Graph-based Simplex Method for Pairwise Energy Minimization with Binary Variables

Daniel Průša
Center for Machine Perception, Faculty of Electrical Engineering, Czech Technical University
Karlovo náměstí 13, 121 35 Prague, Czech Republic
prusapa1@fel.cvut.cz

Abstract

We show how the simplex algorithm can be tailored to the linear programming relaxation of pairwise energy minimization with binary variables. A special structure formed by basic and nonbasic variables in each stage of the algorithm is identified and utilized to perform the whole iterative process combinatorially over the input energy minimization graph rather than algebraically over the simplex tableau. This leads to a new efficient solver. We demonstrate that for some computer vision instances it performs even better than methods reducing binary energy minimization to finding maximum flow in a network.

1. Introduction

Energy minimization is a well known NP-hard combinatorial problem which arises in MAP inference in graphical models [16]. It is of great importance in low-level computer vision. A lot of effort has been spent by researchers to invent methods finding exact or approximate solutions.

Pairwise energy minimization with binary variables has a prominent role. It is easily expressible as quadratic pseudo-boolean optimization (QPBO) [1, 8]. Max-flow/min-cut algorithms can be applied to find a partial optimal solution, with some variables undecided. This shrinks the size of the input problem. The undecided variables can be further resolved by applying *e.g.* the *probing* technique [13] or, in general, by branch and bound. A complete optimal solution is always returned by QPBO for *submodular* instances. This is further utilized in some cases of energy minimization with general variables. Every multi-label energy minimization reduces to the binary one [6, 14], preserving the submodularity property. Further, solving submodular binary instances is a crucial part of approximate minimizations [3].

Since binary energy minimization is cast as a max-flow/min-cut problem, the key prerequisite for solving it efficiently has been so far to come up with a max-flow algorithm working well on vision instances. The popular algorithm by Boykov and Kolmogorov [2] fulfills this role. An empirical comparison of max-flow algorithms recently done by Verma and Batra [15] reveals that some other contemporary implementations are more suitable for instances with dense graphs.

In this paper we introduce a different principle to solve the binary case. We do not perform any translation to max-flow. Our starting point is the linear programming (LP) relaxation of the problem [17]. For binary variables, it is known to be half-integral, *i.e.*, all components of each optimal solution are in $\{0, \frac{1}{2}, 1\}$. Moreover, the solution coincides with the result of QPBO – value $\frac{1}{2}$ indicates undecided variables [1]. We show that the simplex method solving the LP relaxation can be turned into a very efficient algorithm, performed purely over the input energy minimization graph. Special versions of the simplex method with similar properties have already been proposed for *transportation*, *assignment* and *minimum cost-flow* problems [4]. They are known as the *network simplex* algorithms. There is even a customization for the maximum flow problem [5], though it does not figure among the leading implementations.

The proposed algorithm has practical benefit. Our experiments demonstrate there are vision instances, where the algorithm performs better than max-flow based solvers. Besides that, it allows to study the behavior of the simplex method on large-scale data and gives a hope for a generalization to multi-label problems. And finally, its impact may extend beyond the scope of energy minimization as it can be applied to min-cut which is expressible as submodular binary energy minimization. Since the formulation and the underlying structure is different than in the case of the network simplex algorithm for max-flow, new ideas for solving min-cut/max-flow may emerge.

We assume the reader is familiar with the simplex method, as presented *e.g.* in [4]. Knowledge of notions like *basis*, *basic variable*, *basic feasible solution*, *pivoting rule* or *minimum ratio test* is essential for understanding the text.

2. Energy minimization and its LP relaxation

The task of pairwise energy minimization is to compute

$$\min_{\mathbf{k}\in K^V}\left[\sum_{u\in V}\theta_u(k_u)+\sum_{\{u,v\}\in E}\theta_{uv}(k_u,k_v)\right] \tag{1}$$

where V is a finite set of *objects*, $E\subset\binom{V}{2}$ is a set of *object pairs* (*i.e.*, (V,E) is an undirected graph), K is a finite set of *labels*, and the functions $\theta_u\colon K\to\mathbb{R}$ and $\theta_{uv}\colon K\times K\to\mathbb{R}$ are unary and pairwise *interactions*. We adopt that $\theta_{uv}(k,\ell)=\theta_{vu}(\ell,k)$ and refer to the values of θ_u and θ_{uv} as *potentials*. We shortly write $\theta_u(k)$ as $\theta_{u;k}$ and $\theta_{uv}(k,\ell)$ as $\theta_{uv;k\ell}$. The potentials together form a vector $\boldsymbol{\theta}\in\mathbb{R}^I$ with

$$\begin{aligned}I=\{\,(u;k)\mid u\in V,\ k\in K\,\}\cup\\ \{\,(uv;k\ell)\mid\{u,v\}\in E;\ k,\ell\in K\,\}.\end{aligned} \tag{2}$$

Throughout the paper, we consider energy minimization with two labels, so K is fixed as $\{0,1\}$. An instance of problem (1) is thus fully defined by a tuple $(V,E,\boldsymbol{\theta})$. Its linear programming relaxation reads as

$$\operatorname*{argmin}_{\mathbf{x}\in\Lambda}\langle\boldsymbol{\theta},\mathbf{x}\rangle. \tag{3}$$

Here we optimize over polytope Λ which consists of vectors $\mathbf{x}\in\mathbb{R}^I$ that satisfy the constraints

$$\sum_{\ell\in K}x_{uv;k\ell}=x_{u;k},\quad u\in V,\ v\in N_u,\ k\in K \tag{4a}$$

$$\sum_{k\in K}x_{u;k}=1,\qquad u\in V \tag{4b}$$

$$\mathbf{x}\geq\mathbf{0}, \tag{4c}$$

where $N_u=\{\,v\mid\{u,v\}\in E\,\}$ is the set of neighbors of object u. We again adopt $x_{uv;k\ell}=x_{vu;\ell k}$.

