A Comparative Study of Graph Matching Algorithms in Computer Vision

Stefan Haller[*] Lorenz Feineis[*] Lisa Hutschenreiter[*] Florian Bernard[†]
Carsten Rother[*] Dagmar Kainmüller[‡] Paul Swoboda[§] Bogdan Savchynskyy[*]

[*]Heidelberg University, [†]University of Bonn, [‡]MDC Berlin, [§]MPI Informatik Saarbrücken

arXiv:2207.00291v2 [cs.CV] 29 Jul 2022

Abstract

The graph matching optimization problem is an essential component for many tasks in computer vision, such as bringing two deformable objects in correspondence. Naturally, a wide range of applicable algorithms have been proposed in the last decades. Since a common standard benchmark has not been developed, their performance claims are often hard to verify as evaluation on differing problem instances and criteria make the results incomparable. To address these shortcomings, we present a comparative study of graph matching algorithms. We create a uniform benchmark where we collect and categorize a large set of existing and publicly available computer vision graph matching problems in a common format. At the same time we collect and categorize the most popular open-source implementations of graph matching algorithms. Their performance is evaluated in a way that is in line with the best practices for comparing optimization algorithms. The study is designed to be reproducible and extensible to serve as a valuable resource in the future.

Our study provides three notable insights: **(i)** popular problem instances are exactly solvable in substantially less than 1 second, and, therefore, are insufficient for future empirical evaluations; **(ii)** the most popular baseline methods are highly inferior to the best available methods; **(iii)** despite the NP-hardness of the problem, instances coming from vision applications are often solvable in a few seconds even for graphs with more than 500 vertices.

1 Introduction

Finding correspondences between elements of two discrete sets, such as keypoints in images or vertices of 3D meshes, is a fundamental problem in computer vision and as such highly relevant for numerous vision tasks, including 3D reconstruction [49], tracking [64], shape model learning [29], and image alignment [7], among others. Graph matching [24,55,63] is a standard way to address such problems. In graph matching, vertices of the matched graphs correspond to the elements of the discrete sets to be matched. Graph edges define the cost structure of the problem: pairs of matched vertices are penalized in addition to the vertex-to-vertex matchings. This allows to take, e.g., the underlying geometrical relationship between vertices into account, but also makes the optimization problem NP-hard.

Deep graph matching [51] is a modern learning-based approach, that combines neural networks for computing matching costs with combinatorial graph matching algorithms to find a matching. The graph matching algorithm plays a crucial role in this context, as it has to provide high-quality solutions within a limited time budget. The high demand on run-time is also due to back-propagation learning and graph matching minimization being interleaved and executed together many times during training.

Hence, our work focuses on the *optimization* part of the graph matching pipeline. The modeling and learning aspects are beyond its scope. We evaluate a range of existing *open-source* algorithms. Our study compares their performance on a diverse set of computer vision problems. The focus of the evaluation lies on both, speed and objective value of the solution.

Why do we require a benchmark? Dozens of algorithms addressing the graph matching problem have been proposed in the computer vision literature, see, e.g., the surveys [24,55,63], and references therein. Most works promise state-of-the-art performance, which is persuasively demonstrated by experimental evaluation. However, **(i)** results from one article are often incomparable to results from another, since different problem instances with different costs are used, even if these instances are based on the same image data; **(ii)** not every existing method is evaluated on all available problem instances, even if open-source code is available. Some methods, especially those with poor performance on many instances, are very popular as baselines, whereas better performing techniques are hardly considered in comparisons; **(iii)** new algorithms are often only evaluated on easy, small-scale problems. This does not pro-

vide any information on how these algorithms perform on larger, more difficult problem instances. For these reasons, the field of graph matching has, in our view, not developed as well as it could have done. By providing a reproducible and extensible benchmark we hope to change this in the future. Such a benchmark is of particular importance for the fast-moving field of *deep graph matching*, as it helps to select an appropriate, fast solver for the combinatorial part of the learning pipeline.

Graph matching problem. Let $\mathcal{V}$ and $\mathcal{L}$ be the two finite sets, whose elements we want to match to each other. For each pair $i, j \in \mathcal{V}$ and each pair $s, l \in \mathcal{L}$ a *cost* $c_{is,jl} \in \mathbb{R}$ is given. Each pair can be interpreted as an *edge* between a pair of *vertices* of an underlying graph. This is where the term *graph matching* comes from. Note that direct *vertex-i-to-vertex-s* matching costs are defined by the *diagonal* elements $c_{is,is}$ of the resulting *cost* (or *affinity*) *matrix* $C = (c_{is,jl})$ with $is = (i,s) \in \mathcal{V} \times \mathcal{L}$, $jl = (j,l) \in \mathcal{V} \times \mathcal{L}$. The diagonal elements are referred to as *unary costs*, in contrast to the *pairwise costs* defined by non-diagonal entries. The goal of graph matching is to find a *matching*, or mutual *assignment*, between elements of the sets $\mathcal{V}$ and $\mathcal{L}$ that minimizes the total cost for all pairs of assignments. It can be formulated as the following integer quadratic problem[1]:

$$\begin{aligned} &\min_{x \in \{0,1\}^{\mathcal{V}\times\mathcal{L}}} \sum_{i,j\in\mathcal{V}} \sum_{s,l\in\mathcal{L}} c_{is,jl}\, x_{is}\, x_{jl} \\ &\text{s.t.} \begin{cases} \forall i \in \mathcal{V}\colon \sum_{s\in\mathcal{L}} x_{is} \le 1\,, \text{ and} \\ \forall s \in \mathcal{L}\colon \sum_{i\in\mathcal{V}} x_{is} \le 1\,. \end{cases} \end{aligned} \tag{1}$$

