The Lazy Flipper: Efficient Depth-limited Exhaustive Search in Discrete Graphical Models

Bjoern Andres, Jörg H. Kappes, Thorsten Beier, Ullrich Köthe and Fred A. Hamprecht

HCI, University of Heidelberg, Germany

Abstract. We propose a new exhaustive search algorithm for optimization in discrete graphical models. When pursued to the full search depth (typically intractable), it is guaranteed to converge to a global optimum, passing through a series of monotonously improving local optima that are guaranteed to be optimal within a given and increasing Hamming distance. For a search depth of 1, it specializes to ICM. Between these extremes, a tradeoff between approximation quality and runtime is established. We show this experimentally by improving approximations for the non-submodular models in the MRF benchmark [1] and Decision Tree Fields [2].

1 Introduction

Discrete graphical models [3] have become a standard tool in computer vision. A central problem is the minimization of energy functions that decompose according to a graphical model. This combinatorial optimization problem can be solved in polynomial time by dynamic programming if the graph is acyclic [4] or a junction tree with limited treewidth can be constructed [5], and by finding a minimum s-t-cut if the objective function is (permuted) submodular [6–8]. The general problem is, however, NP-hard [9]. In cases where exact optimization is intractable, one has to settle for approximations. Substantial progress has been made in this direction through LP relaxation [10–15] and graph-based optimization [6].

We contribute to the class of graph-based optimization methods a depth-limited exhaustive search algorithm, the Lazy Flipper, that constrains the search space based on the structure of the graphical model. The Lazy Flipper starts from an arbitrary initial assignment of values to the variables and searches for flips of variables that reduce the energy. As soon as such a flip is found, the current configuration is updated accordingly, i.e. in a greedy fashion. In the beginning, only one variable is flipped at a time. Once a configuration is found whose energy can no longer be reduced by flipping of *single* variables, all those subsets of two and successively more variables that are connected via potentials in the graphical model are considered. When a subset of more than one variable is flipped, all smaller subsets that are affected by the flip are revisited. This allows the Lazy Flipper to perform an exhaustive search over all subsets of variables

whose flip potentially reduces the energy. Two special data structures described in Section 3 are used to represent each subset of connected variables precisely once and to exclude subsets from the search whose flip cannot reduce the energy due to the structure of the graphical model and the history of unsuccessful flips.

We evaluate the quality and convergence of Lazy Flipper approximations, first, on simulated submodular problems where the global optimum is accessible and then on the non-submodular problems of the MRF benchmark [1] and Decision Tree Fields for Chinese character inpainting [2].

2 Related Work

In the variety of optimization algorithms for discrete graphical models, there are at least three major classes: (i) Algorithms based on polyhedral approximation that consider outer relaxations of the corresponding linear program constrained to the marginal polytope, (ii) move-making algorithms that start from an initial configuration and seek to iteratively improve the objective in a greedy manner and (iii) Metropolis-Hastings algorithms that explore the configuration space stochastically via a Markov chain.

Polyhedral Approximations. Optimization problems for discrete graphical models can be formulated as linear programming (LP) problems. The complexity of these LPs lies in the marginal polytope which constrains the feasible set. The problem is commonly made tractable by considering an outer relaxation of the marginal polytope, e.g. the local polytope [13]. Well-known algorithms for optimizing over the local polytope are TRBP [15, 13], TRW-S [14], and Dual Decomposition [16]. While the latter provide lower bounds on the global minimum, the primal solutions obtained by these algorithms need not be integral such that rounding techniques are required.

Move-making algorithms. Instead of solving a relaxed problem, move-making algorithms seek to iteratively improve the objective without leaving the discrete feasible set. In each step, a single tractable optimization problem that represents a subset of the combinatorial search space is solved. This set always includes the current configuration which guarantees monotonic decrease of the objective. Since the state space is finite, these algorithm are guaranteed to converge. Fixed points are optimal w.r.t. the set of moves. Algorithms of this class include ICM [17], Block-ICM [18], α-Expansion and $\alpha\beta$-swap [6]. More recently, Jung et al. [19] have suggested and analyzed an algorithm that explores subspaces defined by randomly selected connected subsets of variables.

