FastDOG: Fast Discrete Optimization on GPU

Ahmed Abbas Paul Swoboda
Max Planck Institute for Informatics, Saarland Informatics Campus

Abstract

We present a massively parallel Lagrange decomposition method for solving 0–1 integer linear programs occurring in structured prediction. We propose a new iterative update scheme for solving the Lagrangean dual and a perturbation technique for decoding primal solutions. For representing subproblems we follow [40] and use binary decision diagrams (BDDs). Our primal and dual algorithms require little synchronization between subproblems and optimization over BDDs needs only elementary operations without complicated control flow. This allows us to exploit the parallelism offered by GPUs for all components of our method. We present experimental results on combinatorial problems from MAP inference for Markov Random Fields, quadratic assignment and cell tracking for developmental biology. Our highly parallel GPU implementation improves upon the running times of the algorithms from [40] by up to an order of magnitude. In particular, we come close to or outperform some state-of-the-art specialized heuristics while being problem agnostic. Our implementation is available at https://github.com/LPMP/BDD.

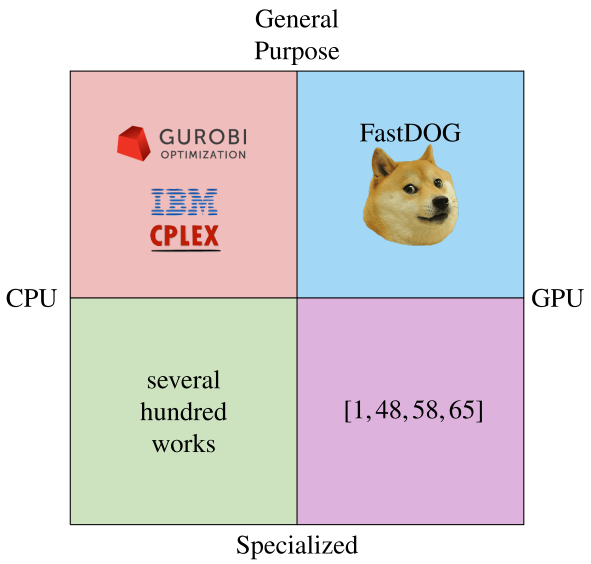


Figure 1. Qualitative comparison of ILP solvers for structured prediction. Our solver (FastDOG) is faster than Gurobi [23] and comparable to specialized CPU solvers, but outperformed by specialized GPU solvers. FastDOG is applicable to a diverse set of applications obviating the human effort for developing solvers for new problem classes.

1. Introduction

Solving integer linear programs (ILP) efficiently on parallel computation devices is an open research question. Done properly it would enable more practical usage of many ILP problems from structured prediction in computer vision and machine learning. Currently, state-of-the-art generally applicable ILP solvers tend not to benefit much from parallelism [45]. In particular, linear program (LP) solvers for computing relaxations benefit modestly (interior point) or not at all (simplex) from multi-core architectures. In particular generally applicable solvers are not amenable for execution on GPUs. To our knowledge there exists no practical and general GPU-based optimization routine and only a few solvers for narrow problem classes have been made GPU-compatible e.g. [1,48,58,65]. This, and the superlinear runtime complexity of general ILP solvers has hindered application of ILPs in large structured prediction problems, necessitating either restriction to at most medium problem sizes or difficult and time-consuming development of specialized solvers as observed for the special case of MAP-MRF [33].

We argue that work on speeding up general purpose ILP solvers has had only limited success so far due to complicated control flow and computation interdependencies. We pursue an overall different approach and do not base our work on the typically used components of ILP solvers. Our approach is designed from the outset to only use operations that offer sufficient parallelism for implementation on GPUs.

We argue that our approach sits on a sweet spot between general applicability and efficiency for problems in structured prediction as shown in Figure 1. Similar to general purpose ILP solvers [15, 23], there is little or no effort to adapt these problems for solving them with our approach. On the other hand we outperform general purpose ILP solvers in terms of execution speed for large problems from structured prediction and achieve runtimes comparable to hand-crafted specialized CPU solvers. We are only significantly outperformed by specialized GPU solvers. However, development of fast specialized solvers especially on GPU is time-consuming and needs to be repeated for every new problem class.

Our work builds upon [40] in which the authors proposed a Lagrange decomposition into subproblems represented by binary decision diagrams (BDD). The authors proposed sequential algorithms as well as parallel extensions for solving the Lagrange decomposition. We improve upon their solver by proposing massively parallelizable GPU amenable routines for both dual optimization and primal rounding. This results in significant runtime improvements as compared to their approach.

2. Related Work

General Purpose ILP Solvers & Parallelism The most efficient implementation of general purpose ILP solvers [15, 23] provided by commercial vendors typically benefit only moderately from parallelism. A recent survey is this direction is given in [45]. The main ways parallelism is utilized in ILP solvers are:

Multiple Independent Executions State-of-the-art solvers [15, 23] offer the option of running multiple algorithms (dual/primal simplex, interior point, different parameters) solving the same problem in parallel until one finds a solution. While easy and worthwhile for problems for which best algorithms and parameters configurations are not known, such a simple approach can deliver parallelization speedups only to a limited degree.

Parallel Branch-and-bound tree traversal While appealing on first glance, it has been observed [46] that the order in which a branch-and-bound tree is traversed is crucial due to exploitation of improved lower and upper bounds and generated cuts. Consequently, it seems hard to obtain significant parallelization speedups and many recent improvements rely on a sequential execution. A separate line of work [50] exploited GPU parallelism for domain propagation allowing to decrease the size of the branch-and-bound tree.

Parallel LP-Solver Interior point methods rely on computing a sequence of solutions of linear systems. This linear algebra can be parallelized for speeding up the optimization [20, 49]. However, for sparse problems sequential simplex solvers still outperform parallelized interior point methods. Also, a crossover step is needed to obtain a suitable basis for the simplex method for reoptimizing for primal rounding and in branch-and-bound searches, limiting the speedup obtainable by this sequential bottleneck. The simplex method is less straightforward to parallelize. The work [28] reports a parallel implementation, however current state-of-the-art commercial solvers outperform it with sequentially executed implementations.

Machine Learning Methods Recently deep learning based methods have been proposed for choosing variables to branch on [17, 43] and for directly computing some easy to guess variables of a solution [43] or improving a given one [51]. While parallelism is not the goal of these works, the underlying deep networks are executed on GPUs and hence the overall computation heavy approach is fast and brings speedups. Still, these parallel components do not replace the sequential parts of the solution process but work in conjunction with them, limiting the overall speedup attainable.

A shortcoming of the above methods in the application to very large structured prediction problems in machine learning and computer vision is that they still do not scale well enough to solve problems with more than a millions variables in a few seconds.

Parallel Combinatorial Solvers For specialized combinatorial problem classes highly parallel algorithms for GPU have been developed. For Maximum-A-Posteriori inference in Markov Random Fields [48, 65] proposed a dual block coordinate ascent algorithm for sparse and [58] for dense graphs. For multicut a primal-dual algorithm has been proposed in [1]. Max-flow GPU implementations have been investigated in [60, 64]. While some parts of the above specialized algorithms can potentially be generalized, other key components cannot, limiting their applicability to new problem classes and requiring time-consuming design of algorithms whenever attempting to solve a different problem class.