The vector x defines the matching as $x_{is} = 1$ corresponds to assigning i to s. The inequalities in (1) allow for this assignment to be *incomplete*, i.e., some elements of both sets may remain unassigned. This is in contrast to *complete* assignments, where each element of $\mathcal{V}$ is matched to *exactly one* element of $\mathcal{L}$ and vice versa. Note that complete assignments require $|\mathcal{V}| = |\mathcal{L}|$.

Relation to the quadratic assignment problem. The graph matching problem (1) is closely related to the NP-hard [50] *quadratic assignment problem (QAP)* [12], which is well-studied in operations research [12,15,48]. The QAP only considers *complete* assignments, i.e., $|\mathcal{V}| = |\mathcal{L}|$ and equality is required in the constraints in (1). In contrast, in the field of computer vision incomplete assignments are often required in the model to allow for, e.g., outliers or matching of images with different numbers of features. Still, graph matching and the QAP are polynomially reducible to each other, see supplement for a proof.

The most famous QAP benchmark is the QAPLIB [11] containing 136 problem instances. However, the benchmark problems in computer vision (CV) substantially differ from those in QAPLIB both by the feasible set that includes incomplete assignments, as well as by the structure of the cost matrix C: **(i)** CV problems are usually of a general, more expressive *Lawler* form [42], whereas QAPLIB considers factorizable costs $c_{is,jl} = f_{ij} d_{sl}$ known as *Koopmans-Beckmann* form. The latter allows for more efficient specialized algorithms. **(ii)** the cost matrix C in QAPLIB is often dense, whereas in CV problems it is typically sparse, i.e., a large number of entries in C are 0; **(iii)** for CV problem instances the cost matrix C may contain *infinite* costs on the diagonal to prohibit certain vertex-to-vertex mappings; **(iv)** QAPLIB problems are different from an optimization point of view. For instance, while the classical LP relaxation [1] is often quite loose for QAPLIB problem instances, it is tight or nearly tight for typical instances considered in CV.

Consequently, comparison results on QAPLIB and CV instances differ significantly. It is also typical for NP-hard problems that instances coming from different applied areas require different optimization techniques. Therefore, a dedicated benchmarking on the CV datasets is required.

Contributions. Our contribution is three-fold: **(i)** Based on open source data, we collected, categorized and generated 451 *existing* graph matching instances grouped into 11 datasets in a common format. Most graph matching papers use only a small subset of these datasets for evaluation. Our format provides a *ready-to-use* cost matrix C and does not require any image analysis to extract the costs. **(ii)** We collected and categorized 20 open-source graph matching algorithms and evaluated them on the above datasets. During that we adapted the cost matrix to requirements of particular algorithms where needed. For each method we provide a brief technical description. We did not consider algorithms with no publicly available open source code. **(iii)** To allow our benchmark to grow further, we set up a web site[2] with all results. Our benchmark is reproducible, extensible and follows the best practices of [6]. We will maintain its web-page in the future and welcome scientists to add problem instances as well as algorithms.

Our work significantly excels evaluations in *all* the papers introducing the algorithms we study. This implies also to the largest existing comparison [31] so far. The latter considers only 8 out of the 11 datasets and evaluates 6 algorithms out of our 20.

2 Background to algorithms

In this section we briefly review the main theoretical concepts and building blocks of the considered approaches.

[1] For sets A and B the notation $x \in A^B$ denotes a vector x whose coordinates take on values from the set A and are indexed by elements of B, i.e., each element of B corresponds to a value from A.

[2] The web site for the benchmark is available at https://vislearn.github.io/gmbench/.

Linearization. In case all pairwise costs are zero, the objective in (1) linearizes to $\sum_{i\in\mathcal{V},s\in\mathcal{L}} c_{is,is}x_{is}$, turning (1) into the *incomplete linear assignment problem (iLAP)*. A typical way of how the iLAP is obtained in existing algorithms is by considering the Taylor expansion of the objective (1) in the vicinity of a given point x. The linear term of this expansion forms the iLAP objective. The iLAP can be reduced to a complete linear assignment problem (LAP) [10], see supplement, and addressed by, e.g., Hungarian [41] or auction [8] algorithms. Below, when we refer to LAP this includes both LAP and iLAP problems.

Birkhoff polytope and permutation matrices. For $|\mathcal{V}| = |\mathcal{L}|$ and $x \in [0,1]^{\mathcal{V}\times\mathcal{L}}$, where $[0,1]$ denotes the closed interval from 0 to 1, the constraints $\sum_{s\in\mathcal{L}} x_{is}$=1, $\forall i \in \mathcal{V}$, and $\sum_{i\in\mathcal{V}} x_{is}$=1, $\forall s \in \mathcal{L}$, define the set of *doubly-stochastic matrices* also known as *Birkhoff polytope*. Its restriction to binary vectors $x \in \{0,1\}^{\mathcal{V}\times\mathcal{L}}$ is called the *set of permutations* or *permutation matrices*.