Metropolis-Hastings algorithms such as the well-known Gibbs sampler and the Swendsen-Wang sampler [20] construct a Markov chain whose stationary distribution corresponds to the Gibbs distribution of the graphical model.

The Lazy Flipper belongs to the class of move-making algorithms and generalizes ICM [17]. While ICM leaves all variables except one fixed in each step, the Lazy Flipper can optimize over larger (for small models: all) connected subgraphs of a graphical model. It extends Block-ICM [18] that optimizes over specific subsets of variables in grid graphs to irregular and higher-order graphical models.

Moreover, the Lazy Flipper is a deterministic counterpart of [19]. Exactly as in [19], sets of variables which are connected in the graphical model are considered and variables flipped if these flips improve the objective. In contrast to [19], unique representatives of these sets are visited in a deterministic order. Conceptually, the Lazy Flipper is a reverse search algorithm [21] to which we add a data structure for indexing connected subsets.

3 The Lazy Flipper Data Structures

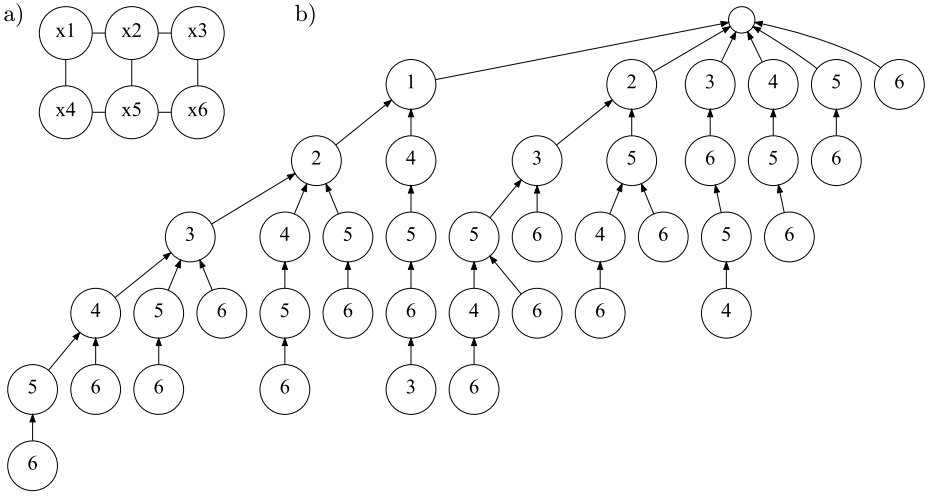

Fig. 1. All connected subgraphs of a graphical model (a) can be represented uniquely in a connected subgraph tree (CS-tree) (b). Every path from a node in the CS-tree to the root node corresponds to a connected subgraph in the graphical model. While there are $2^6 = 64$ subsets of variables in total in this example, only 40 of these subsets are connected.

Naïve attempts to generalize ICM and Block-ICM to optimize over subgraphs of size k consider all sequences of pairwise distinct variables in which each variable is connected to at least one of its predecessors and ignore the fact that many of these sequences represent the same set. However, the redundancy is large. As an example, consider the grid model with six variable nodes in Fig. 1a. Although there is only one connected set that contains all six variables, 208 out of the $6! = 720$ possible sequences of these nodes meet the requirement that each node is connected to at least one of its predecessors. Finding a unique representative for each connected subset of variables is therefore important.

Even with a list of the connected subsets of size k at hand, it is inefficient to iterate over these sets and search all possible labelings of one set exhaustively

in each iteration because smaller subsets in the intersection of subsets of size k are searched implicitly and repeatedly.

We use two special data structures to avoid this redundancy, at the cost of storing one unique representative for each connected subgraph of size smaller than or equal to k. The first data structure that we call a *connected subgraph tree (CS-tree)* ensures that only connected subsets of variables are considered, i.e. sets of variables which are connected via potentials in the graphical model. Moreover, it ensures that every such subset is represented precisely once (and not repeatedly) by an ordered sequence of its variables, cf. [22].