Specialized CPU solvers There is a large literature of specialized CPU solvers for specific problem classes in structured prediction. For an overview of pursued algorithmic techniques for the special case of MRFs we refer to the overview article [33]. Most related to our approach are the so called dual block coordinate ascent (a.k.a. message passing) algorithms which optimize a Lagrange decomposition. Solvers have been developed for MRFs [19, 30, 31, 36, 37, 42, 47, 58, 59, 61, 62], graph matching [54, 55, 66], multicut [1, 39, 52], multiple object tracking [27] and cell tracking [24]. Most of the above algorithms require a sequential computation of update steps.

Optimization with Binary Decision Diagrams Our work builds upon [40]. The authors proposed a Lagrange decomposition of ILPs that can be optimized via a sequential

x_i	Optimization variable $i \in [n]$
$\mathcal{X}_j$	Feasible set of constraint $j \in [m]$
$\mathcal{I}_j$	Set of variables in constraint $j \in [m]$
$\mathcal{J}_i$	Set of constraints containing variable $i \in [n]$
m_{ij}^{β}	Min-marginal for variable i taking value β in subproblem $j \in [m]$
λ_i^j	Lagrange multiplier for variable i in subproblem j

Table 1. Notation of symbols used in our problem decomposition.

dual block coordinate ascent method or a decomposition based approach that can utilize multiple CPU cores.

The works [5, 6, 41] similarly consider decompositions into multiple BDDs and solve the resulting problem with general purpose ILP solvers. The work [7] investigates optimization of Lagrange decompositions with multi-valued decision diagrams with subgradient methods. An extension for job sequencing was proposed in [26] and in [14] for routing problems. Hybrid solvers using mixed integer programming solvers were investigated in [21, 22, 56]. The works [3, 8, 9] consider stable set and max-cut and propose optimizing (i) a relaxation to get lower bounds [3] or (ii) a restriction to generate approximate solutions [8, 9].

In contrast to previous BDD-based optimization methods we propose a highly parallelizable and problem agnostic approach that is amenable to GPU computation.

3. Method

We first introduce the optimization problem and its Lagrange decomposition. Next we elaborate our parallel update scheme for optimizing the Lagrangean dual followed by our parallel primal rounding algorithm. For the problem decomposition and dualization we follow [40]. Our notation is summarized for reference in Table 1.

Definition 1 (Binary Program)**.** Consider a linear objective $c \in \mathbb{R}^n$ and m variable subsets $\mathcal{I}_j \subset [n]$ of constraints with feasible set $\mathcal{X}_j \subset \{0,1\}^{\mathcal{I}_j}$ for $j \in [m]$. The corresponding binary program is defined as

$$\min_{x \in \{0,1\}^n} \; c^\top x \quad \text{s.t.} \quad x_{\mathcal{I}_j} \in \mathcal{X}_j \quad \forall j \in [m]\,, \tag{BP}$$

where $x_{\mathcal{I}_j}$ is the restriction to variables in $\mathcal{I}_j$.

Example 1 (ILP)**.** Consider the 0–1 integer linear program

$$\min \quad c^\top x \quad \text{s.t.} \quad Ax \leq b, \; x \in \{0,1\}^n. \tag{ILP}$$

The system of linear constraints $Ax \leq b$ may be split into m blocks, each block representing a single (or multiple) rows of the system. For instance, let $a_j^\top x \leq b_j$ denote the j-th row of $Ax \leq b$, then the problem can be written in the form (BP) by setting $\mathcal{I}_j = \{i \in [n] : a_{ji} \neq 0\}$ and $\mathcal{X}_j = \{x \in \{0,1\}^{\mathcal{I}_j} : \sum_{i \in \mathcal{I}_j} a_{ji} x_i \leq b_j\}$.

3.1. Lagrangean Dual

While (BP) is NP-hard to solve, optimization over a single constraint is typically easier for example by using Binary Decision Diagrams. To make use of this possibility we dualize the original problem using Lagrange decomposition similarly to [40]. This allows us to solve the Lagrangean dual of the full problem (BP) by solving only the subproblems.

Definition 2 (Lagrangean dual problem)**.** Define the set of subproblems that constrain variable x_i as $\mathcal{J}_i = \{j \in [m] \mid i \in \mathcal{I}_j\}$. Let the energy for subproblem $j \in [m]$ w.r.t. Lagrangean dual variables $\lambda^j \in \mathbb{R}^{\mathcal{I}_j}$ be

$$E^j(\lambda^j) = \min_{x \in \mathcal{X}_j} x^\top \lambda^j \,. \tag{1}$$

Then the Lagrangean dual problem is defined as

$$\max_{\lambda} \sum_{j \in [m]} E^j(\lambda^j) \quad \text{s.t.} \quad \sum_{j \in \mathcal{J}_i} \lambda_i^j = c_i \quad \forall i \in [n]. \tag{D}$$

If optima of the individual subproblems $E^j(\lambda^j)$ agree with each other then the consensus vector obtained from stitching together individual subproblem solutions solves the original problem (BP). In general, (D) is a lower bound on (BP). Formal derivation of (D) is given in [40].

3.2. Min-Marginals

To optimize the dual problem (D) and also to obtain a primal solution we use min-marginals [40] defined as

Definition 3 (Min-marginals)**.** For $i \in [n]$, $j \in \mathcal{J}_i$ and $\beta \in \{0,1\}$ let

$$m_{ij}^{\beta} = \min_{x \in \mathcal{X}_j} x^\top \lambda^j \quad \text{s.t.} \quad x_i = \beta \tag{MM}$$

denote the *min-marginal* w.r.t. primal variable i, subproblem j and β.

Definition 4 (Min-marginal differences)**.** For notational convenience let us also define

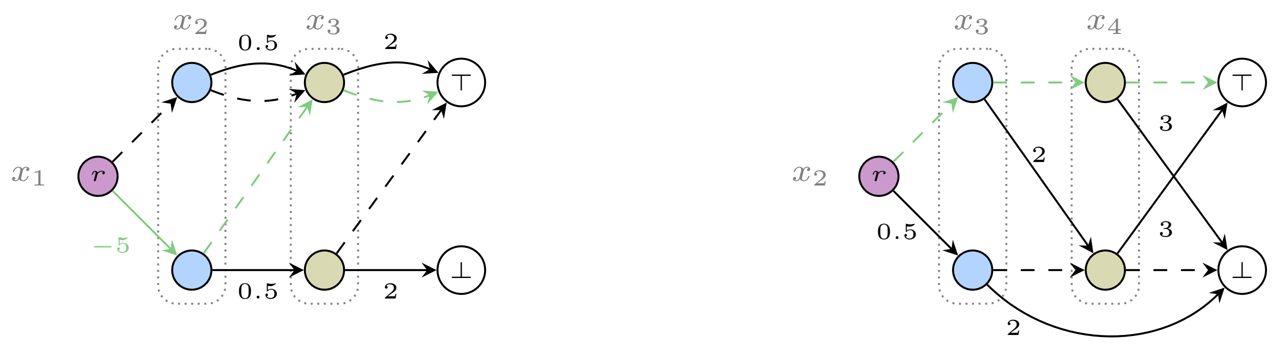
$$M_{ij} = m_{ij}^1 - m_{ij}^0, \tag{MD}$$

which denote *min-marginal difference* computed through (MM).