3. Applying simplex method – preparation

We work with the variant of the simplex method which assumes the input linear program to be in the *standard form*:

$$\min\{\langle\boldsymbol{\theta},\mathbf{x}\rangle\mid\mathbf{Ax}=\mathbf{b},\ \mathbf{x}\geq\mathbf{0}\} \tag{5}$$

where $\mathbf{A}\in\mathbb{R}^{m\times n}$, $m\leq n$, $\operatorname{rank}(\mathbf{A})=m$, $\mathbf{b}\in\mathbb{R}^m$, and $\boldsymbol{\theta}\in\mathbb{R}^n$.

The LP relaxation of an energy minimization given by $(V,E,\boldsymbol{\theta})$ induces overall $n=2|V|+4|E|$ variables and $m=|V|+3|E|$ linearly independent equations [9].

For a given *basis* $\mathcal{B}\subset I$, Figure 1 shows how we visualize *basic* and *nonbasic* LP relaxation variables. The scheme includes the input graph (V,E). The variables form an underlying "microstructure". Nonbasic variables are colored

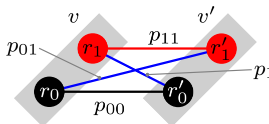

Figure 1. An LP relaxation diagram. The energy minimization graph consists of objects v, v' and object pair $\{v,v'\}$. LP relaxation variables are denoted as $r_k=x_{v;k}$, $r'_k=x_{v';k}$, and $p_{k\ell}=x_{vv';k\ell}$. Nonbasic variables are red, basic variables are black or blue. The selected basis is $\mathcal{B}=\{(v;0),(vv';00),(v';0),(vv';01),(vv';10)\}$.

in red. They always attain zero value in the induced feasible solution. Basic variables attaining nonzero or zero value are black or blue, respectively. Each such scheme is called the *LP relaxation diagram*. In accordance with the diagram appearance, every unary or pairwise LP relaxation variable is simply called a *node* (variable) or *edge* (variable), respectively.

The set of basic variables represented in Figure 1 results in the following *simplex tableau*.

	r_0	r_1	r'_0	r'_1	p_{00}	p_{01}	p_{10}	p_{11}	
	θ_0	θ_1	θ'_0	θ'_1	θ_{00}	θ_{01}	θ_{10}	θ_{11}	0
r_0	1	1	0	0	0	0	0	0	1
r'_0	0	0	1	1	0	0	0	0	1
p_{00}	0	1	0	1	1	0	0	-1	1
p_{01}	0	0	0	-1	0	1	0	1	0
p_{10}	0	-1	0	0	0	0	1	1	0

Constraints (4a), (4b) give 6 linear equations, but only 5 of them are linearly independent. Each row of the tableau is a linear combination of the constraints expressing a basic variable. The induced basic solution is *feasible* since all elements in the rightmost column are nonnegative. The simplex algorithm could be launched if the row with potentials (*objective costs*) is adjusted to contain zeros in all basic columns.

Performance of the standard, tableau-based simplex algorithm is influenced by two factors.

- The number of performed iterations. It tends to be steadily $\mathcal{O}(n)$ in practical applications [4]. Some pathological LP instances result in an exponential amount of iterations. On the other hand, polynomial upper bounds were proved *e.g.* for the assignment problem mentioned in the introduction.
- Memory and running time required to represent and update the simplex tableau. This is what makes the method computationally infeasible for large-scale instances, quadratic time and space is required. A usage of a sparse matrix might be helpful, however, the number of nonzero elements keeps growing and the

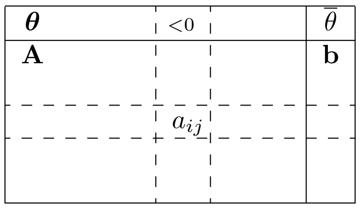

Figure 2. Having chosen pivot a_{ij}, only nonzero elements in the i-th row and j-th column are needed to update $\boldsymbol{\theta}$, $\overline{\theta}$ and $\mathbf{b}$ properly.

update is still a time-consuming operation. The problems are partially addressed by the so called *revised simplex* method [4], but still, the general algorithm is not fast enough to be able to compete with max-flow based QPBO solvers.

The key concept leading to an efficient algorithm is explained in Figure 2. Each iteration of the simplex algorithm updates cost vector $\boldsymbol{\theta}$ and the objective function value $\overline{\theta}$ based on the pivotal row, while the right-hand sides vector $\mathbf{b}$ is updated based on the pivotal column. Moreover, only nonzero values have an impact. We show that the LP relaxation diagram enables a cheap retrieval of all such nonzero elements. It is thus enough to maintain only $\boldsymbol{\theta}$, $\overline{\theta}$ and $\mathbf{b}$.

4. Structure of basis

It is a well known fact that each basic variable is expressible as a linear function of nonbasic variables. Moreover, it is expressed in a unique way.