The rationale behind this concept is the following: If the flip of one variable and the flip of another variable not connected to the first one do not reduce the energy then it is pointless to try a simultaneous flip of both variables because the (energy increasing) contributions from both flips would sum up. Furthermore, if the flip of a disconnected set of variables reduces the energy then the same and possibly better reductions can be obtained by flipping connected subsets of this set consecutively, in any order. All disconnected subsets of variables can therefore be excluded from the search.

The second data structure is a *tag list* that prevents the repeated assessment of unsuccessful flips. The idea is the following: If some variables have been flipped in one iteration (and the current best configuration has been updated accordingly), it suffices to revisit only those sets of variables that are connected to at least one variable that has been flipped. All other sets of variables are excluded from the search because the potentials that depend on these variables are unaffected by the flip and have been assessed in their current state before.

3.1 Connected Subgraph Tree (CS-tree)

The CS-tree represents subsets of connected variables uniquely. Every node in the CS-tree, except the root node that represents the empty set, is labeled with the integer index of one variable in the graphical model. The CS-tree is constructed such that every connected subset of variables in the graphical model corresponds to precisely one path in the CS-tree from a node to the root node, the node labels along the path indicating precisely the variables in the subset, and vice versa, there exists precisely one connected subset of variables in the graphical model for each path in the CS-tree from a node to the root node.

In order to guarantee by construction of the CS-tree that each subset of connected variables is represented precisely once, the variable indices of each subset are put in a special order, namely the lexicographically smallest order in which each variable is connected to at least one of its predecessors. The following definition of these sequences of variable indices is recursive and therefore motivates an algorithm for the construction of the CS-tree for the Lazy Flipper. A small grid model and its complete CS-tree are depicted in Fig. 1.

Definition 1 (CSR-Sequence). *Given an undirected graph $G = (V, E)$ whose $m \in \mathbb{N}$ vertices $V = \{1, \ldots, m\}$ are integer indices, every sequence that consists of only one index is called* connected subset representing (CSR). *Given $n \in \mathbb{N}$*

and a CSR-sequence $(v_1, \ldots, v_n)$*, a sequence* $(v_1, \ldots, v_n, v_{n+1})$ *of* $n+1$ *indices is called a* CSR-sequence *precisely if the following conditions hold:*

(i) v_{n+1} *is not among its predecessors, i.e.* $\forall j \in \{1, \ldots, n\} : v_j \neq v_{n+1}$.

(ii) v_{n+1} *is connected to at least one of its predecessors, i.e.* $\exists j \in \{1, \ldots, n\} : \{v_j, v_{n+1}\} \in E$.

(iii) $v_{n+1} > v_1$.

(iv) If $n \geq 2$ *and* v_{n+1} *could have been added at an earlier position* $j \in \{2, \ldots, n\}$ *to the sequence, fulfilling (i)–(iii), all subsequent vertices* $v_j, \ldots, v_n$ *are smaller than* v_{n+1}*, i.e.* $\forall j \in \{2, \ldots, n\}$*:*

$$\{v_{j-1}, v_{n+1}\} \in E \;\Rightarrow\; \forall k \in \{j, \ldots, n\} : v_k < v_{n+1} \quad .$$

Based on this definition, three functions are sufficient to recursively build the CS-tree T of a graphical model G, starting from the root node. The function $q = \mathit{growSubset}(T, G, p)$ appends to a node p in the CS-tree the smallest variable index that is not yet among the children of p and fulfills (i)–(iv) for the CSR-sequence of variable indices on the path from p to the root node. It returns the appended node or the empty set if no suitable variable index exists. The function $q = \mathit{firstSubsetOfSize}(T, G, n)$ traverses the CS-tree on the current deepest level $n-1$, calling the function *growSubset* for each leaf until a node can be appended and thus, the first subset of size n has been found. Finally, the function $q = \mathit{nextSubsetOfSameSize}(T, G, p)$ starts from a node p, finds its parent and traverses from there in level order, calling *growSubset* for each node to find the length-lexicographic successor of the CSR-sequence associated with the node p, i.e. the representative of the next subset of the same size. These functions are used in Algorithm 1 to *construct* the CS-tree. In contrast, the *traversal* of already constructed parts of the CS-tree (when revisiting subsets of variables after successful flips) is performed by functions associated with tag lists.