If $M_{ij} > 0$ then assigning a value of 0 to variable i has a lower cost than assigning a 1 in the subproblem j and viceversa. Thus, the quantity $|M_{ij}|$ indicates by how much $E^j(\lambda^j)$ increases if x_i is fixed to 1 (if $M_{ij} > 0$), respectively 0 (if $M_{ij} < 0$).

Min-marginals have been used in various ways to design dual block coordinate ascent algorithms [1, 4, 19, 24, 27, 30, 31, 36, 37, 40, 42, 47, 52, 54, 55, 58, 59, 61–63, 66].

$$\boxed{\begin{array}{c}\min_{x\in\{0,1\}} -5x_1 + x_2 + 4x_3 + 3x_4 \\ \mathcal{X}_1 : x_1 + x_2 + x_3 \leq 2, \qquad \mathcal{X}_2 : x_2 + x_3 - x_4 = 0\end{array}}$$



$$\lambda^1 = (-5, 0.5, 2),\ (x_1, x_2, x_3) \in \mathcal{X}_1 \qquad \lambda^2 = (0.5, 2, 3),\ (x_2, x_3, x_4) \in \mathcal{X}_2$$

Figure 2. Example decomposition of a binary program into two subproblems, one for each constraint. Each subproblem is represented by a weighted BDD where solid arcs model the cost λ of assigning a 1 to the variable and dashed arcs have 0 cost which model assigning a 0. All $r - \top$ paths in BDDs encode feasible variable assignments of corresponding subproblems (and $r - \perp$ infeasible). Optimal assignments w.r.t current (non-optimal) λ are highlighted in green i.e. $x_1 = 1, x_2 = x_3 = 0$ for $\mathcal{X}_1$ and $x_2 = x_3 = x_4 = 0$ for $\mathcal{X}_2$. Our dual update scheme processes multiple variables in parallel which are indicated in same color (e.g. x_1, x_2 in $\mathcal{X}_1, \mathcal{X}_2$ resp.).

Algorithm 1: Parallel Deferred Min-Marg. Averaging

```
Input: Lagrange variables $\lambda_i^j \in \mathbb{R}\ \forall i \in [n], j \in \mathcal{J}_i$,
       Constraint sets $\mathcal{X}_j \subset \{0,1\}^{\mathcal{I}_j}\ \forall j \in [m]$,
       Damping factor $\omega \in (0,1]$
1  Initialize deferred min-marginal diff. $\overline{M} = \mathbb{0}$
2  while (stopping criterion not met) do
3      for $j \in \mathcal{J}$ in parallel do
4          for $i \in \mathcal{I}_j$ in ascending order do
5              Compute min-marginal diff. $M_{ij}$ (MD)
6              Update dual variables $\lambda_i^j$ via (3)
7      Update deferred min-marginal diff. $\overline{M} \leftarrow M$
8      Repeat lines 3-6 in descending order of $\mathcal{I}_j$
9  for $j \in \mathcal{J}, i \in \mathcal{I}_j$ do
10     Add deferred min-marginal differences:
         $\lambda_i^j \mathrel{+}= \omega \overline{M}_{ij}$
```

3.3. Parallel Deferred Min-Marginal Averaging

To exploit GPU parallelism in solving the dual problem (D) we would like to update multiple dual variables in parallel. However, conventional dual update schemes are not friendly for parallelization. For example the dual update scheme of [40] for variable i in subproblem j is

$$\lambda_i^j \leftarrow \lambda_i^j - M_{ij} + \underbrace{\frac{1}{|\mathcal{J}_i|}\sum_{k\in\mathcal{J}_i} M_{ik}}_{\text{min-marginal averaging}}, \tag{2}$$

where M_{ij} is defined in (MD). This update scheme (2) requires communication between all subproblems $\mathcal{J}_i$ containing variable i for the min-marginal averaging step and thus requires synchronization. To overcome this limitation we propose a novel dual optimization procedure which performs this averaging step on min-marginal differences $\overline{M}$ from the previous iteration as follows

$$\lambda_i^j \leftarrow \lambda_i^j - \omega M_{ij} + \frac{\omega}{|\mathcal{J}_i|}\sum_{k\in\mathcal{J}_i} \overline{M}_{ik}. \tag{3}$$

Since $\overline{M}$ was computed in the previous iteration, the above dual updates can be performed in parallel for all subproblems without requiring synchronization. Following [63] we use a damping factor $\omega \in (0,1)$ (0.5 in our experiments) to obtain better final solutions.

Our proposed scheme is given in Algorithm 1. We iterate in parallel over each subproblem j. For each subproblem, variables are visited in order and min-marginals are computed and stored for updates in the next iteration (lines 4-5). The current min-marginal difference is subtracted and the one from previous iteration is added (line 6) by distributing it equally among subproblems $\mathcal{J}_i$. At termination (line 10) we perform a min-marginal averaging step to account for the deferred update from last iteration. For stopping criteria we use relative change in dual objective between two subsequent iterations. We initialize the input Lagrange variables by $\lambda_i^j = c_i/|\mathcal{J}_i|, \forall i \in [n], j \in \mathcal{J}_i$.

Proposition 1. *In each dual iteration the Lagrange multipliers along with the deferred min-marginals can be used* *to satisfy dual feasibility and the dual lower bound* (D) *is non-decreasing.*

Similar to other dual block coordinate ascent schemes Algorithm 1 can get stuck in suboptimal points, see [62, 63]. As seen in our experiments these are usually not far away from the optimum, however.

In Section 5 we will explain how we can incrementally compute min-marginals reusing previous computations if we represent subproblems as Binary Decision Diagrams. This saves us from computing min-marginals from scratch leading to greater efficiency.

4. Primal Rounding

In order to obtain a primal solution to (BP) from an approximative dual solution to (D) we propose a GPU friendly primal rounding scheme based on cost perturbation. We iteratively change costs in a way that variable assignments across subproblems agree with each other. If all variables agree by favoring a single assignment, we can reconstruct a primal solution (not necessarily the optimal). Instead of only using variable assignments of all subproblems we use min-marginal differences (MD) as they additionally indicate how strongly a variable favours a particular assignment.

Algorithm 2 details our method. We iterate over all variables in parallel and check min-marginal differences. If for a variable i all min-marginal differences indicate that the optimal solution is 0 (resp. 1) Lagrange variables λ are increased (resp. decreased) leaving even more certain min-marginals differences for these variables. This step imitates variable fixation as done in branch-and-bound, however we only perform soft fixation implicitly through cost perturbation. In case min-marginal differences are equal we randomly perturb corresponding dual costs. Lastly, if min-marginals differences indicate conflicting solutions we compute total min-marginal difference and decide accordingly. In the last two cases we add more perturbation to force towards non-conflicts. For faster convergence we increase the perturbation magnitude after each iteration.