Theorem 1. *Let $\mathcal{B} \subset I$ be a basis of (3) and let $\mathcal{N} = I \setminus \mathcal{B}$ be the set of nonbasic indices. There are unique coefficients $b_i, a_{ij} \in \mathbb{R}$ ($i \in \mathcal{B}, j \in \mathcal{N}$) such that each feasible vector $\mathbf{x} \in \Lambda$ satisfies*

$$x_i = b_i - \sum_{j \in \mathcal{N}} a_{ij} x_j, \quad \forall i \in \mathcal{B}. \tag{6}$$

Proof. Consider a linear program with constraint equations $\mathbf{C}\mathbf{x} = \mathbf{c}$ where $\mathbf{C} \in \mathbb{R}^{m \times n}$, $\mathbf{c} \in \mathbb{R}^m$ and $\operatorname{rank}(\mathbf{C}) = m$. It can be written as $\mathbf{B}\mathbf{x}_{\mathcal{B}} + \mathbf{N}\mathbf{x}_{\mathcal{N}} = \mathbf{c}$ where $\mathbf{B}$ is an invertible matrix and $\mathbf{x}_{\mathcal{B}}$, $\mathbf{x}_{\mathcal{N}}$ are vectors of basic and nonbasic variables, respectively [4]. This implies that $\mathbf{x}_{\mathcal{B}} = \mathbf{B}^{-1}\mathbf{c} - \mathbf{B}^{-1}\mathbf{N}\mathbf{x}_{\mathcal{N}}$. Assume that also $\mathbf{x}_{\mathcal{B}} = \mathbf{d} - \mathbf{D}\mathbf{x}_{\mathcal{N}}$ for some $\mathbf{D} \in \mathbb{R}^{m \times (n-m)}$, $\mathbf{d} \in \mathbb{R}^m$. Setting nonbasic variables to zeros gives the (only) basic solution corresponding to basis $\mathcal{B}$, hence $\mathbf{d} = \mathbf{B}^{-1}\mathbf{c}$. This further implies that $\mathbf{D}\mathbf{x}_{\mathcal{N}} = \mathbf{B}^{-1}\mathbf{N}\mathbf{x}_{\mathcal{N}}$ for all $\mathbf{x}_{\mathcal{N}} \in \mathbb{R}^{n-m}$, which holds only if $\mathbf{D} = \mathbf{B}^{-1}\mathbf{N}$. □

Note that coefficients a_{ij} and b_i in Theorem 1 correspond to elements in the simplex tableau composed for basis $\mathcal{B}$. In what follows, we examine how they relate to the structure of the LP relaxation diagram.

We say that a basic variable x_i *depends* on a nonbasic variable x_j if $a_{ij} \neq 0$. Two useful corollaries can be obtained from Theorem 1.

Corollary 2. *Let $\mathcal{B} \subset I$ be a basis of (3), $\mathcal{N} = I \setminus \mathcal{B}$ and $\mathbf{y}, \mathbf{z} \in \Lambda$. If $y_i = z_i$ for all $i \in \mathcal{N}$, then $\mathbf{y} = \mathbf{z}$.*

Corollary 3. *Let $\mathcal{B} \subset I$ be a basis of (3), $\mathcal{N} = I \setminus \mathcal{B}$ and $\mathbf{y}, \mathbf{z} \in \Lambda$. Let there be $k \in \mathcal{N}$ such that $y_k \neq z_k$ and $y_i = z_i$ for all $i \in \mathcal{N} \setminus \{k\}$. For $j \in \mathcal{B}$, if x_j is a basic variable in (6) which depends on nonbasic variable x_k, then $y_j \neq z_j$.*

The first corollary states that a feasible vector is fully determined by its nonbasic components. The second corollary states that if two feasible vectors have the same nonbasic components except one, then they differ in all basic components which depend on the distinctive nonbasic component.

Given a basis $\mathcal{B}$, two special elements in (V, E) are important for expressing basic node variables in the form (6).

- An object $u \in V$ is called a *dependency root* if each basic node variable $x_{u;k}$ ($k \in \{0, 1\}$) depends only on the other node $x_{u;1-k}$ and/or on edge variables in object pairs adjacent to u. Two possible configurations forming a dependency root are depicted in Figure 3.

- An object pair $\{u, v\} \in E$ is called a *dependency object pair* if there are exactly two nonbasic edges $x_{uv;k\ell}$ which are either "parallel" or "intersecting" as shown in Figure 4.

Note that Figure 3(b) and Figure 4 show only snippets of an LP relaxation diagram, not standalone diagrams with a valid basis.

Define a *dependency graph* $D(V, E, \mathcal{B}) = (V, E')$ as the subgraph of (V, E) where E' consists of all dependency object pairs in E. Moreover, for $u \in V$, define $D_u(V, E, \mathcal{B})$ as the connected component of $D(V, E, \mathcal{B})$ containing u. An example of a LP relaxation diagram and the induced dependency graph is depicted in Figure 6. The following theorem gives its characterization.

Theorem 4. *Let $(V, E, \boldsymbol{\theta})$ be an instance of binary energy minimization and let $\mathcal{B}$ be a basis of its LP relaxation. Then, $D(V, E, \mathcal{B})$ has the following structure.*

- *Each component has at most one cycle,*

- *if a component is a tree, it contains exactly one dependency root, and*

- *if a component has a cycle, it does not contain any dependency root.*

Moreover, each basic node in an object $u \in V$ depends only on nonbasic variables located in the following elements:

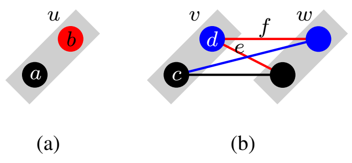

Figure 3. Objects u and v are dependency roots since (a) $a = 1 - b$, (b) $d = e + f$ and $c = 1 - e - f$.

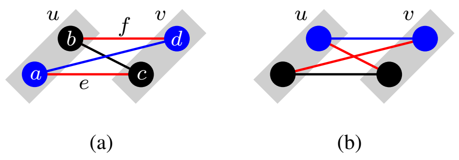

Figure 4. Two (a) parallel or (b) intersecting nonbasic edges allow to delegate expressing basic nodes in u to expressing basic nodes in v (and vice versa). For example, $a = e - f + d$ and $b = f - e + c$.