3.2 Tag Lists

Tag lists are used to tag variables that are affected by flips. A variable is affected by a flip either because it has been flipped itself or because it is connected in the graphical model to a flipped variable. The tag list data structure comprises a Boolean vector in which each entry corresponds to a variable, indicating whether or not this variable is affected by recent flips. As the total number of variables can be large (10^6 is not exceptional) and possibly only a few variables are affected by flips, a list of all affected variables is maintained in addition to the vector. This list allows the algorithm to untag all tagged variables without re-initializing the entire Boolean vector. The two fundamental operations on a tag list L are $\mathit{tag}(L, x)$ which tags the variable with the index x, and $\mathit{untagAll}(L)$.

For the Lazy Flipper, three special functions are used in addition: Given a tag list L, a (possibly incomplete) CS-tree T, the graphical model G, and a node $s \in T$, $\mathit{tagConnectedVariables}(L, T, G, s)$ tags all variables on the path from s to the root node in T, as well as all nodes that are connected in the graphical model to at least one of these nodes. The function $s = \mathit{firstTaggedSubset}(L, T)$

traverses the first level of T and returns the first node s whose variable is tagged (or the empty set if all variables are untagged). Finally, the function $t = \mathit{nextTaggedSubset}(L, T, s)$ traverses T in level order, starting with the successor of s, and returns the first node t for which the path to the root contains at least one tagged variable. These functions, together with those of the CS-tree, are sufficient for the Lazy Flipper, Algorithm 1.

Algorithm 1: Lazy Flipper

Input: G: graphical model with $m \in \mathbb{N}$ binary variables, $c \in \{0,1\}^m$: initial configuration, $n_{\max} \in \mathbb{N}$: maximum size of subgraphs to be searched
Output: $c \in \{0,1\}^m$ (modified): configuration corresponding to the smallest upper bound found (c is optimal within a Hamming radius of $n_{\max}$).

```
1  n ← 1; CS-Tree T ← {root}; TagList L1 ← ∅, L2 ← ∅;
2  repeat
3      s ← firstSubsetOfSize(T, G, n);
4      if s = ∅ then break;
5      while s ≠ ∅ do
6          if energyAfterFlip(G, c, s) < energy(G, c) then
7              c ← flip(c, s);
8              tagConnectedVariables(L1, T, G, s);
9          end
10         s ← nextSubsetOfSameSize(T, G, s);
11     end
12     repeat
13         s2 ← firstTaggedSubset(L1, T);
14         if s2 = ∅ then break;
15         while s2 ≠ ∅ do
16             if energyAfterFlip(G, c, s2) < energy(G, c) then
17                 c ← flip(c, s2);
18                 tagConnectedVariables(L2, T, G, s2);
19             end
20             s2 ← nextTaggedSubset(L1, T, s2);
21         end
22         untagAll(L1); swap(L1, L2);
23     end
24     if n = nmax then break;
25     n ← n + 1;
26 end
```

4 The Lazy Flipper Algorithm

4.1 For Binary Variables

In the main loop (lines 2–26) of Algorithm 1, the size n of subsets is incremented until the limit $n_{\max}$ is reached (line 24). This loop falls into two parts, the *explo-*

ration part (lines 3–11) and the *revisiting part* (lines 12–23). In the exploration part, flips of previously unseen subsets of n variables are assessed. The current best configuration c is updated in a greedy fashion, i.e. whenever a flip yields a lower energy. At the same time, the CS-tree is grown, using the functions defined in Section 3.1. In the revisiting part, all subsets of sizes 1 through n that are affected by recent flips are assessed iteratively, until no flip of any of these subsets reduces the energy (line 14). The indices of affected variables are stored in the tag lists L_1 and L_2 (cf. Section 3.2). In practice, the Lazy Flipper can be stopped at any point, e.g. when a time limit is exceeded, and the current best configuration c taken as the output. It eventually reaches configurations for which it is guaranteed that no flip of n or less variables can yield a lower energy because all such flips that could potentially lower the energy have been assessed (line 14). Such configurations are therefore guaranteed to be optimal within a Hamming radius of n:

Definition 2 (Hamming-n bound). *Given a function* $E : \{0,1\}^m \to \mathbb{R}$*, a configuration* $c \in \{0,1\}^m$*, and* $n \in \mathbb{N}$*,* $E(c)$ *is called a* Hamming-n upper bound *on the minimum of* E *precisely if* $\forall c' \in \{0,1\}^m (\|c' - c\|_0 \leq n \Rightarrow E(c) \leq E(c'))$.