Note that the modified λ variables via Alg. 2 need not be feasible for the dual problem (D). Although, our primal rounding algorithm is not guaranteed to terminate, in our experiments a solution was always found in less than 100 iterations.

Remark. The primal rounding scheme in [40] and typical primal ILP heuristics [10] are sequential and build upon sequential operations such as variable propagation. Our primal rounding lends itself to parallelism since we perturb costs on all variables simultaneously and reoptimize via Algorithm 1.

Algorithm 2: Perturbation Primal Rounding

```
Input: Lagrange variables $\lambda_i^j \in \mathbb{R}\, \forall i \in [n], j \in \mathcal{J}_i$,
       Constraint sets $\mathcal{X}_j \subset \{0,1\}^{\mathcal{I}_j}\, \forall j \in [m]$,
       Initial perturbation strength $\delta \in \mathbb{R}_+$,
       perturbation growth rate $\alpha$
Output: Feasible labeling $x \in \{0,1\}^n$
1  Compute min-marginal differences $M_{ij}\, \forall i, j$ (MD)
2  while $\exists i \in [n]$ and $j \neq k \in \mathcal{J}_i$ s.t.
     $\text{sign}(M_{ij}) \neq \text{sign}(M_{ik})$ do
3      for $i = 1, \ldots, n$ in parallel do
4          Sample $r$ uniformly from $[-\delta, \delta]$
5          if $M_{ij} > 0\, \forall j \in \mathcal{J}_i$ then
6              $\lambda_i^j \mathrel{+}= \delta \quad \forall j \in \mathcal{J}_i$
7          else if $M_{ij} < 0\, \forall j \in \mathcal{J}_i$ then
8              $\lambda_i^j \mathrel{-}= \delta \quad \forall j \in \mathcal{J}_i$
9          else if $M_{ij} = 0\, \forall j \in \mathcal{J}_i$ then
10             $\lambda_i^j \mathrel{+}= r \cdot \delta \quad \forall j \in \mathcal{J}_i$
11         else
12             Compute total min-marginal difference:
               $M_i = \sum_{j \in \mathcal{J}_i} M_{ij}$
13             $\lambda_i^j \mathrel{+}= \text{sign}(M_i) \cdot |r| \cdot \delta \quad \forall j \in \mathcal{J}_i$
14     Increase perturbation: $\delta \leftarrow \delta \cdot \alpha$
15     Reoptimize perturbed $\lambda$ via Algorithm 1
16     Recompute $M_{ij}\, \forall i, j$ w.r.t optimized $\lambda$
```

5. Binary Decision Diagrams

We use Binary Decision Diagrams (BDDs) to represent the feasibility sets $\mathcal{X}_j$, $j \in [m]$ and compute their min-marginals (MM). BDDs are in essence directed acyclic graphs whose paths between two special nodes (root and terminal) encode all feasible solutions. Specifically, we use reduced ordered Binary Decision Diagrams [12] as in [40].

Definition 5 (BDD)**.** Let an ordered variable set $\mathcal{I} = \{w_1, \ldots, w_k\} \subset [n]$ corresponding to a constraint be given. A corresponding BDD is a directed acyclic graph $D = (V, A)$ with

Special nodes: root node r, terminals $\bot$ and $\top$.

Outgoing Arcs: each node $v \in V \backslash \{\top, \bot\}$ has exactly two successors $s^0(v), s^1(v)$ with outgoing arcs $vs^0(v) \in A$ (the zero arc) and $vs^1(v) \in A$ (the one arc).

Partition: the node set V is partitioned by $\{\mathcal{P}_1, \ldots, \mathcal{P}_k\}$, $\dot\cup_i \mathcal{P}_i = V \backslash \{\top, \bot\}$. Each partition holds all the nodes corresponding to a single variable *e.g.* $\mathcal{P}_i$ corresponds to variable w_i. It holds that $\mathcal{P}_1 = \{r\}$ *i.e.* it only contains the root node.

Partition Ordering: when $v \in \mathcal{P}_i$ then $s^0(v), s^1(v) \in \mathcal{P}_{i+1} \cup \{\bot\}$ for $i < k$ and $s^0(v), s^1(v) \in \{\bot, \top\}$ for $v \in \mathcal{P}_k$.

Definition 6 (Constraint Set Correspondence)**.** Each BDD defines a constraint set $\mathcal{X}$ via the relation

$$x \in \mathcal{X} \Leftrightarrow \begin{array}{c} \exists(v_1,\ldots,v_k,v_{k+1}) \in \text{Paths}(V,A) \text{ s.t.} \\ v_1 = r, v_{k+1} = \top, \\ v_{i+1} = s^{x_i}(v_i)\, \forall i \in [k] \end{array} \tag{4}$$

Thus each path between root r and terminal $\top$ in the BDD corresponds to some feasible variable assignment $x \in \mathcal{X}$.

Figures 2 and 3 illustrate BDD encoding of feasible sets of linear inequalities.

Remark. In the literature [12, 35] BDDs have additional requirements, mainly that there are no isomorphic subgraphs. This allows for some additional canonicity properties like uniqueness and minimality. While all the BDDs in our algorithms satisfy the additional canonicity properties, only what is required in Definition 5 is needed for our purposes, so we keep this simpler setting.

5.1. Efficient Min-Marginal Computation

In order to compute min-marginals for subproblems we need to consider weighted BDDs. For notational convenience we will drop dependence on subproblem j in upcoming text *e.g.*, we will use λ_i instead of λ_i^j.

Definition 7 (Weighted BDD)**.** A weighted BDD is a BDD with arc costs. Let a function $f(x)$ be defined as

$$f(x) = \begin{cases} x^\top \lambda & x \in \mathcal{X} \\ \infty & \text{otherwise} \end{cases}. \tag{5}$$

The weighted BDD represents f if it satisfies Def. 6 for the given $\mathcal{X}$ and the arc costs for an $i \in [k], v \in \mathcal{P}_i, vw \in A$ are set as $\begin{cases} 0 & w \in s^0(v) \\ \lambda_i & w \in s^1(v) \end{cases}$.

Min-marginals for variable $i \in \mathcal{I}$ of a subproblem can be computed by its weighted BDD by calculating shortest path distances from r to all nodes in $\mathcal{P}_i$ and shortest path distances from all nodes in $\mathcal{P}_{i+1}$ to $\top$. We use $\text{SP}(v, w)$ to denote the shortest path distance between nodes v and w of a weighted BDD. An example shortest path calculation is shown in Figure 3. The min-marginals as defined in (MM) can be computed as

$$m_i^\beta = \min_{\substack{vs^\beta(v) \in A \\ v \in \mathcal{P}_i}} \left[\text{SP}(r, v) + \beta \cdot \lambda_i + \text{SP}(s^\beta(v), \top)\right] \tag{6}$$

For efficient min-marginal computation in Algorithm 1 we reuse shortest path distances used in (6). Specifically, for computing min-marginals in lines 4-5 of Alg. 1 we use Alg. 3 for ascending variable order and Alg. 4 for descending variable order in line 8.