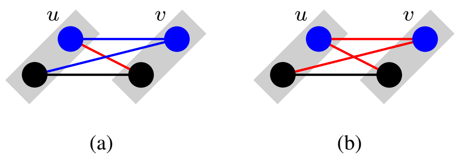

Figure 5. (a) Objects u and v do not depend on a single nonbasic edge in $\{u, v\}$. (b) Three nonbasic edges in $\{u, v\}$ induce dependency roots u, v.

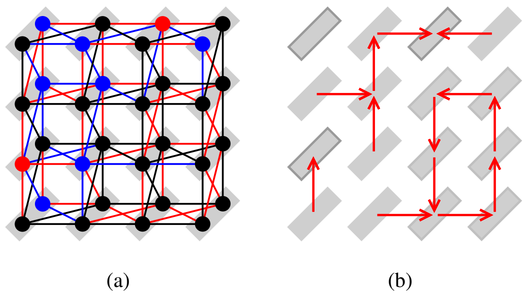

Figure 6. An example of (a) an LP relaxation diagram and (b) its dependency graph. Dependency object pairs are displayed as directed edges oriented towards the root or cycle of the component. Edges in a cycle follow one of the orientations which closes a loop. Dependency roots as well as objects in cycles are highlighted.

- *objects and object pairs forming the path from u to the dependency root or the cycle of $D_u(V, E, \mathcal{B})$,*
- *objects and object pairs of the cycle of $D_u(V, E, \mathcal{B})$, and*
- *the object pair containing nonbasic edges establishing the root of $D_u(V, E, \mathcal{B})$ (refers to Figure 3(b)).*

Proof. We list all locally admissible configurations of basic and nonbasic variables (up to permutations). After that, we derive how a basic node is expressed in the form (6).

As the first step, observe there is at most one nonbasic node within one object. Since nonbasic nodes attain zero in feasible solutions, two nonbasic nodes do not fulfill constraint (4b). In addition, each object pair $\{u, v\}$ contains one, two or three nonbasic edges. This is proved by contradiction. Let all four edges $x_{uv;ij}$, $i, j \in \{0, 1\}$ be nonbasic (and thus of zero value). Constraint (4a) forces $x_{u;0} = x_{u;1} = 0$ which again does not fulfill (4b). On the other side, let all four $x_{uv;ij}$ be basic. Take a feasible solution $\mathbf{y}$ where $y_{uv;00} = y_{uv;11} = 1/2$ and $y_{uv;01} = y_{uv;10} = 0$. Create vector $\mathbf{z}$ from $\mathbf{y}$ as follows: set $z_{uv;00} = z_{uv;11} = z_{uv;01} = z_{uv;10} = 1/4$ and copy values of all other components. Since $\mathbf{z}$ is again a feasible solution, pair $\mathbf{y}$, $\mathbf{z}$ contradicts Corollary 2.

We further inspect, how the configurations participate in expressing nodes in the form (6). If there is only one nonbasic edge (see Figure 5(a)), we can prove that none of the basic nodes in the adjacent objects depends on it. Consider the same vectors $\mathbf{y}$ and $\mathbf{z}$ as specified before. W.l.o.g., let $x_{uv;10}$ be the only nonbasic edge within $\{u, v\}$ and, w.l.o.g., let $x_{u;0}$ depend on $x_{uv;10}$. Observe that $y_{uv;10}$ differs from $z_{uv;10}$, but $y_{u;0}$ equals $z_{u;0}$. This contradicts Corollary 3.

If there are two nonbasic edges with a shared end-node in object v (Figure 3(b)), then, by definition, v is a dependency root. Three nonbasic edges induce one pair of edges with a shared end-node in each of the objects (Figure 5(b)), two dependency roots are thus formed.

The only configurations of nonbasic edges allowing dependance of basic nodes on more distant nonbasic variables are those ones defining dependency object pairs (Figure 4). In this case, nodes in one object can be locally expressed using nodes in the opposite object (and vice versa). For a parallel dependency object pair (Figure 4(a)) we derive

$$x_{u;0} = x_{uv;00} - x_{uv;11} + x_{v;1},$$

$$x_{u;1} = x_{uv;11} - x_{uv;00} + x_{v;0}.$$

Analogously, for an intersecting object pair (Figure 4(b)) we get

$$x_{u;0} = x_{uv;01} - x_{uv;10} + x_{v;0},$$

$$x_{u;1} = x_{uv;10} - x_{uv;01} + x_{v;1}.$$

Since $x_{uv;00}$ and $x_{uv;11}$ are nonbasic variables, to complete the expression of $x_{u;0}$ and $x_{u;1}$ as (6) would require to express $x_{v;1}$ and $x_{v;0}$. This again could delegate the process to a neighboring object. Such a procedure produces a complete expression of a basic variable when a dependency root is reached, as demonstrated in Figure 7(a). Following the path from u to w yields

$$x_{u;0} = x_{uv;00} - x_{uv;11} + x_{vw;10} - x_{vw;01} + x_{w;1}.$$

If a sequence of objects $(u_i)_{i=1}^k$ forms a path in (V,E), each $\{u_i,u_{i+1}\}$, $1 \le i \le k-1$ is w.l.o.g. an intersecting dependency object pair and $x_{u_k;0}$ is the only nonbasic node among all nodes in u_i's, then basic nodes in u_1 are expressed by nonbasic variables as follows.

$$x_{u_1;0} = \sum_{i=1}^{k-1} x_{u_iu_{i+1};01} - \sum_{i=1}^{k-1} x_{u_iu_{i+1};10} + x_{u_k;0}, \tag{7}$$

$$x_{u_1;1} = 1 - \sum_{i=1}^{k-1} x_{u_iu_{i+1};01} + \sum_{i=1}^{k-1} x_{u_iu_{i+1};10} - x_{u_k;0}. \tag{8}$$