4.2 For the Multi-Label Case

In principle, any encoding of a graphical model with multiple labels as a graphical model with binary labels can be used to apply Algorithm 1. However, this is not the most efficient approach in practice because the CS-tree is grown for each fully connected subgraph of binary variables that encode a non-binary variable in the original model.

To avoid this, we use a specialized function *energyAfterFlip* for the multi-label case. For a subset of n variables, this function searches all labelings of these variables in which each label differs from the label assigned to the same variable in the current best configuration, i.e. all labelings in which all labels change. This corresponds to an exhaustive search over the label space that still avoids the repeated assessment of smaller subgraphs.

4.3 Initialization

Despite its optimality guarantees for sufficiently large search depths and fixed points at limited search depth, the Lazy Flipper is still a greedy algorithm and is useful in practice only in conjunction with good initializations. TRW-S and α-expansion provide such initializations for a wide range of models.

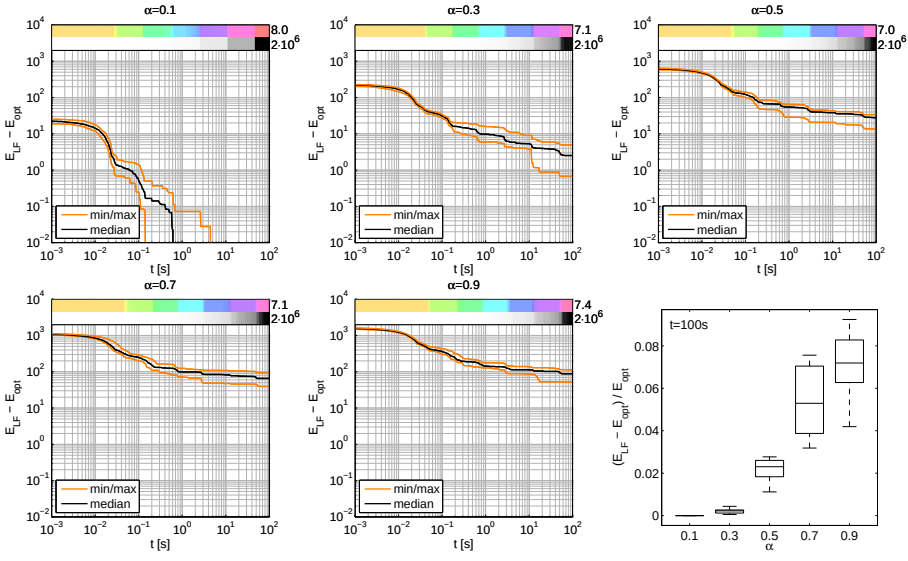

Fig. 2. Upper bounds on the minimum energy of a graphical model can be found by flipping subsets of variables. The deviation of these bounds from the minimum is shown above for ensembles of ten random Ising models (Section 5.1). Compared to ICM where only one variable is flipped at a time, the Lazy Flipper finds significantly tighter bounds by flipping also larger subsets. The deviations increase with the coupling strength α. Color scales and gray scales indicate the size and the total number of searched subsets.

5 Experiments

5.1 Ferromagnetic Ising model

To study the Lazy Flipper in a simulated setting where global optima are accessible, we consider an Ising model with $m \in \mathbb{N}$ binary variables which are associated with points on a 2-dimensional Cartesian grid and connected via potentials to their nearest neighbors. For $\alpha \in (0, 1)$ and $\forall x \in \{0, 1\}^m$:

$$E(x) = \sum_{j=1}^{m} E_j(x_j) + \alpha \sum_{j=1}^{m} \sum_{\substack{k=j+1 \\ k \sim j}}^{m} |x_j - x_k| \quad . \tag{1}$$

The global minimum of this submodular function is found via a graph cut. For $50 \cdot 50 = 2500$ variables and each $\alpha \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$, an ensemble of ten random Ising models is simulated in order to compare Lazy Flipper approximations to the global optimum. Additionally, ensembles of different sizes are simulated in order to measure how the runtime of lazy flipping depends on the size of the model and the coupling strength α. The first order potentials E_j