Inequality to equality transformation. By adding *slack* or *dummy* variables, indexed by #, with zero cost in the objective in (1), the uniqueness constraints in (1) can be rewritten as equalities for $\mathcal{V}^{\#} := \mathcal{V}\dot\cup\{\#\}$, $\mathcal{L}^{\#} := \mathcal{L}\dot\cup\{\#\}$, and $x \in [0,1]^{\mathcal{V}^{\#}\times\mathcal{L}^{\#}}$, where $\dot\cup$ is the disjoint union:

$$\mathcal{B} := \left\{ x \;\middle|\; \begin{array}{ll} \forall i \in \mathcal{V}\colon & \sum_{s\in\mathcal{L}^{\#}} x_{is} = 1\,, \quad \text{and} \\ \forall s \in \mathcal{L}\colon & \sum_{i\in\mathcal{V}^{\#}} x_{is} = 1 \end{array} \right\}. \tag{2}$$

Here, $x_{i\#} = 1$ (or $x_{\#s} = 1$) means that the node i (or label s) is unassigned. Following [65], we refer to the elements of $\mathcal{B}$ as *doubly-semi-stochastic matrices*.

Doubly-stochastic relaxation. Replacing the *integrality constraints* $x \in \{0,1\}^{\mathcal{V}\times\mathcal{L}}$ in (1) with the respective *box constraints* $x \in [0,1]^{\mathcal{V}\times\mathcal{L}}$ leads to a *doubly-stochastic relaxation* of the graph matching problem.[3] Despite the convexity of its feasible set, the doubly-stochastic relaxation is NP-hard because of the non-convexity of its quadratic objective in general [53].

Probabilistic interpretation. Doubly-stochastic relaxations are often motivated from a probabilistic perspective, where the individual matrix entries represent matching probabilities. An alternative probabilistic interpretation is to consider the *product graph* between $\mathcal{V}$ and $\mathcal{L}$, in which the edge weights directly depend on the cost matrix C. This way, graph matching can be understood as selecting reliable nodes in the product graph, e.g., by random walks [17].

Injective and bijective formulations. Assume $|\mathcal{V}| \le |\mathcal{L}|$. A number of existing approaches consider an asymmetric formulation of the graph matching problem (1), where the uppermost constraint in (1) is exchanged for equality, i.e., $\forall i \in \mathcal{V}\colon \sum_{s\in\mathcal{L}} x_{is} = 1$. We call this formulation *injective*. The strict inequality case $|\mathcal{V}| < |\mathcal{L}|$ is also referred to as an *unbalanced QAP* in the literature. Note that to address problems of the general form (1) by such algorithms, it is necessary to extend the set $\mathcal{L}$ with $|\mathcal{V}|$ dummy elements. This is similar to the reduction from graph matching to QAP described in the supplement. Since availiable implementations of multiple considered algorithms are additionally restricted to the case $|\mathcal{V}| = |\mathcal{L}|$, i.e. to the classical QAP as introduced in Section 1, we adopt the term *bijective* to describe the corresponding algorithms and datasets.

Spectral relaxation. The graph matching objective in (1) can be compactly written as $x^\top Cx$. Instead of the uniqueness constraints in (1) the *spectral relaxation* considers the non-convex constraint $x^\top x = n$. This constraint includes all matchings with exactly n assignments, which is of interest when the total number of assignments n is known, e.g., for the injective formulation where $n = |\mathcal{V}|$. The minimization of $x^\top Cx$ subject to $x^\top x = n$ reduces to an *eigenvector problem*, i.e., finding a vector x corresponding to the smallest eigenvalue of the matrix C. The latter amounts to minimizing a Rayleigh quotient [30].

Path following. Another way to deal with the non-convexity of the graph matching problem is *path-following*, represented by [69] in our study. The idea is to solve a sequence of optimization problems with objective $f_{\alpha^t}(x) = (1-\alpha^t)f_{\text{cvx}}(x) + \alpha^t f_{cav}(x)$ for α^t, $t \in 1,\dots,N$, gradually growing from 0 to 1. The (approximate) solution of each problem in the sequence is used as a starting point for the next. The hope is that this iterative process, referred to as *following the convex-to-concave path*, leads to a solution with low objective value for the whole problem. The objective for $\alpha^1 = 0$ is equal to $f_{\text{cvx}}(x)$ and is convex, therefore it can be solved to global optimality. The objective for $\alpha^N = 1$ is equal to $f_{\text{cav}}(x)$ and is concave. Its local optima over the set of doubly-stochastic matrices are guaranteed to be binary, and, therefore, *feasible assignments*, i.e., they satisfy all constraints of (1).

Graphical model representation. The graph matching problem can be represented in the form of a *maximum a posteriori (MAP) inference* problem for discrete graphical models [54], known also as *Markov random field (MRF) energy minimization* and closely related to *valued and weighted constraint satisfaction* problems. As several graph matching works in computer vision [31, 57, 67], use this representation, we provide it below in more detail.