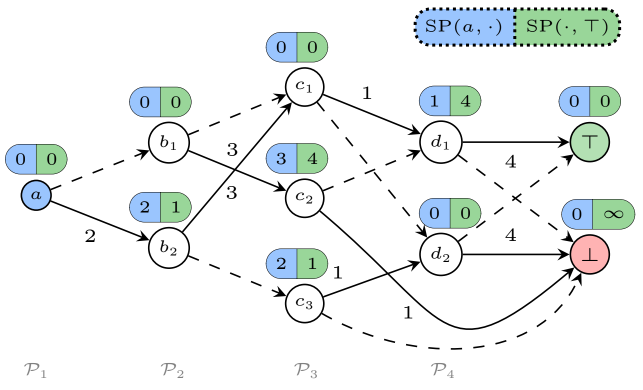


Figure 3. Weighted BDD of a subproblem containing variables: $\mathcal{I} = \{a, b, c, d\}$ with costs (λ): $2, 3, 1, 4$ resp. and constraint $a - b - c + d = 0$. Shortest path costs from the root node a and the terminal node $\top$ are shown for each node. Here $\mathcal{P}_1 = \{a\}, \mathcal{P}_2 = \{b_1, b_2\}, \mathcal{P}_3 = \{c_1, c_2, c_3\}, \mathcal{P}_4 = \{d_1, d_2\}$, $s^0(c_2) = d_1$ and $s^1(c_2) = \perp$. Dashed arcs have cost 0 as they model assigning a 0 value to the corresponding variable.

Algorithm 3: Forward Pass Min-Marginal Computation

```
1 for $v \in \mathcal{P}_i$ do
2     $\text{SP}(r,v) = \min \left\{ \begin{array}{c} \min_{u:s^0(u)=v} \text{SP}(r,u), \\ \min_{u:s^1(u)=v} \text{SP}(r,u) + \lambda_i \end{array} \right\}$
3 Compute $m_i^\beta$ via (6)
```

Algorithm 4: Backward Pass Min-Marginal Computation

```
1 for $v \in \mathcal{P}_{i+1}$ do
2     $\text{SP}(v,\top) = \min \left\{ \begin{array}{c} \text{SP}(s^0(v),\top), \\ \text{SP}(s^1(v),\top) + \lambda_{i+1} \end{array} \right\}$
3 Compute $m_i^\beta$ via (6)
```

Efficient GPU implementation In addition to solving all subproblems in parallel, we also exploit parallelism within each subproblem during shortest path updates. Specifically in Alg. 3, we parallelize over all $v \in \mathcal{P}_i$ and perform the min operation atomically. Similarly in Alg. 4 we parallelize over all $v \in \mathcal{P}_{i+1}$ but without requiring atomic update.

To enable fast GPU memory access via memory coalescing we arrange BDD nodes in the following fashion. First, all nodes within a BDD which belong to the same partition $\mathcal{P}$ (thus corresponding to same variable) are laid out consecutively. Secondly, across different BDDs, nodes are ordered w.r.t increasing hop distance from their corresponding root nodes. Such arrangement for the ILP in Figure 2 is shown in Figure 4.

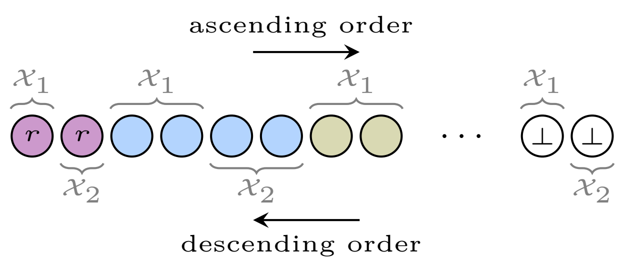


Figure 4. Arrangement of BDD nodes in GPU memory for the ILP in Figure 2. For ascending order in Alg. 1 we proceed from root to terminal nodes and vice versa for descending.

6. Experiments

We show the effectiveness of our solver against a state-of-the-art ILP solver [23], the general purpose BDD-based solver [40] and specialized CPU solvers for specific problem classes. We have chosen some of the largest structured prediction ILPs we are aware of in the literature that are publicly available. Our results are computed on a single NVIDIA Volta V100 (16GB) GPU unless stated otherwise. For CPU solvers we use AMD EPYC 7702 CPU.

Datasets Our benchmark problems obtained from [53] can be categorized as follows.

Cell tracking: Instances from [24] which we partition into small and large instances as also done in [40].

Graph matching (GM): Quadratic assignment problems (often called graph matching in the literature) for correspondence in computer vision [57] (`hotel`, `house`) and developmental biology [32] (`worms`).

Markov Random Field (MRF): Several datasets from the OpenGM [33] benchmark, containing both small and large instances with varying topologies and number of labels. We have chosen the datasets *color-seg*, *color-seg-n4*, *color-seg-n8* and *object-seg*.

QAPLib: The widely used benchmark dataset for quadratic assignment problems used in the combinatorial optimization community [13]. We partition QAPLib instances into small (up to 50 vertices) and large (up to 128 vertices) instances. For large instances we use NVIDIA RTX 8000 (48GB) GPU.

Algorithms We compare results on the following algorithms.

`Gurobi`: The commercial ILP solver [23] as reported in [40]. The barrier method is used for QAPLib and dual simplex for all other datasets.

`BDD-CPU`: BDD-based min-marginal averaging approach of [40]. The algorithm runs on CPU with 16 threads for parallelization. Primal solutions are rounded using their BDD-based depth-first search scheme.

`Specialized solvers`: State-of-the-art problem specific solver for each dataset. For cell-tracking we use the solver from [24], the `AMP` solver for graph matching proposed in [55] and `TRWS` for MRF [36].

`FastDOG`: Our approach where for the GPU implementation we use the CUDA [44] and Thrust [25] programming frameworks. For rounding primal solutions with Algorithm 2 we set $\delta = 1.0$ and $\alpha = 1.2$. For constructing BDDs out of linear (in)equalities we use the same approach as for `BDD-CPU`.

For *MRF*, parallel algorithms such as [58] exist however `TRWS` is faster on the sparse problems we consider. While we are aware of even faster purely primal heuristics [11, 38] for *MRF* and *e.g.* [29] for *graph matching* they do not optimize a convex relaxation and hence do not provide lower bounds. Hence, we have chosen `TRWS` [36] for *MRF* and `AMP` [55] for *graph matching* which, similar to `FastDOG`, optimize an equivalent resp. similar Lagrange decomposition and hence can be directly compared.

Results In Table 2 we show aggregated results over all instances of each specific benchmark dataset. Runtimes are taken w.r.t. computation of both primal and dual bounds. A more detailed table with results for each instance is given in the Appendix.

In Figure 5 we show averaged convergence plots for various solvers. In general we offer a very good anytime performance producing at most times and in general during the beginning better lower bounds than our baselines.