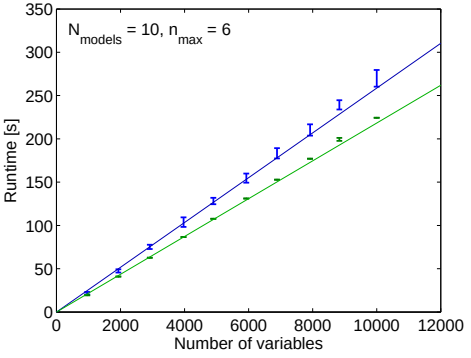

Fig. 3. For a fixed maximum subgraph size ($n_{\max} = 6$), the runtime of lazy flipping scales only slightly more than linearly with the number of variables in the Ising model. It is measured for coupling strengths $\alpha = 0.25$ (upper curve) and $\alpha = 0.75$ (lower curve). Error bars indicate the standard deviation over 10 random models, and lines are fitted by least squares. Lazy flipping takes longer (0.0259 seconds per variable) for $\alpha = 0.25$ than for $\alpha = 0.75$ (0.0218 s/var) because more flips are successful and thus initiate revisiting.

are initialized randomly by drawing $E_j(0)$ uniformly from the interval $[0, 1]$ and setting $E_j(1) := 1 - E_j(0)$.

For each model, the Lazy Flipper is initialized with a configuration that minimizes the sum of the first order potentials. Upper bounds on the minimum energy found by means of lazy flipping converge towards the global optimum as depicted in Fig. 2. Color scales and gray scales in this figure respectively indicate the maximum size and the total number of distinct subsets that have been searched, averaged over all models in the ensemble. Upper bounds are tightened significantly by searching larger subsets of variables, independent of the coupling strength α. It takes the Lazy Flipper less than 100 seconds (on a single CPU of an Intel Quad Xeon E7220 at 2.93GHz) to exhaustively search all connected subsets of 6 variables. The size of the CS-tree (approximately 50 MB in this case) becomes limiting for large problems. However, for regular graphs, implicit representations can be envisaged which overcome this limitation.

For a fixed maximum subgraph size, the runtime of lazy flipping scales only slightly more than linearly with the number of variables (Fig. 3).

5.2 MRF Benchmark

To study whether lazy flipping can improve labelings close to the LP lower bound found by state-of-the-art algorithms with different fixed points, we consider all non-submodular problems in the MRF benchmark [1] and initialize the Lazy Flipper with the approximations found by the best performing algorithm [1], i.e. either TRW-S or α-expansion.

It can be seen from the results in Tab. 1 that all approximations are improved. Moreover, it is benefitial in all cases to search subsets of more than one variable, i.e. to go beyond ICM.

5.3 Decision Tree Fields

Finally, we consider a hard problem: optimization of Decision Tree Fields that are learned to inpaint Chinese characters [2]. The 100 models from the supple-

Dataset	Algorithm	E	ΔE	H	t [s]
Tsukuba	TRW-S	369290	-	-	-
	LF-1	369266	24	1	0.4
	LF-2	369255	35	3	23.3
	LF-3	369219	71	8	1526.0
Venus	TRW-S	3048387	-	-	-
	LF-1	3048387	0	0	$1.4 \cdot 10^{-6}$
	LF-2	3048325	62	2	61.3
	LF-3	3048325	62	2	147.8
Teddy	α-exp.	1343326	-	-	-
	LF-1	1343326	0	0	$7.4 \cdot 10^{-6}$
	LF-2	1343182	144	42	1314.5
Panorama	α-exp.	151202	-	-	-
	LF-1	151200	2	1	1.5
	LF-2	151171	31	8	36.6
	LF-3	151171	31	8	70.9
	LF-4	151168	34	17	26781.6
Family	TRW-S	184825	-	-	-
	LF-1	184825	0	0	$1.8 \cdot 10^{-6}$
	LF-2	184813	12	2	14.8
	LF-3	184813	12	2	33.2
	LF-4	184813	12	2	603.9
House	TRW-S	37580724	-	-	-
	LF-1	37580618	106	24	7.8
	LF-2	37580552	172	46	8290.1
Penguin	TRW-S	15349028	-	-	-
	LF-1	15347532	1496	20	2.7
	LF-2	15347193	1835	43	2828.0

Table 1. Lazy flipping improves labelings found by TRW-S for all non-submodular models in the MRF benchmark [1]. E denotes the absolute energy, ΔE the improvement, H the Hamming distance to the TRW-S result and t the runtime.