Let $(\mathcal{V},\mathcal{E})$ be an undirected graph, with the finite set $\mathcal{V}$ introduced above being the *set of nodes* and $\mathcal{E} \subseteq \binom{\mathcal{V}}{2}$ being the set of *edges*. For convenience we denote edges $\{i,j\} \in \mathcal{E}$ simply by ij. Let the finite set $\mathcal{L}$ introduced above be the set of *labels*. We associate with each node $i \in \mathcal{V}$ a set $\mathcal{L}_i^{\#} = \mathcal{L}_i \dot\cup \{\#\}$ with $\mathcal{L}_i \subseteq \mathcal{L}$. Like above, # stands for the *dummy label* distinct from all labels in $\mathcal{L}$ to encode that *no label* is selected. With each label $s \in \mathcal{L}_i^{\#}$ in each node $i \in \mathcal{V}$ the *unary cost* $\theta_{is} := c_{is,is}$ (0 for $s = \#$) is associated. The case $|\mathcal{L}_i| < |\mathcal{L}|$ corresponds to infinite unary costs $c_{is,is} = \infty$, $s \in \mathcal{L}\backslash\mathcal{L}_i$, as the respective

[3] Strictly speaking, the term *doubly-stochastic* corresponds to the case when equality constraints are considered in (1). In [65] the inequality variant is called *doubly semi-stochastic* but we use *doubly-stochastic* in both cases.

assignments can be excluded from the very beginning. Likewise, with each edge $ij \in \mathcal{E}$ and each label pair $sl \in \mathcal{L}_i^\# \times \mathcal{L}_j^\#$, the pairwise cost $\theta_{is,jl} = c_{is,jl} + c_{jl,is}$ (0 for s or $l = \#$) is associated. The graph $(\mathcal{V}, \mathcal{E})$ being undirected implies $ij = ji$ and $\theta_{is,jl} = \theta_{jl,is}$. An edge ij belongs to $\mathcal{E}$ only if there is a label pair $sl \in \mathcal{L}_i \times \mathcal{L}_j$ such that $\theta_{is,jl} \neq 0$. In this way a sparse cost matrix C may translate into a sparse graph $(\mathcal{V}, \mathcal{E})$.

The problem of finding an optimal assignment of labels to nodes, equivalent to the *graph matching* problem (1), can thus be stated as

$$\begin{aligned} &\min_{y \in \mathcal{Y}} \left[E(y) := \sum_{i \in \mathcal{V}} \theta_{iy_i} + \sum_{ij \in \mathcal{E}} \theta_{iy_i, jy_j} \right] \\ &\text{s.t.} \quad \forall\, i, j \in \mathcal{V}, i \neq j \colon y_i \neq y_j \text{ or } y_i = \# \end{aligned} \tag{3}$$

where $\mathcal{Y}$ stands for the Cartesian product $\bigtimes_{i \in \mathcal{V}} \mathcal{L}_i^\#$, and $y_i = s$, $s \in \mathcal{L}_i$, is equivalent to $x_{is} = 1$ in terms of (1). Essentially, (3) corresponds to a *MAP inference problem for discrete graphical models* [54] with additional uniqueness constraints for the labels.

ILP representation and LP relaxations. Based on (3) the graph matching problem can be expressed by a linear objective subject to linear and integrality constraints by introducing variables $x_{is,jl} = x_{jl,is}$ for each pair of labels $sl \in \mathcal{L}_i^\# \times \mathcal{L}_j^\#$ in neighboring nodes $ij \in \mathcal{E}$, and enforcing the equality $x_{is,jl} = x_{is} x_{jl}$ with suitable linear constraints. An *integer linear program (ILP)* formulation of the graph matching problem (1) can then be written as:

$$\min_{x \in \{0,1\}^{\mathcal{J}}} \sum_{\substack{i \in \mathcal{V} \\ s \in \mathcal{L}_i^\#}} c_{is} x_{is} + \sum_{\substack{ij \in \mathcal{E} \\ sl \in \mathcal{L}_i^\# \times \mathcal{L}_j^\#}} (c_{is,jl} + c_{jl,is})\, x_{is,jl} \tag{4}$$

$$\forall i \in \mathcal{V} \colon \sum_{s \in \mathcal{L}_i^\#} x_{is} = 1\,, \quad \forall s \in \mathcal{L} \colon \sum_{i \in \mathcal{V}} x_{is} \leq 1\,, \tag{5}$$

$$\forall ij \in \mathcal{E},\ l \in \mathcal{L}_j^\# \colon \sum_{s \in \mathcal{L}_i^\#} x_{is,jl} = x_{jl}\,. \tag{6}$$

Here $\mathcal{J} = \{(i,s) \colon i \in \mathcal{V},\ s \in \mathcal{L}_i^\#\} \cup \{(is,jl) \colon ij \in \mathcal{E}, sl \in \mathcal{L}_i^\# \times \mathcal{L}_j^\#\}$ denotes the set of coordinates of the vector x. The formulation (4)-(6) differs from the standard ILP representation for discrete graphical models by the label uniqueness constraints (5, rightmost). Substitution of the integrality constraints $x \in \{0,1\}^{\mathcal{J}}$ in (4) with the box constraints $x \in [0,1]^{\mathcal{J}}$ results in the respective *LP relaxation*.

3 Graph matching algorithms

Below we summarize the graph matching methods that we consider in our comparison, see Table 1 for an overview of their characteristics and references.