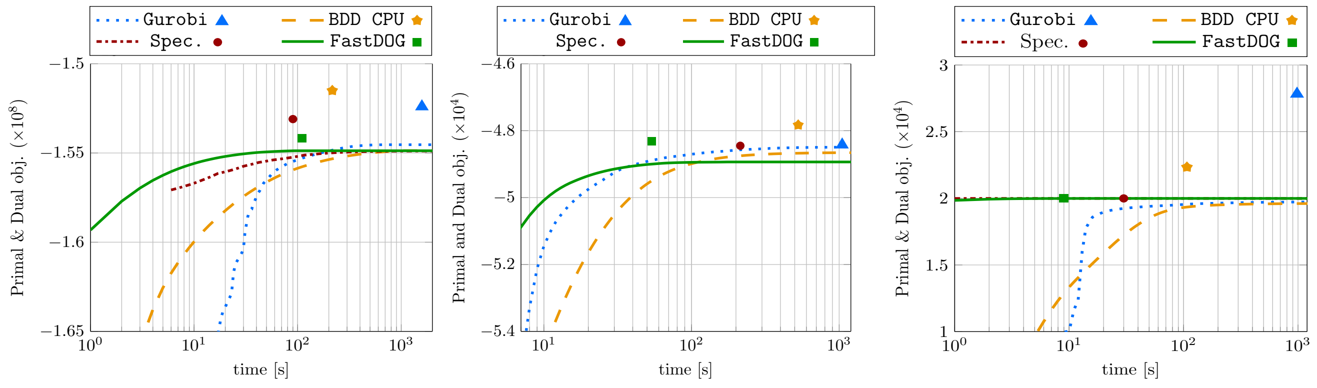
Discussion In general, we are always faster (up to a factor of 10) than `BDD-CPU` [40] and except on *worms* we achieve similar or better lower bounds. In comparison to the respective hand-crafted `Specialized` CPU solvers we also achieve comparable runtimes with comparable lower and upper bounds. While `Gurobi` achieves, if given unlimited time, better lower bounds and primal solutions, our `FastDOG` solver outperforms it on the larger instances when we abort `Gurobi` early after hitting a time limit. We argue that we outperform `Gurobi` on larger instances due to its superlinear iteration complexity.

When comparing the number of dual iterations to `BDD-CPU` we need roughly 3-times as many to reach the same lower bound. Nonetheless, as we can perform more iterations per second this still leads to an overall faster algorithm.

Since we are solving a relaxation the lower bounds and quality of primal solutions are dependent on the tightness of this relaxation. For all datasets except *QAPLib* our (and also baselines') lower and upper bounds are fairly close, reflecting the nature of commonly occurring structured prediction problems.

	Cell tracking		*Graph matching*			*MRF*				*QAPLib*	
	Small	*Large*	*Hotel*	*House*	*Worms*	*C-seg*	*C-seg-n4*	*C-seg-n8*	*Obj-seg*	*Small*	*Large*
# instances	10	5	105	105	30	3	9	9	5	105	29
n_{max}	1.2M	10M	0.3M	0.3M	1.5M	3.3M	1.2M	1.4M	681k	3M	49M
m_{max}	0.2M	2.3M	52k	52k	0.2M	13.6M	4.2M	8.3M	2.2M	245k	2M
Dual objective (lower bound) ↑											
`Gurobi` [23]	**−4.382e6**	**−1.545e8**	**−4.293e3**	**−3.778e3**	−4.849e4	**3.085e8**	1.9757e4	1.9729e4	3.1311e4	2.913e6	4.512e4
`BDD-CPU` [40]	−4.387e6	−1.549e8	**−4.293e3**	**−3.778e3**	−4.878e4	3.085e8	1.9643e4	1.9631e4	3.1248e4	3.675e6	8.172e6
`Specialized`	−4.385e6	−1.551e8	**−4.293e3**	**−3.778e3**	**−4.847e4**	**3.085e8**	**2.0012e4**	**1.9991e4**	**3.1317e4**	-	-
`FastDOG`	−4.387e6	−1.549e8	**−4.293e3**	**−3.778e3**	−4.893e4	3.085e8	2.0011e4	1.9990e4	**3.1317e4**	**3.747e6**	**8.924e6**
Primal objective (upper bound) ↓											
`Gurobi` [23]	**−4.382e6**	−1.524e8	**−4.293e3**	**−3.778e3**	−4.842e4	**3.085e8**	2.8464e4	2.7829e4	1.4981e5	5.186e7	1.431e8
`BDD-CPU` [40]	−4.337e6	−1.515e8	**−4.293e3**	**−3.778e3**	−4.783e4	3.086e8	2.1781e4	2.2338e4	3.1525e4	5.239e7	1.452e8
`Specialized`	−4.361e6	−1.531e8	**−4.293e3**	**−3.778e3**	**−4.845e4**	**3.085e8**	**2.0012e4**	**1.9991e4**	**3.1317e4**	-	-
`FastDOG`	−4.376e6	**−1.541e8**	**−4.293e3**	**−3.778e3**	−4.831e4	3.085e8	2.0016e4	1.9995e4	3.1322e4	**4.330e7**	**1.376e8**
Runtimes [s] ↓											
`Gurobi` [23]	**1**	1584	4	7	1048	132	980	1337	1506	3948	6742
`BDD-CPU` [40]	14	216	6	12	528	70	107	218	232	357	**5952**
`Specialized`	1.5	**90**	3	3	214	155	**9**	30	**3**	-	-
`FastDOG`	13	110	**0.2**	**0.4**	**54**	**14**	**9**	**13**	39	137	6928

Table 2. Results comparison on all datasets where the values are averaged within a dataset. For each dataset, the results on corresponding specialized solvers are computed using [24, 34, 55]. Numbers in bold highlight the best performance. n_{max}, m_{max}: Maximum number of variables, constraints in the category.



(a) *Cell tracking: Large* (b) *Graph Matching: Worms* (c) *MRF: Color-seg-n8*

Figure 5. Convergence plots averaged over all instances of a dataset. Lower curves depict increasing lower bounds while markers denote objectives of rounded primal solutions. The x-axis is plotted logarithmically.

7. Conclusion

We have proposed a massively parallelizable generic algorithm that can solve a wide variety of ILPs on GPU. Our results indicate that the performance of specialized efficient CPU solvers can be matched or even surpassed by a completely generic GPU solver. Our implementation is a first prototype and we conjecture that more speedups can be gained by elaborate implementation techniques, *e.g.* compression of the BDD representation, better memory layout for better memory coalescing, multi-GPU support etc. We argue that future improvements in optimization algorithms for structured prediction can be made by developing GPU friendly problem specific solvers and with improvements in our or other generic GPU solvers that can benefit many problem classes simultaneously. Another future avenue is optimization of ILPs from other domains, *e.g.* on the MIPLib benchmark [18]. These problems include constraints that are harder to represent as BDDs and additional encoding techniques are needed [2, 16].

8. Acknowledgments

We would like to thank all reviewers, especially Reviewer 1 for valuable feedback and Jan-Hendrik Lange for insightful discussions.

