of this section, which is the proof of Theorem 5.

Proof for Theorem 5. Let $\mathbf{x}$ be the primal solution at the start of the current c-iteration, let $\mathbf{x}'$ be the solution selected by Fast-PD at the end of the current c-iteration, and let also $\bar{\mathbf{x}}$ be any solution which coincides with a c-expansion of solution $\mathbf{x}$.

To prove the theorem, we need to show that:

$$\text{primal}(\mathbf{x}') \le \text{primal}(\bar{\mathbf{x}}) \tag{58}$$

To this end, it suffices to show that the following conditions hold true:

$$\text{primal}(\mathbf{x}') = \sum_p h'_p(x'_p) \tag{59}$$

$$\sum_p h'_p(x'_p) \le \sum_p h'_p(\bar{x}_p) \tag{60}$$

$$\sum_p h'_p(\bar{x}_p) \le \text{primal}(\bar{\mathbf{x}}) \tag{61}$$

Regarding equation (59), this follows directly by applying Lemma B.1 to the primal solution $\mathbf{x}'$ generated by the Fast-PD algorithm.

To prove inequality (60), one can first show that $h'_p(x'_p) = \min\{h'_p(x_p), h'_p(c)\}$. In addition to that, it will also hold either $\bar{x}_p = x_p$ or $\bar{x}_p = c$ (since $\bar{\mathbf{x}}$ is a c-expansion of $\mathbf{x}$). By combining these facts, it then results that $h'_p(x'_p) \le h'_p(\bar{x}_p)$, and thus (60) follows directly.

Finally, inequality (59) will hold true because:

$$\begin{aligned}
\mathrm{primal}(\bar{\mathbf{x}}) &= \sum_p \mathfrak{c}_p(\bar{x}_p) + \sum_{pq \in \mathcal{E}} w_{pq} d(\bar{x}_p, \bar{x}_q) \overset{(54)}{\geq} \sum_p \mathfrak{c}_p(\bar{x}_p) + \sum_{pq \in \mathcal{E}} \mathrm{load}'(\bar{x}_p, \bar{x}_q) \\
&= \sum_p \mathfrak{c}_p(\bar{x}_p) + \sum_{pq \in \mathcal{E}} \big(y'_{pq}(\bar{x}_p) + y'_{qp}(\bar{x}_q) \big) = \sum_p \mathfrak{c}_p(\bar{x}_p) + \sum_p \sum_{q: pq \in \mathcal{E}} y'_{pq}(\bar{x}_p) \\
&= \sum_p \big(\mathfrak{c}_p(\bar{x}_p) + \sum_{q: pq \in \mathcal{E}} y'_{pq}(\bar{x}_p) \big) \overset{(23)}{=} \sum_p h'_p(\bar{x}_p)
\end{aligned}$$

□

APPENDIX C: PROOF OF THEOREM 7

Theorem 7. *For Fast-PD, the primal-dual gap at the current inner-iteration forms an approximate upper bound for the number of augmenting paths at each iteration thereafter.*

Proof. The same dual linear program as in [5] has been used, and so the cost of a dual solution is defined as:

$$\text{dual cost} = \sum_p \min_{a \in L} h_p(a) \ , \tag{62}$$

which implies that:

$$\text{dual cost} \leq \sum_p \min(h_p(c), h_p(x_p)) \tag{63}$$

Furthermore, in the case of the Fast-PD algorithm, it can be shown that the following equality will hold before the start of max-flow at an inner-iteration (see lemma B.1):

$$\text{primal cost} = \sum_p h_p(x_p) \tag{64}$$

Based on (63), (64), the following inequality then results:

$$\begin{aligned} \text{primal dual gap} = \text{primal cost} - \text{dual cost} &\geq \sum_p h_p(x_p) - \sum_p \min(h_p(c), h_p(x_p)) \\ &= \sum_p [h_p(x_p) - h_p(c)]^+ = \sum_p \text{cap}_{\mathfrak{s}p}. \end{aligned} \tag{65}$$

But the quantity $\sum_p \text{cap}_{\mathfrak{s}p}$ obviously forms an upper-bound on the maximum flow during a c-iteration, which, in turn, upper-bounds the number of augmenting paths (assuming integral flows). In addition to that, the upper bound defined by $\sum_p \text{cap}_{\mathfrak{s}p}$ will not increase during any of the next c-iterations (which means that the number of augmentations will keep decreasing over time), and so the current primal-dual gap will be an approximate upper bound for the number of augmentations of the next c-iterations as well.

The fact that the upper bound $\sum_p \text{cap}_{\mathfrak{s}p} = \sum_p [h_p(x_p) - h_p(c)]^+$ will not increase during any of the next iterations may be justified by that any of the terms $[h_p(x_p) - h_p(c)]^+$ can increase only during either PREEDIT_DUALS or POSTEDIT_DUALS (it is easy to show that UPDATE_DUALS_PRIMALS may only decrease the value of these terms). However, both PREEDIT_DUALS and POSTEDIT_DUALS modify the height variables $h_p(\cdot)$ only in very rare occasions during the execution of Fast-PD (*e.g* if $d(\cdot, \cdot)$ is a metric, one may prove that none of the height variables need to be altered by POSTEDIT_DUALS). Hence, the terms $[h_p(x_p) - h_p(c)]^+$ will typically not be altered by these routines (or they will be altered by a negligible amount at most), and so only UPDATE_DUALS_PRIMALS may modify these terms, thus decreasing their values.

□