3.1 Primal heuristics

Linearization based. These methods are based on iterative linearizations of the quadratic objective (1) derived from its Taylor expansion.

Iterated projected fixed point (`ipfp`) [46] solves on each iteration the LAP obtained through linearization in the vicinity of a current, in general non-integer, assignment. Between iterations the quadratic objective is optimized along the direction to the obtained LAP solution, which yields a new, in general non-integer assignment. We evaluate two versions of `ipfp` which differ by their initialization: `ipfpu` is initialized with $x^0 \in [0,1]^{\mathcal{V} \times \mathcal{L}}$, where $x^0_{is} = 1/\sqrt{N}$ if $c_{is,is} < \infty$, and $x^0_{is} = 0$ otherwise. Here, $N := |\{is \in \mathcal{V} \times \mathcal{L} \mid c_{is,is} < \infty\}|$. `ipfps` starts from the result of the spectral matching `sm` [43] described below.

Graduated assignment (`ga`) [27] optimizes the doubly-stochastic relaxation. On each iteration it approximately solves the LAP obtained through linearization in the vicinity of a current, in general non-integer, assignment utilizing the Sinkhorn algorithm [40] for a given fixed temperature. The obtained approximate solution is used afterwards as the new assignment. The temperature is decreased over iterations to gradually make the solutions closer to integral.

Fast approximate quadratic programming (`fw`) [62] considers the Frank-Wolfe method [25] for optimizing over the set $\mathcal{B}$, c.f. (2). Each iteration first solves a LAP to find the optimum of the linearization at the current solution, followed by a line search in order to find the best convex combination of the current and the new solution. To obtain an integer solution, the objective of the LAP solution is evaluated in each iteration, and the lowest one among all solutions is kept. The initial LAP is based on the unary costs only. The implementation [56] we evaluate is applicable to the general Lawler form of the problem (1), in contrast to the Koopmans-Beckmann form addressed in [62].

Norm constraints based. *Spectral matching* (`sm`) [43] uses a spectral relaxation that amounts to a Rayleigh quotient problem [30] which can be optimized by the power iteration method. Here, each update comprises of a simple matrix multiplication and a subsequent normalization, so that x^t is iteratively updated via $x^{t+1} = -Cx^t / \|Cx^t\|_2$.

Spectral matching with affine constraints (`smac`) [21] is similar to `sm`, but additionally takes into account affine equality constraints that enforce one-to-one matchings. The resulting formulation amounts to a Rayleigh quotient problem under affine constraints, that can efficiently be computed in terms of the eigenvalue decomposition.

Max-pooling matching (`mpm`) [18] resembles `sm`, but it replaces the sum-pooling implemented in terms of the matrix multiplication $-Cx$ in the power iteration update of `SM` by a max-pooling operation. With that, only

method	IQP	ILP	bijective	non-pos.	0-unary	linearizing	norm	doubly	spectral	discret.	path fol.	fusion	duality	SGA	BCA	Matlab	C++
`fgmd` [69]	+		+								+					[68]	
`fm` [31]		+										+					[32]
`fw` [62]	+					+		+									[56]
`ga` [27]	+		+			+		+		+						[20]	
`ipfps` [46]	+		+	+		+		+		+						[44]	
`ipfpu` [46]	+		+			+		+		+						[44]	
`lsm` [33]	+		+	+			+			+						[66]	
`mpm` [18]	+		+	+			+			+						[19]	
`pm` [65]	+		+	+	+			+		+						[68]	
`rrwm` [17]	+		+			+		+		+						[16]	
`smac` [21]	+		+	+			+		+	+						[20]	
`sm` [43]	+		+	+			+		+	+						[44]	
`dd-ls(0/3/4)` [59]		+											+	+			[38]
`fm-bca` [31]		+										+	+		+		[32]
`hbp` [67]		+	+										+		+	[66]	
`mp(-mcf/-fw)` [57]		+											+		+		[56]

Meaning of properties ('+' indicates presence): *IQP*: addresses IQP formulation; *ILP*: addresses ILP formulation; *bijective*: addresses bijective formulation; *non-pos.*: requires non-positive costs, see Remark 1; *0-unary*: requires zero unary costs; *linearizing*: linearization-based method; *norm*: imposes norm-constraints; *doubly*: addresses doubly-stochastic relaxation; *spectral*: solves spectral relaxation; *discret.*: discretization as in Remark 2; *path fol.*: path following method; *fusion*: utilizes fusion; *duality*: Lagrange duality-based; *SGA*: uses dual sub-gradient ascent; *BCA*: uses dual block-coordinate ascent; *Matlab*: implemented in Matlab [reference to code]; *C++*: implemented in C++ [reference to code].

Table 1: Method properties. Purely primal heuristics are separated from the dual methods by a horizontal line.

candidate matches with the smallest costs are taken into account.

Local sparse model (`lsm`) [33] solves the relaxation $\max_x x^\top Cx$, s.t. $||x||_{1,2}^2 = \sum_{i=1}^{|\mathcal{V}|}\left(\sum_{k=1}^{|\mathcal{L}|}|x_{ik}|\right)^2 = 1$, $x \geq 0$. The $l_{1,2}$-norm $||x||_{1,2}$ should encourage the solution of the above relaxation to be sparse in each row when treating x as a matrix. This resembles the sparsity property of permutation matrices, which satisfy $||x||_{1,2} = |\mathcal{V}|$.