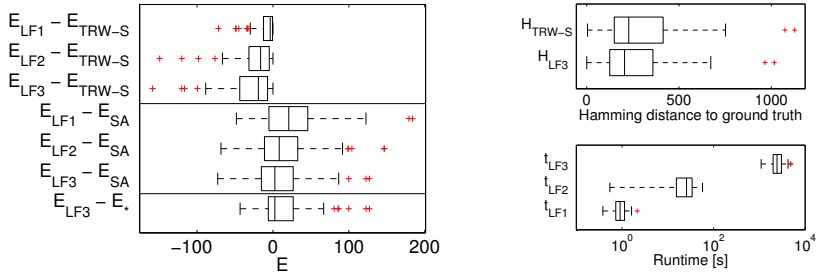

Fig. 4. Lazy flipping improves labelings found by TRW-S for Decision Tree Fields [2]. While these labelings deviate by a Hamming distance of 228 in the median from the ground truth, lazy flipping reduces this deviation by 10% to 205. For 45 out of the 100 models, these improved labelings have a lower energy than those found by TRW-S *and* simulated annealing and are thus an improvement over the state of the art.

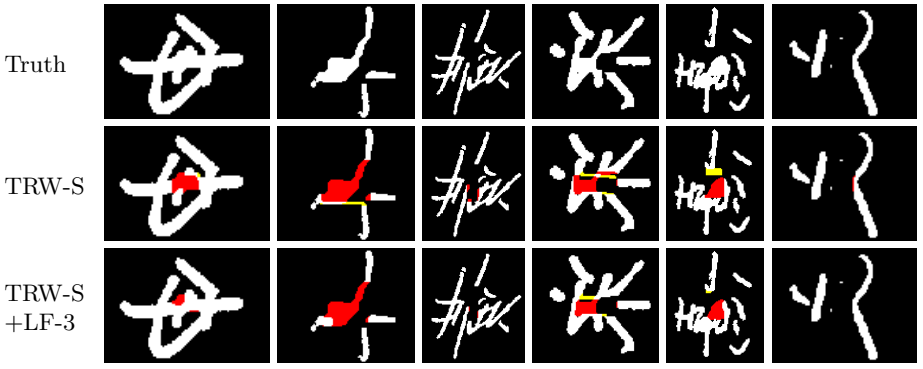

Fig. 5. Lazy flipping corrects both false positives (yellow) and false negatives (red) in labelings found by TRW-S for Decision Tree Fields for Chinese character inpainting. The results for all 100 images are contained in the supplementary material.

mentary material of [2] have between 4992 and 17856 binary variables and first order factors (median: 8920) as well as between 141232 and 535870 second order factors (median: 260246). Variable are connected via factors to as many as 64 other variables. The state of the art in optimizing these objective functions [2] is TRW-S (41 models) and simulated annealing (59 models).

It can be seen from the results in Fig. 4 that lazy flipping improves labelings found by TRW-S which are used as initializations. While labelings found by TRW-S deviate by a Hamming distance of 228 in the median from the ground truth, lazy flipping reduces this deviation by 10% to 205 (Fig. 4 and 5). For 45 out of the 100 models, the improved Hamming-3 optimal labelings found by lazy flipping have a lower energy than the labelings found by both TRW-S and simulated annealing and are thus an improvement over the state of the art. The runtime (43 minutes) is competitive with that of simulated annealing.

6 Conclusion

The optimum of a function of discrete variables that decomposes according to a graphical model can be found by an exhaustive search over only the connected subgraphs. We implemented this search, using a CS-tree to efficiently and uniquely enumerate the subgraphs. The algorithm is guaranteed to converge to a global minimum when searching through all subgraphs which is typically intractable. With limited runtime, approximations can be found by restricting the search to subgraphs of a given maximum size. These approximations are guaranteed to be optimal within equal Hamming distance.