References

[1] Ahmed Abbas and Paul Swoboda. RAMA: A Rapid Multicut Algorithm on GPU. *arXiv preprint arXiv:2109.01838*, 2021. 1, 2, 3

[2] Ignasi Abío, Robert Nieuwenhuis, Albert Oliveras, Enric Rodríguez-Carbonell, and Valentin Mayer-Eichberger. A new look at bdds for pseudo-boolean constraints. *Journal of Artificial Intelligence Research*, 45:443–480, 2012. 8

[3] Henrik Reif Andersen, Tarik Hadzic, John N Hooker, and Peter Tiedemann. A constraint store based on multivalued decision diagrams. In *International Conference on Principles and Practice of Constraint Programming*, pages 118–132. Springer, 2007. 3

[4] Chetan Arora and Amir Globerson. Higher order matching for consistent multiple target tracking. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 177–184, 2013. 3

[5] David Bergman and Andre A. Cire. Decomposition based on decision diagrams. In Claude-Guy Quimper, editor, *Integration of AI and OR Techniques in Constraint Programming*, pages 45–54, Cham, 2016. Springer International Publishing. 3

[6] David Bergman and Andre A Cire. Discrete nonlinear optimization by state-space decompositions. *Management Science*, 64(10):4700–4720, 2018. 3

[7] David Bergman, Andre A Cire, and Willem-Jan van Hoeve. Lagrangian bounds from decision diagrams. *Constraints*, 20(3):346–361, 2015. 3

[8] David Bergman, Andre A Cire, Willem-Jan Van Hoeve, and John Hooker. *Decision diagrams for optimization*, volume 1. Springer, 2016. 3

[9] David Bergman, Andre A Cire, Willem-Jan van Hoeve, and John N Hooker. Discrete optimization with decision diagrams. *INFORMS Journal on Computing*, 28(1):47–66, 2016. 3

[10] Timo Berthold. Primal heuristics for mixed integer programs. 2006. 5

[11] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on pattern analysis and machine intelligence*, 23(11):1222–1239, 2001. 7

[12] Randal E Bryant. Graph-based algorithms for boolean function manipulation. *Computers, IEEE Transactions on*, 100(8):677–691, 1986. 5, 6

[13] Rainer E Burkard, Stefan E Karisch, and Franz Rendl. QAPLIB–a quadratic assignment problem library. *Journal of Global optimization*, 10(4):391–403, 1997. 7

[14] Margarita P Castro, Andre A Cire, and J Christopher Beck. An mdd-based lagrangian approach to the multicommodity pickup-and-delivery tsp. *INFORMS Journal on Computing*, 32(2):263–278, 2020. 3

[15] Cplex, IBM ILOG. CPLEX optimization studio 12.10, 2019. 1, 2

[16] M. Fujita, Y. Lu, E. Clarke, and J. Jain. Efficient variable ordering using abdd based sampling. In *Design Automation Conference*, pages 687–692, Los Alamitos, CA, USA, jun 2000. IEEE Computer Society. 8

[17] Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. *arXiv preprint arXiv:1906.01629*, 2019. 2

[18] Ambros Gleixner, Gregor Hendel, Gerald Gamrath, Tobias Achterberg, Michael Bastubbe, Timo Berthold, Philipp M. Christophel, Kati Jarck, Thorsten Koch, Jeff Linderoth, Marco Lübbecke, Hans D. Mittelmann, Derya Ozyurt, Ted K. Ralphs, Domenico Salvagnin, and Yuji Shinano. MIPLIB 2017: Data-Driven Compilation of the 6th Mixed-Integer Programming Library. *Mathematical Programming Computation*, 2021. 8

[19] Amir Globerson and Tommi S Jaakkola. Fixing max-product: Convergent message passing algorithms for MAP LP-relaxations. In *Advances in neural information processing systems*, pages 553–560, 2008. 2, 3

[20] Jacek Gondzio and Robert Sarkissian. Parallel interior-point solver for structured linear programs. *Mathematical Programming*, 96(3):561–584, 2003. 2

[21] Jaime E González, Andre A Cire, Andrea Lodi, and Louis-Martin Rousseau. BDD-based optimization for the quadratic stable set problem. *Discrete Optimization*, page 100610, 2020. 3

[22] Jaime E González, Andre A Cire, Andrea Lodi, and Louis-Martin Rousseau. Integrated integer programming and decision diagram search tree with an application to the maximum independent set problem. *Constraints*, pages 1–24, 2020. 3

[23] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2021. 1, 2, 7, 8

[24] Stefan Haller, Mangal Prakash, Lisa Hutschenreiter, Tobias Pietzsch, Carsten Rother, Florian Jug, Paul Swoboda, and Bogdan Savchynskyy. A primal-dual solver for large-scale tracking-by-assignment. In *AISTATS*, 2020. 2, 3, 7, 8

[25] Jared Hoberock and Nathan Bell. Thrust: A parallel template library, 2010. Version 1.7.0. 7

[26] John N. Hooker. Improved job sequencing bounds from decision diagrams. In Thomas Schiex and Simon de Givry, editors, *Principles and Practice of Constraint Programming*, pages 268–283, Cham, 2019. Springer International Publishing. 3

[27] Andrea Hornakova, Timo Kaiser, Paul Swoboda, Michal Rolinek, Bodo Rosenhahn, and Roberto Henschel. Making higher order MOT scalable: An efficient approximate solver for lifted disjoint paths. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6330–6340, 2021. 2, 3

[28] Qi Huangfu and J. A. J. Hall. Parallelizing the dual revised simplex method. *Math. Program. Comput.*, 10(1):119–142, 2018. 2

[29] Lisa Hutschenreiter, Stefan Haller, Lorenz Feineis, Carsten Rother, Dagmar Kainmüller, and Bogdan Savchynskyy. Fusion moves for graph matching. 2021. 7

[30] Jeremy Jancsary and Gerald Matz. Convergent decomposition solvers for tree-reweighted free energies. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 388–398, 2011. 2, 3

[31] Jason K Johnson, Dmitry M Malioutov, and Alan S Willsky. Lagrangian relaxation for MAP estimation in graphical models. *arXiv preprint arXiv:0710.0013*, 2007. 2, 3

[32] Dagmar Kainmueller, Florian Jug, Carsten Rother, and Gene Myers. Active graph matching for automatic joint segmentation and annotation of C. elegans. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 81–88. Springer, 2014. 7

[33] Jörg H. Kappes, Björn Andres, Fred A. Hamprecht, Christoph Schnörr, Sebastian Nowozin, Dhruv Batra, Sungwoong Kim, Bernhard X. Kausler, Thorben Kröger, Jan Lellmann, Nikos Komodakis, Bogdan Savchynskyy, and Carsten Rother. A comparative study of modern inference techniques for structured discrete energy minimization problems. *International Journal of Computer Vision*, 115(2):155–184, 2015. 1, 2, 7

[34] Jörg Hendrik Kappes, Markus Speth, Gerhard Reinelt, and Christoph Schnörr. Towards efficient and exact MAP-inference for large scale discrete computer vision problems via combinatorial optimization. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1752–1758, 2013. 8

[35] Donald E Knuth. *The art of computer programming, volume 4A: combinatorial algorithms, part 1*. Pearson Education India, 2011. 6