The second possibility how to terminate the described process of expressing a basic node is to close a loop. This is demonstrated in Figure 7(b) where

$$x_{u;0} = \frac{1}{2}(x_{uv;00} - x_{uv;11} + x_{vs;10} - x_{vs;01} + x_{st;11} - x_{st;00} + x_{tu;00} - x_{tu;11} + 1).$$

In general, let $(u_i)_{i=1}^k$ be a sequence of objects forming a cycle in (V,E). W.l.o.g., let $\{u_i,u_{i+1}\}$ be intersecting for $1 \le i \le k-1$ and let $\{u_1,u_k\}$ be parallel. Assume than all nodes in u_i's are basic. Node $x_{u_1,0}$ can be again expressed as (7), but this is not yet the form (6) since $x_{u_k;0}$ is basic node. However, if we substitute

$$x_{u_k;0} = x_{u_ku_1;00} - x_{u_ku_1;11} + 1 - x_{u_1;0},$$

we obtain an equation which gives

$$x_{u_1;0} = \frac{1}{2}\Big(1 + \sum_{i=1}^{k-1} x_{u_iu_{i+1};01} - \sum_{i=1}^{k-1} x_{u_iu_{i+1};10} + x_{u_ku_1;00} - x_{u_ku_1;11}\Big). \tag{9}$$

Node $x_{u_1;1}$ can be now expressed as (6) using the equality $x_{u_1;1} = 1 - x_{u_1;0}$. Note it was essential that $\{u_1,u_k\}$ is parallel. Changing it to intersecting would result in an equation where both, $x_{u_1;0}$ and $x_{u_1;1}$ are missing. To express these variables, it is necessary to have a dependency cycle where all basic edges and nodes form a connected component in the underlying microstructure.

Finally, the most general situation when there is a path starting at object u and leading to object v which is part of a cycle (and it is the first such an object in the path) is handled in two steps. First, nodes in u are expressed along to the path to v as (7), (8), and second, nodes of v are expressed within the cycle as (9). To finish the proof it suffices to realize it is not possible to have more choices of following dependency object pairs to a root or cycle since (6) is unique for each basic variable. □

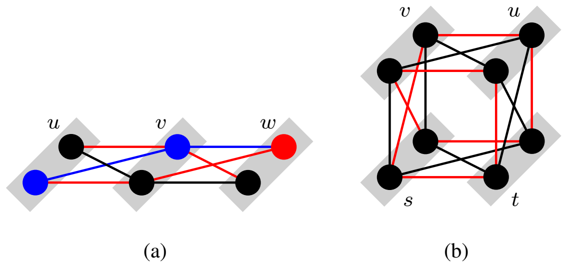

Figure 7. Expressing a nonbasic node variable in u along a path formed of dependency object pairs terminates only if (a) a dependency root is reached or (b) a loop is closed.

Theorem 5. *Given a binary energy minimization $(V,E,\boldsymbol{\theta})$ and a basis $\mathcal{B}$ of its LP relaxation, each basic edge of an object pair $e = \{u,v\} \in E$ depends only on a subset of nonbasic variables in e, $C_u = D_u(V,E,\mathcal{B})$, $C_v = D_v(V,E,\mathcal{B})$ and on nonbasic edges establishing dependency roots of C_u and C_v.*

Proof. If there are at least two nonbasic edges in $e = \{u,v\}$, then each basic edges in e is adjacent to a nonbasic edge in some node (inspect all possible configurations in Figures 3(b), 4(a), 4(b), 5(b)). It is thus possible to express every basic edge in e as a linear function of one nonbasic edge and one (basic or nonbasic) node. For example, we derive in Figure 4(a)

$$x_{uv;01} = x_{v;1} - x_{uv;11}. \tag{10}$$

If there is only one nonbasic edge in e (Figure 5(a)), then the basic edge which is not adjacent to it in any node is expressed using two nodes. In Figure 5(a), it holds

$$x_{uv;01} = x_{u;0} - x_{v;0} + x_{uv;10}. \tag{11}$$

Formula (6) for any basic edge in e is obtained if we substitute the basic nodes with their expressions (6). □

Theorem 6. *Given a binary energy minimization $(V,E,\boldsymbol{\theta})$, a feasible basis $\mathcal{B}$ of its LP relaxation, $i \in \mathcal{B}$ and x_i expressed in the form (6). It holds $b_i \in \{0, \frac{1}{2}, 1\}$ and, for all $j \in I \setminus \mathcal{B}$, $a_{ij} \in \{0, \pm\frac{1}{2}, \pm 1, \pm 2\}$.*

Proof. Possible values of b_i are prescribed by the half-integrality of basic feasible solutions [1]. As observable in Figure 6(a), values $\frac{1}{2}$ are assigned to basic variables in dependency components with a cycle and to basic edge variables in object pairs adjacent to such components.

By checking expressions (7), (8) and (9) derived in the proof of Theorem 5, we find that a_{ij} is in $\{0, \pm\frac{1}{2}, \pm 1\}$ for each basic node. The same conclusion holds for basic edges which can be expressed as (10). Consider a basic edge expressed as (11). Substitute for $x_{u;0}$ and $x_{v;0}$ their expressions (6). If $D_u(V,E,\mathcal{B})$ and $D_v(V,E,\mathcal{B})$ differ, then the

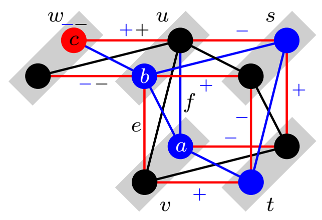

Figure 8. The blue signs show dependency of a on nonbasic variables (each dependency coefficient is either $+1$ or -1), while the black signs show dependency of b. Since $f = e + a - b$, edge f does not depend on node c, neither on the nonbasic edges in $\{w, u\}$. If the length of the dependency path from w to v was an even number, the dependency coefficient of f on c would be 2.