In practice, the algorithm is useful in conjunction with good initializations. Starting from fixed points of TRW-S and α-expansion, better approximations for all non-submodular problems in the MRF benchmark [1] were found. For the problem of inpainting Chinese characters by optimizing Decision Tree Fields,

TRW-S in conjunction with the proposed algorithm is an improvement over the state of the art for 45 out of 100 models.

For large problems, the applicability of the proposed algorithm is limited by the memory required for the CS-tree. For regular graphs, this limit can be overcome by an implicit representation of the CS-tree which is subject of future research.

References

1. Szeliski, R., Zabih, R., Scharstein, D., Veksler, O., Kolmogorov, V., Agarwala, A., Tappen, M., Rother, C.: A comparative study of energy minimization methods for markov random fields with smoothness-based priors. TPAMI **30** (2008) 1068–1080
2. Nowozin, S., Rother, C., Bagon, S., Sharp, T., Yao, B., Kohli, P.: Decision tree fields. In: ICCV. (2011)
3. Koller, D., Friedman, N.: Probabilistic Graphical Models. MIT Press (2009)
4. Pearl, J.: Probabilistic reasoning in intelligent systems: networks of plausible inference. Morgan Kaufmann, San Francisco, CA, USA (1988)
5. Lauritzen, S.L.: Graphical Models. Statistical Science. Oxford (1996)
6. Boykov, Y., Veksler, O., Zabih, R.: Fast approximate energy minimization via graph cuts. TPAMI **23** (2001) 1222–1239
7. Kolmogorov, V., Zabin, R.: What energy functions can be minimized via graph cuts? TPAMI **26** (2004) 147–159
8. Schlesinger, D.: Exact solution of permuted submodular MinSum problems. In: EMMCVPR. (2007)
9. Shimony, S.E.: Finding MAPs for belief networks is NP-hard. Artificial Intelligence **68** (1994) 399–410
10. Batra, D., Nowozin, S., Kohli, P.: Tighter relaxations for MAP-MRF inference: A local primal-dual gap based separation algorithm. JMLR (Proceedings Track) **15** (2011) 146–154
11. Komodakis, N., Paragios, N., Tziritas, G.: MRF energy minimization and beyond via Dual Decomposition. TPAMI **33** (2011) 531–552
12. Sontag, D., Meltzer, T., Globerson, A., Jaakkola, T., Weiss, Y.: Tightening LP relaxations for MAP using message passing. In: UAI. (2008)
13. Wainwright, M.J., Jordan, M.I.: Graphical Models, Exponential Families, and Variational Inference. Now Publishers Inc., Hanover, MA, USA (2008)
14. Kolmogorov, V.: Convergent tree-reweighted message passing for energy minimization. TPAMI **28** (2006) 1568–1583
15. Wainwright, M.J., Jaakkola, T., Willsky, A.S.: MAP estimation via agreement on trees: message-passing and linear programming. Transactions on Information Theory **51** (2005) 3697–3717
16. Komodakis, N., Paragios, N., Tziritas, G.: MRF energy minimization and beyond via dual decomposition. TPAMI **33** (2011) 531–552
17. Besag, J.: On the statisical analysis of dirty pictures. J. of the Royal Statistical Society B **48** (1986) 259–302
18. Frey, B.J., Jojic, N.: A comparison of algorithms for inference and learning in probabilistic graphical models. TPAMI **27** (2005) 1392–1416
19. Jung, K., Kohli, P., Shah, D.: Local rules for global MAP: When do they work? In: NIPS. (2009)

20. Swendsen, R.H., Wang, J.S.: Nonuniversal critical dynamics in monte carlo simulations. Physical Review Letters **58** (1987) 86–88
21. Avis, D., Fukuda, K.: Reverse search for enumeration. Discrete Appl. Math. **65** (1996) 21–46
22. Moerkotte, G., Neumann, T.: Analysis of two existing and one new dynamic programming algorithm for the generation of optimal bushy join trees without cross products. In: Proc. of the 32nd Int. Conf. on Very Large Data Bases. (2006)