[36] Vladimir Kolmogorov. Convergent tree-reweighted message passing for energy minimization. *IEEE transactions on pattern analysis and machine intelligence*, 28(10):1568–1583, 2006. 2, 3, 7

[37] Vladimir Kolmogorov. A new look at reweighted message passing. *IEEE transactions on pattern analysis and machine intelligence*, 37(5):919–930, 2014. 2, 3

[38] Nikos Komodakis and Georgios Tziritas. Approximate labeling via graph cuts based on linear programming. *IEEE transactions on pattern analysis and machine intelligence*, 29(8):1436–1453, 2007. 7

[39] Jan-Hendrik Lange, Andreas Karrenbauer, and Bjoern Andres. Partial optimality and fast lower bounds for weighted correlation clustering. In *International Conference on Machine Learning*, pages 2892–2901. PMLR, 2018. 2

[40] Jan-Hendrik Lange and Paul Swoboda. Efficient message passing for 0–1 ILPs with binary decision diagrams. In *International Conference on Machine Learning*, pages 6000–6010. PMLR, 2021. 1, 2, 3, 4, 5, 7, 8

[41] Leonardo Lozano, David Bergman, and J Cole Smith. On the consistent path problem. *Optimization Online e-prints*, 2018. 3

[42] Talya Meltzer, Amir Globerson, and Yair Weiss. Convergent message passing algorithms-a unifying view. *arXiv preprint arXiv:1205.2625*, 2012. 2, 3

[43] Vinod Nair, Sergey Bartunov, Felix Gimeno, Ingrid von Glehn, Pawel Lichocki, Ivan Lobov, Brendan O'Donoghue, Nicolas Sonnerat, Christian Tjandraatmadja, Pengming Wang, et al. Solving mixed integer programs using neural networks. *arXiv preprint arXiv:2012.13349*, 2020. 2

[44] NVIDIA, Péter Vingelmann, and Frank H.P. Fitzek. CUDA, release: 11.2, 2021. 7

[45] Kalyan Perumalla and Maksudul Alam. Design Considerations for GPU-Based Mixed Integer Programming on Parallel Computing Platforms. Association for Computing Machinery, New York, NY, USA, 2021. 1, 2

[46] Ted Ralphs, Yuji Shinano, Timo Berthold, and Thorsten Koch. Parallel solvers for mixed integer linear optimization. In *Handbook of parallel constraint reasoning*, pages 283–336. Springer, 2018. 2

[47] B. Savchynskyy, S. Schmidt, Jörg H. Kappes, and Christoph Schnörr. Efficient MRF energy minimization via adaptive diminishing smoothing. *UAI. Proceedings*, pages 746–755, 2012. 1. 2, 3

[48] Alexander Shekhovtsov, Christian Reinbacher, Gottfried Graber, and Thomas Pock. Solving dense image matching in real-time using discrete-continuous optimization. In *Proceedings of the 21st Computer Vision Winter Workshop (CVWW)*, page 13, 2016. 1, 2

[49] Edmund Smith, Jacek Gondzio, and Julian Hall. GPU acceleration of the matrix-free interior point method. In Roman Wyrzykowski, Jack Dongarra, Konrad Karczewski, and Jerzy Waśniewski, editors, *Parallel Processing and Applied Mathematics*, pages 681–689, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. 2

[50] Boro Sofranac, Ambros Gleixner, and Sebastian Pokutta. Accelerating domain propagation: an efficient GPU-parallel algorithm over sparse matrices. In *2020 IEEE/ACM 10th Workshop on Irregular Applications: Architectures and Algorithms (IA3)*, pages 1–11. IEEE, 2020. 2

[51] Nicolas Sonnerat, Pengming Wang, Ira Ktena, Sergey Bartunov, and Vinod Nair. Learning a large neighborhood search algorithm for mixed integer programs. *arXiv preprint arXiv:2107.10201*, 2021. 2

[52] Paul Swoboda and Bjoern Andres. A message passing algorithm for the minimum cost multicut problem. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1617–1626, 2017. 2, 3

[53] Paul Swoboda, Andrea Hornakova, Paul Roetzer, and Ahmed Abbas. Structured prediction problem archive. *arXiv preprint arXiv:2202.03574*, 2022. 7

[54] Paul Swoboda, Ashkan Mokarian, Christian Theobalt, Florian Bernard, et al. A convex relaxation for multi-graph matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 11156–11165, 2019. 2, 3

[55] Paul Swoboda, Carsten Rother, Hassan Abu Alhaija, Dagmar Kainmuller, and Bogdan Savchynskyy. A study of lagrangean decompositions and dual ascent solvers for graph matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1607–1616, 2017. 2, 3, 7, 8

[56] Christian Tjandraatmadja and Willem-Jan van Hoeve. Incorporating bounds from decision diagrams into integer programming. *Mathematical Programming Computation*, pages 1–32, 2020. 3

[57] Lorenzo Torresani, Vladimir Kolmogorov, and Carsten Rother. Feature correspondence via graph matching: Models and global optimization. In *European conference on computer vision*, pages 596–609. Springer, 2008. 7

[58] Siddharth Tourani, Alexander Shekhovtsov, Carsten Rother, and Bogdan Savchynskyy. MPLP++: Fast, parallel dual block-coordinate ascent for dense graphical models. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 251–267, 2018. 1, 2, 3, 7

[59] Siddharth Tourani, Alexander Shekhovtsov, Carsten Rother, and Bogdan Savchynskyy. Taxonomy of dual block-coordinate ascent methods for discrete energy minimization. In *AISTATS*, 2020. 2, 3

[60] Vibhav Vineet and PJ Narayanan. CUDA cuts: Fast graph cuts on the GPU. In *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 1–8. IEEE, 2008. 2

[61] Huayan Wang and Daphne Koller. Subproblem-tree calibration: A unified approach to max-product message passing. In *ICML (2)*, pages 190–198, 2013. 2, 3

[62] Tomas Werner. A linear programming approach to max-sum problem: A review. *IEEE transactions on pattern analysis and machine intelligence*, 29(7):1165–1179, 2007. 2, 3, 5

[63] Tomáš Werner, Daniel Průša, and Tomáš Dlask. Relative interior rule in block-coordinate descent. In *Proceedings of the IEEE International Conference on Computer Vision*, 2020. To appear. 3, 4, 5

[64] Jiadong Wu, Zhengyu He, and Bo Hong. Chapter 5 - efficient CUDA algorithms for the maximum network flow problem. In Wen mei W. Hwu, editor, *GPU Computing Gems Jade Edition*, Applications of GPU Computing Series, pages 55–66. Morgan Kaufmann, Boston, 2012. 2

[65] Zhiwei Xu, Thalaiyasingam Ajanthan, and Richard Hartley. Fast and differentiable message passing on pairwise markov random fields. In *Proceedings of the Asian Conference on Computer Vision*, 2020. 1, 2

[66] Zhen Zhang, Qinfeng Shi, Julian McAuley, Wei Wei, Yanning Zhang, and Anton Van Den Hengel. Pairwise matching through max-weight bipartite belief propagation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1202–1210, 2016. 2, 3