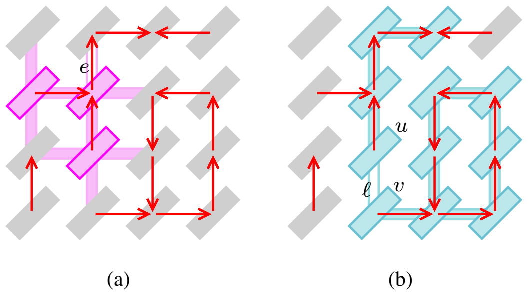

Figure 9. (a) The highlighted elements contain all the basic variables that depend on a nonbasic edge variable in object pair e. (b) A basic variable in object pair $\ell = \{u, v\}$ is expressed along paths from u and v to the root or cycle.

substituted expressions do not share variables with nonzero coefficients. However, if $D_u(V, E, \mathcal{B}) = D_v(V, E, \mathcal{B})$ then a nonbasic variable x_q with a nonzero coefficient can be present in both expressions, with coefficients of the same absolute value, hence the dependency of $x_{uv;01}$ on x_q either vanishes or the dependency coefficient is doubled. This is demonstrated in Figure 8. $\square$

5. Algorithm

Section 4 has established theoretical basis for a special version of the simplex algorithm executed for the LP relaxation of a binary energy minimization $(V, E, \boldsymbol{\theta})$. Here we put together all needed ingredients and give a description of the algorithm itself.

The algorithm performs steps over the LP relaxation diagram induced by the actual feasible basis $\mathcal{B}$. The diagram enables an efficient retrieval of dependency coefficients which corresponds to elements in the simplex tableau. For a faster traversal of the diagram, the direction of dependency object pairs towards the dependency root or cycle, as given in Figure 6(b), is maintained. Each node and edge variable is assigned by a boolean flag determining its basic/nonbasic status. Black/blue color of basic variables is recorded only for nodes. Color of basic edges can be derived from a local context.

For a nonbasic variable, it is possible to locate all basic variables that depend on it by traversing the dependency component in the direction opposite to the orientation of dependency object pairs in Figure 9(a). Similarly, for a basic variable, it is possible to traverse all nonbasic variables on which it depends. In this case a directed path to the dependency root or cycle is followed in one or two components, as can be seen in Figure 9(b). The dependency coefficient can also be determined by traversing components.

We apply a modified Dantzig's pivoting rule. The original variant chooses a nonbasic variable with the lowest negative cost (such a variable enters the basis), however, this is time-consuming. We rather use r doubly linked lists $\mathcal{L}_1, \ldots, \mathcal{L}_r$ and negative threshold values $\tau_1 < \tau_2 < \ldots < \tau_r = 0$. An objective cost θ is stored in $\mathcal{L}_s$ if $\tau_{s-1} < \theta \leq \tau_s$ (where $\tau_0 = -\infty$). The list can be found by a binary search in $\lceil \log_2 r \rceil$ steps. From performance reasons, we also do not implement any anti-cycling strategy. This is a normal approach in general LP solvers. No cycling has been observed during our numerous experiments. The experiments further revealed that it is sufficient to choose $r = 8$. Setting $r > 8$ has not resulted in any considerable improvement for the tested instances. It has also turned out that the number of iterations performed by the algorithm is nearly identical as in the case of the regular Dantzig rule.

The initial feasible basis follows the uniform pattern in Figure 1. A description of the algorithm follows.

Initialization. Create the initial feasible basis. For every object $u \in V$, mark $x_{u;1}$ as a nonbasic node. For every object pair $\{u, v\} \in E$, mark $x_{uv;11}$ as a nonbasic edge. All the other nodes and edges are basic variables. Apply a reparametrization (see section 2.1 of [8]) to $\boldsymbol{\theta}$ which changes costs of all basic variables to zero. Insert negative costs into lists $\mathcal{L}_i$.

Entering variable selection. Find the first nonempty list $\mathcal{L}_i$ and take its first element. The nonbasic variable x_e referenced by it is the *entering variable*. Assume it is located in an object or object pair e.

Leaving variable selection. Mark each object u that passes through e when traversing from u to the root or cycle, respectively. Search through the dependent variables and find a leaving variable (x_ℓ in an object or object pair ℓ) fulfilling the *minimum ratio test*. Finish the search prematurely if a zero ratio is found. Unmark all the marked objects.

Iteration. Update costs of all variables on which x_ℓ depends. Remove positive costs from lists, append costs that became negative. If a negative cost in $\mathcal{L}_i$ is changed and belongs now to $\mathcal{L}_j$, remove it form $\mathcal{L}_i$ and append it to $\mathcal{L}_j$. Update basic/nonbasic flag of x_e and x_ℓ. Update the representation of dependency components. Update colors of nodes.

Termination. Finish when all lists $\mathcal{L}_i$ are empty.

Time of one iteration is proportional to size of the involved dependency components. A good performance can be expected for those energy minimization instances which induce dependency components whose average size is either constant or slowly growing in size of the energy minimization graph. The experimental evaluation showed that such small components usually emerge for nonsubmodular instances and submodular instances where pairwise potentials are less dominant. On the other hand, submodular instances with more dominant pairwise potentials (*e.g.* segmentation instances), which result in solutions containing large regions assigned by the same label, tend to form large dependency components during the late iterations of the algorithm. In the next section, we report details on the favorable type of problems.