Remark 1. *All of the norm constraints based algorithms described above require non-positive*[4] *costs in order to guarantee convergence of the underlying iterative techniques. This condition can be w.l.o.g. assumed for any graph matching problem. The corresponding cost transformation is described in the supplement.*

Probabilistic interpretation based. *Reweighted Random Walks Matching* (`rrwm`) [17] interprets graph matching as the problem of selecting reliable nodes in an *association graph*, whose weighted adjacency matrix is given by $-C$. Nodes are selected through a random walk that starts from one node and randomly visits nodes according to a Markov transition matrix derived from the edge weights of the association graph. In order to take into account matching constraints, the authors of [17] consider a reweighted random walk strategy.

Probabilistic matching (`pm`) [65] considers a probabilistic formulation of graph matching in which the quadratic objective is replaced by a relative entropy objective. It is shown that by doing so one can obtain a convex problem formulation via marginalization, which is optimized in terms of an iterative successive projection algorithm.

Remark 2. *Most of the primal heuristics considered above aim to optimize the quadratic objective* (1) *over a continuous set such as, e.g., the Birkhoff polytope. The resulting assignment $x \in \mathbb{R}^{\mathcal{V}\times\mathcal{L}}$ is, therefore, not guaranteed to be integer. As suggested in [17], to obtain an integer assignment we solve a LAP with $(-x)$ treated as the cost matrix. We apply this procedure as a postprocessing step for* **`ipfp`**, **`ga`**, **`sm`**, **`smac`**, **`mpm`**, **`lsm`**, **`rrwm`**, *and* **`pm`**. *Note that this postprocessing does not change an integer assignment.*

Path following based. *Factorized graph matching* (`fgmd`) [69] proposes an efficient factorization of the cost matrix to speed-up computations, and is based on the convex-concave path following strategy, see Section 2. Individual problems from the path are solved with the Frank-Wolfe method [25].

Randomized generation and fusion based. *Fusion moves with a greedy heuristic* (`fm`) [31] is based on the graphical model representation and consists of two parts: A randomized greedy assignment generation, and *fusion* of the assignments. The randomized generator greedily fixes labels in the nodes in a way that minimizes the objective value restricted to the already fixed labels. The fusion procedure merges the current assignment with the next generated one by approximately solving an auxiliary *binary* MAP inference problem utilizing QPBO-I [52]. The merged solution is guaranteed to be at least as good as the two input assignments. This property guarantees monotonic improvement of the objective value.

3.2 Lagrange duality-based techniques

The methods below consider the Lagrange decompositions [28] of the graph matching problem (1) [59], or its graphical model representation (3) [31, 57, 67], and optimize the corresponding dual. The methods differ in the dual optimization and chosen primal solution reconstruction algorithms.

[4] Non-negative in original maximization formulations

	(i) graphical model	(ii) LAP	(iii) primal
`hbp`	MPLP [26]	Hungarian [41]	branch & bound
`mp(-mcf)`	anisotropic diffusion [54]	network simplex [2]	LAP
`mp-fw`			`fw`
`fm-bca`	MPLP++ [60]	BCA	`fm`

Algorithms used for optimizing the *graphical model* and *LAP* part, as well as technique used to obtain a *primal* solution. For *LAP* in the *primal* column the solution of the LAP subproblem is reused as feasible assignment. Instead of solving the LAP subproblem, `fm-bca` performs a series of *BCA* steps wrt. the LAP dual variables.

Table 2: Characterization of dual BCA algorithms.

Block-coordinate methods (`hbp`, `mp-*`, `fm-bca`). The works [31, 57, 67] employ a block-coordinate ascent (BCA) technique to optimize the dual problem obtained by relaxing the coupling (6) and label uniqueness constraints (5, rightmost). Since the dual is piece-wise linear, BCA algorithms may not attain the dual optimum, but may get stuck in a sub-optimal fixed point [9, 54].

Although the elementary operations performed by these algorithms are very similar, their convergence speed and attained fixed points differ drastically. In a nutshell, these methods decompose the problem (3) into the graphical model without uniqueness constraints, and the LAP problem, as described, e.g., in [31]. Dual algorithms reparametrize the problem making it more amenable to primal techniques [54]. Table 2 gives an overview of the evaluated combinations for **(i)** optimizing the dual of the graphical model, **(ii)** optimizing the LAP, and **(iii)** obtaining the primal solution from the reparametrized costs[5], which influence the practical performance of BCA solvers. Additionally, `mp-mcf` and `mp-fw` tighten the relaxation by considering triples of graph nodes as subproblems.

Subgradient method (`dd-ls*`). The algorithms denoted as `dd-ls*` with `*` being `0`, `3` or `4` represent different variants of a dual subgradient optimization method [59]. The variant `dd-ls0` addresses the relaxation equivalent to a symmetrized graphical model formulation, see supplement for a description. This is achieved by considering the Lagrange decomposition of the problem into two graphical models, with $\mathcal{V}$ and $\mathcal{L}$ being the set of nodes, respectively, and a LAP subproblem. The graphical models are further decomposed into acyclic ones, i.e. trees, solvable by dynamic programming, see, e.g., [54, Ch.9]. The *tree decomposition* is not described in [59], and we reconstructed it based on the source code [38] and communication with the authors. As we observed it to be more efficient than the *max-flow subproblems* suggested in the paper [59] the latter were not used in our evaluation.