6. Experiments

We have implemented the specialized simplex algorithm (SA) in C++. It supports the creation of general graphs and computes with double precision floating point numbers. The max-flow based QPBO implementation by Kolmogorov [18] (BK) is used for comparison. All the sources were compiled in Microsoft Visual Studio 2012 and run on a notebook with Intel Core i5-4300M 2.6 GHz, 12 GB RAM and 64-bit Windows 7. The evaluation is done for vision problems and random data.

6.1. Vision problems

Test data are taken mainly from the empirical comparison of max-flow algorithms [15] which targets a wide range of max-flow algorithms based on augmenting-path, push-relabel or pseudoflow principles. Since their performance relative to BK algorithm is known, it is possible to compare SA with them. The data are available at [20] in the form of QPBO graphs. We turned them into an energy minimization format.

Decision Tree Field (DTF) is a recently introduced model by Nowozin *et al.* [11] that combines random forests and conditional random fields. 99 instances are available, giving a sufficiently representative sample. The problem is nonsubmodular and involves dense graphs. We measure performance of SA relative to BK. Ratios of running times (SA/BK) sorted in ascending order are plotted in Figure 10. SA is at least two times faster than BK for 80 instances. It is slower only for 4 instances. The average time of the best performing max-flow algorithm over DTFs reported in [15] is about 46% of the average time of BK (provided that the initialization time is counted). SA achieves 31% of BK.

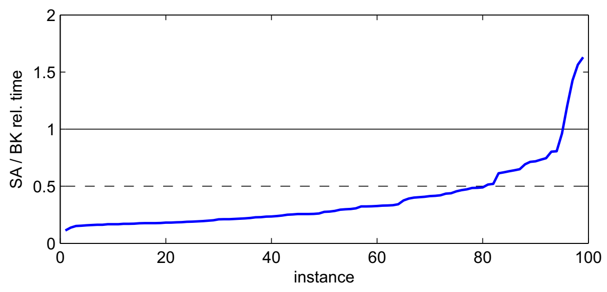

Figure 10. SA/BK relative time for Decision Tree Fields.

The amount of samples available for the next problems is considerably lower. We have collected two instances of *Super Resolution* (a sparse nonsubmodular problem) and six instances of *Deconvolution* with 3x3 or 5x5 blur-kernel (a dense nonsubmodular problem, graph connectivity is 24 or 80, respectively). These instances were considered by Rother *et al.* in [13]. Some of them (#3, #7, #9) are provided at [20]. *Shape Fitting* [10] is a representative of a sparse submodular problem. An instance (`LB07-bunny-sml`) has been taken at [19].

Measured absolute running times are in Table 1. Two selected DTFs are included there for comparison. To demonstrate a huge gain over a general LP solver, running times of IBM ILOG CPLEX 12.6 for the instances are also included. We tested the primal as well as the dual simplex method. Time limit of 10 minutes was applied for each computation.

We can see that BK performs better than SA for instances #3 – 8. However, the responsiveness of SA is within reasonable limits. The situation is different for deconvolution with a 5x5 blur-kernel. BK outperforms SA for #9. Conversely, SA outperforms BK for #10. Note that there are max-flow algorithms better than BK for this denser variant of deconvolution (approx. 1.5 times faster [15]).

6.2. Scalable random data

Here we study how the performance of SA scales in the size of the input graph. For this purpose we need a dataset of instances with equally growing number of objects. We generate a subset of square grids from 10×10 to 500×500 with 8-neighborhood system. Unary potentials are generated as independent gaussians $\theta_{u;0}, \theta_{u;1} \sim \mathcal{N}(0,1)$. Pairwise potentials are set to zero for $\theta_{uv;00}$ and $\theta_{uv;11}$. Values of $\theta_{uv;01}$ and $\theta_{uv;10}$ are generated as $\mathcal{N}(0,2)$. This setup is an instance of the Ising model with mixed potentials. Overall, we obtain sparse nonsubmodular inputs containing about 66% undecided variables in the optimal LP relaxation solution.

The evaluation is also done for the variant of SA which strictly follows Danzig's pivoting rule (a binary heap is used to store negative objective costs in this case). This variant

#	description	objects	obj. pairs	SA [ms]	BK [ms]	CPLEX primal [ms]	CPLEX dual [ms]
1	dtf-78	7776	217414	**149**	1337	time limit exceeded	96315
2	dtf-94	8384	234237	**276**	1523	time limit exceeded	129574
3	sup. res. (4-con)	5246	10345	3.7	**1.5**	218	421
4	sup. res. (8-con)	5246	20545	10.4	**3.2**	483	1388
5	shape fit. (6-con)	805800	2391242	628	**135**	out of memory	out of memory
6	deconv. 3x3	1024	11346	5.5	**2.2**	1451	296
7	deconv. 3x3	1000	10968	4.9	**3.1**	1185	358
8	deconv. 3x3	1000	10968	4.6	**2.5**	858	405
9	deconv. 5x5	1000	33900	71.7	**8.3**	17924	1872
10	deconv. 5x5	1024	35400	**7.5**	23.9	22730	2496
11	deconv. 5x5	880	29820	**32.1**	46.4	1841	1342

Table 1. Running time of SA, BK and CPLEX 12.6 for vision instances.

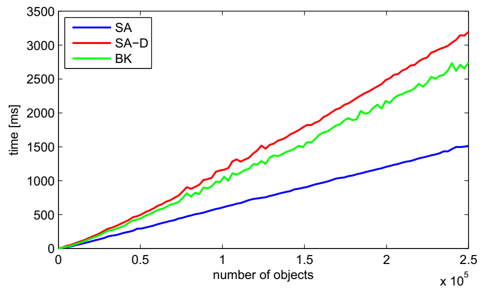

Figure 11. Running time of SA, SA-D and BK for random grids.