Variants `dd-ls3` and `dd-ls4` tighten the relaxation of `dd-ls0` by considering *local subproblems* of both graphical models in the decomposition. These are obtained by reducing the node sets $\mathcal{V}$ and $\mathcal{L}$ to 3 or respectively 4 elements inducing a connected subgraph of the graphical model, see [59] for details.

[5] *Reparametrized* costs are also known as *reduced* costs, e.g., in the simplex tableau.

4 Benchmark

Datasets. The 11 datasets we collected for evaluation of the graph matching algorithms stem from applications in computer vision and bio-imaging. All existing graph matching papers use only a subset of these datasets for evaluation purposes. Together these datasets contain 451 problem instances. Table 3 gives an overview of their characteristics. We modified costs in several datasets to make them amenable to some algorithms, see supplement. Our modification results in a constant shift of the objective value for each feasible assignment, and, therefore, does not influence the quality of the solution.

Below we give a brief description of each dataset. Along with the standard computer vision datasets with small-sized problems, `hotel`, `house-dense/sparse`, `car`, `motor` and `opengm` with $|\mathcal{V}|$ up to 52, our collection contains the middle-sized problems `flow`, with $|\mathcal{V}|$ up to 126, and the large-scale `worms` and `pairs` problems with $|\mathcal{V}|$ up to 565.

Wide baseline matching (`hotel`, `house-dense/sparse`) is based on a series of images of the same synthetic object with manually selected landmarks from different viewing angles based on the work by [14]. For `hotel` and `house-dense` we use the same models as in [57] published in [58]. `house-sparse` consists of the same image pairs as `house-dense`, but the cost structure is derived following the approach of [67] that results in significantly sparser problem instances. Graphs with the landmarks as nodes are obtained by Delaunay triangulation. The costs are set to $c_{is,jl} = -\exp(-(d_{ij} - d_{sl})^2/2500)A^1_{ij}A^2_{sl}$ where d_{ij}, d_{sl} are Euclidean distances between two landmarks and $A^1 \in \{0,1\}^{\mathcal{V}\times\mathcal{V}}$, $A^2 \in \{0,1\}^{\mathcal{L}\times\mathcal{L}}$ are adjacency matrices of the corresponding graphs. The unary costs are zero.

Keypoint matching (`car`, `motor`) contains *car* and *motor*bike images from the PASCAL VOC 2007 Challenge [23] with the features and costs from [47]. We use the instances available from [32].

Large displacement flow (`flow`) was introduced by [3] for key point matching on scenes with large motion. We use the instances from [32] which use keypoints and costs as in [57].

OpenGM matching (`opengm`) is a set of non-rigid point matching problems by [39], now part of the *OpenGM* Benchmark [36]. We use the instances from [32].

dataset	#inst.	#opt.	bijective	injective	non-pos.	0-unary	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{L}\rvert/\lvert\mathcal{V}\rvert$	density (%)	diagonal dens. (%)	data
`caltech-large` [17]	9	1		+	+	+	36-219	0.4-2.3	0.55	18.1	[16]
`caltech-small` [17]	21	12		+	+	+	9-117	0.4-3	0.99	26.8	[16]
`car` [23,47]	30	30	+		+		19-49	1	2.9	100	[45]
`flow` [3,57]	6	6					48-126	≈ 1	0.39	15.8	[4,5]
`hotel` [14,59]	105	105	+				30	1	12.8	100	[13,58]
`house-dense` [14,59]	105	105	+				30	1	12.6	100	[13,58]
`house-sparse` [14,67]	105	105	+		+	+	30	1	1.5	100	[13]
`motor` [23,47]	20	20	+		+		15-52	1	3.8	100	[45]
`opengm` [36,39]	4	4	+				19-20	1	74.8	100	[37]
`pairs` [31,34]	16	0					511-565	≈ 1	0.0019	3.7	[32]
`worms` [34]	30	28					558	≈ 2.4	0.00038	1.6	[35]

Meaning of properties:
#inst.: number of problem instances; *#opt.*: number of known optima; *bijective/injective*: bi-/injective assignment is assumed; *non-pos.*: all costs are non-positive; *0-unary*: datasets with zero unary costs; $|\mathcal{V}|$: number of elements in $\mathcal{V}$; $|\mathcal{L}|/|\mathcal{V}|$: ratio of the number of elements in $\mathcal{L}$ to the number of elements in $\mathcal{V}$; *density (%)*: percentage of non-zero elements in the cost matrix C; *diag. dens. (%)*: percentage of non-infinite elements on the diagonal of C; *data*: [references] to problem instances, images, feature coordinates or ground truth.

Table 3: Dataset properties. A '+' indicates that all problem instances of the dataset have the respective property.

The `caltech` dataset was proposed in [17]. The data available at the project page [16] contains the *mutual projection error* matrix $D = (d_{is,jl})$, lists of possible assignments, and partial ground truth. We reconstructed the dataset from this data. Unary costs are set to zero. Pairwise costs for pairs of possible assignments are set to $c_{is,jl} = -\max(50 - d_{is,jl}, 0)$. We divided the dataset into `caltech-small` and `caltech-large`, where all instances with more than 40000 non-zero pairwise costs are considered as large.