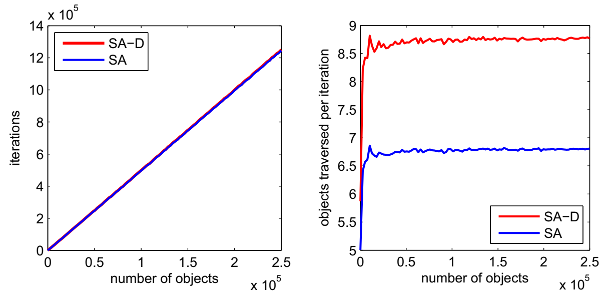

Figure 12. SA and SA-D: the number of performed iterations and the average number of objects traversed per iteration.

is denoted as SA-D. The dependency of running time on the number of objects is plotted in Figure 11. SA performs better than BK and the measured time is linear. In contrast, a non-linearity caused by the binary heap usage is observable for SA-D. It is also interesting that the function for SA is the smoothest one.

Figure 12 reveals two dependencies. The number of iterations performed by the simplex algorithm is linear. It is almost identical for both, SA and SA-D, the displayed functions coalesce. Moreover, the average number of objects traversed within one iteration is almost constant. Surprisingly, the constant is greater for SA-D.

In conclusion, SA demonstrated a very good performance and stability in this test.

7. Conclusion

We have presented a graph-based version of the simplex method for pairwise energy minimization with binary variables. The experiments confirmed that the proposed algorithm is efficient for certain types of vision problems. Outperforming other solvers on DTF instances represents an immediate practical benefit.

We believe the method has a very good potential for further research. We obtained an algorithm competitive with best solvers based on finding maximum flow in a network, which has been intensively studied by many researches for a long time. Our algorithm has not undergone such evolution. Promising opportunities for improvements are thus expectable. For example, a different pivoting rule might result in a smaller size of dependency components, a more advanced representation of components might reduce the number of traversed objects when searching for the leaving variable, *etc*. The algorithm may also be suitable for parallelization.

An interesting question is whether the approach can be efficiently generalized to the LP relaxation of multi-label energy minimization problems. The situation is surely more difficult. More labels induce richer relations among basic and nonbasic variables. Except that, the LP relaxation becomes as hard as general LP [12] and, as a consequence, components of optimal solutions are, roughly speaking, arbitrary fractions. On the other hand, known methods for solving the LP relaxation of multi-label problems like message passing [7] are considerably slower than max-flow algorithms. Even a slower retrieval of simplex tableau elements (*e.g.* by solving subsystems of linear equations within dependency components) could still result in a method with better performance. The presented applicability of the simplex algorithm in the binary setting should encourage such considerations.

Acknowledgment

The author was supported by the Czech Science Foundation project P202/12/2071.

References

[1] E. Boros and P. L. Hammer. Pseudo-Boolean optimization. *Discrete Applied Mathematics*, 123(1-3):155–225, 2002. 1, 5

[2] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 26(9):1124–1137, Sept. 2004. 1

[3] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 23(11):1222–1239, Nov. 2001. 1

[4] G. Dantzig and M. Thapa. *Linear Programming 1: Introduction*. Springer, 1997. 1, 2, 3

[5] D. Goldfarb and J. Hao. A primal simplex algorithm that solves the maximum flow problem in at most nm pivots and O(n^2m) time. *Mathematical Programming*, 47(1-3):353–365, 1990. 1

[6] H. Ishikawa. Exact optimization for Markov random fields with convex priors. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 25(10):1333–1336, Oct. 2003. 1

[7] V. Kolmogorov. Convergent tree-reweighted message passing for energy minimization. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 28(10):1568–1583, 2006. 8

[8] V. Kolmogorov and C. Rother. Minimizing nonsubmodular functions with graph cuts-a review. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 29(7):1274–1279, 2007. 1, 6

[9] A. Koster, C. P. M. van Hoesel, and A. W. J. Kolen. The partial constraint satisfaction problem: Facets and lifting theorems. *Operations Research Letters*, 23(3–5):89–97, 1998. 2

[10] V. Lempitsky and Y. Boykov. Global optimization for shape fitting. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, June 2007. 7

[11] S. Nowozin, C. Rother, S. Bagon, T. Sharp, B. Yao, and P. Kohli. Decision tree fields. In D. N. Metaxas, L. Quan, A. Sanfeliu, and L. J. V. Gool, editors, *IEEE International Conference on Computer Vision*, pages 1668–1675. IEEE, 2011. 7

[12] D. Průša and T. Werner. Universality of the local marginal polytope. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 37(4):898–904, April 2015. 8

[13] C. Rother, V. Kolmogorov, V. S. Lempitsky, and M. Szummer. Optimizing binary MRFs via extended roof duality. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2007. 1, 7

[14] D. Schlesinger and B. Flach. Transforming an arbitrary Min-Sum problem into a binary one. Technical Report TUD-FI06-01, Dresden University of Technology, Germany, April 2006. 1

[15] T. Verma and D. Batra. Maxflow revisited: An empirical comparison of maxflow algorithms for dense vision problems. In *Proceedings of the British Machine Vision Conference*, pages 61.1–61.12. BMVA Press, 2012. 1, 7

[16] M. J. Wainwright and M. I. Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 1(1-2):1–305, 2008. 1

[17] T. Werner. A linear programming approach to max-sum problem: A review. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 29(7):1165–1179, July 2007. 1

[18] V. Kolmogorov – web pages. `http://pub.ist.ac.at/~vnk/`. 7

[19] University of Western Ontario – Max-flow problem instances in vision. `http://vision.csd.uwo.ca/data/maxflow/`. 7

[20] T. Verma, D. Batra – MaxFlow Revisited. `http://ttic.uchicago.edu/~dbatra/research/mfcomp/`. 7