Worm atlas matching (`worms`) has the goal to annotate nuclei of *C. elegans*, a famous model organism used in developmental biology, by assigning nuclei names from a known atlas of the organism. A detailed description can be found in [34]. We use the instances obtained from [32] which are originally from [35].

Worm-to-worm matching (`pairs`) directly matches the cell nuclei of individual *C. elegans* worms to each other. The resulting models are much coarser than those of the `worms` dataset. We consider the same 16 problem instances as [31] using the models from [32].

Evaluation metrics. For *fixed-time* performance evaluation [6] we restrict run-time (1, 10, 100 s) and evaluate attained objective values E, lower bound D and, for datasets with ground truth available, accuracy *acc*. We also report the number of optimally solved instances per dataset.

For *fixed-target* performance evaluation [6] we measure the time $t_s(p)$ until each solver s solves the problem p within an optimality tolerance of 0.1%. For instances with unknown optimum, we consider the best achieved objective value across all methods as optimum as suggested in [6]. The performance ratio to the best solver is computed by $r_s(p) = \frac{t_s(p)}{\min\{t_s(p):\forall s\}}$. We create a performance profile [6, 22] by computing $\rho_s(\tau) = \frac{1}{|P|} \cdot |\{r_s(p) \leq \tau : \forall p\}|$ for each solver s where $|P|$ denotes the total number of problem instances. Intuitively, $\rho_s(\tau)$ is the probability of solver s being at most τ times slower than the fastest solver.

5 Empirical Results

Fixed-time evaluation presented in Table 4 addresses small problem instances, whereas Table 5 addresses mid-size and large problem instances. The performance profile for fixed-target evaluation is presented in Figure 1. More detailed results are available in the supplement. Results have been obtained by taking the minimum run-time across five trials on an AMD EPYC 7702 2.0 GHz processor. Randomized alogrithms were made deterministic by fixing their random seed (`fm` and `fm-bca`). We equally treat Matlab and C++ implementations, in spite of the apparent efficiency considerations. The reason for this is that the solution quality of *all* Matlab algorithms in our evaluation is inferior to the C++ techniques, even if run-time is ignored.

For **small problems** we show results for 1 second in Table 4, as the best methods already solve almost all instances to optimality within this time. The best methods on these datasets are `fm`, `fm-bca` and `dd-ls0`. `dd-ls3/4` have higher costs per iteration, and require more than 1 second to arrive at the solution quality of `dd-ls0`. The other dual BCA-based methods perform almost as good on all but the `opengm` dataset, which seems to be the most difficult dataset amongs the one in Table 4. Apart from `fm` pure primal heuristics are unable to compete with duality-based techniques. The comparison of the results for `house-dense` and `house-sparse` shows that most of the primal heuristics perform much better on sparse problems.

For **larger problems** the most representative times shown in Table 5 are 1, 10 and 100 seconds, depending on the dataset. Again, the duality-based methods and the `fm` heuristic lead the table. The `fm-bca` method consistently attains the best or close to best objective and accuracy values on all datasets, whereas its lower bound is often worse than the lower bounds obtained by the `mp-*` and `dd-ls*` methods. In contrast, most of the primal heuristics as well as `hbp` fail, and, for brevity, are omitted in Table 5.

	hotel (1s)			house-dense (1s)			house-sparse (1s)			car (1s)			motor (1s)			opengm (1s)		caltech-small (1s)		
	opt	*E*	acc	opt	*E*	acc	opt	*E*	acc	opt	*E*	acc	opt	*E*	acc	opt	*E*	opt	*E*	acc
fgmd	0	—*		0	—*		0	—*		0	—*		0	—*		0	—*	0	—*	
fm	97	-4292	100	**100**	**-3778**	100	**100**	**-67**	100	77	**-69**	88	90	**-63**	93	**100**	**-171**	**52**	-8906	58
fw	97	-4288	99	**100**	**-3778**	100	0	0	0	7	-63	63	20	-58	70	0	-152	0	0	0
ga	0	947	15	0	3491	8	**100**	**-67**	100	57	-68	84	55	-62	90	50	-167	0	—*	
ipfps	0	1051	14	0	3654	8	**100**	**-67**	100	10	-65	80	25	-61	85	0	-95	19	**-8983**	67
ipfpu	0	1062	15	0	3659	8	**100**	**-67**	100	7	-60	69	15	-58	77	0	-86	10	-8829	62
lsm	0	—*		0	—*		46	-65	96	0	-51	52	10	-52	64	0	-67	0	—*	
mpm	43	-2585	78	0	1260	53	0	-60	90	7	—*		5	—*		0	-94	0	—*	
pm	0	775	33	0	3262	18	0	-54	83	0	-35	23	0	-35	32	0	-83	0	-6510	51
rrwm	0	744	15	0	2895	10	**100**	**-67**	100	37	-68	87	50	-62	89	0	-154	5	—*	
sm	0	1086	13	0	3789	9	96	**-67**	100	7	-63	76	40	-60	87	0	-101	0	-3932	36
smac	1	-1571	61	0	2817	31	37	-44	63	0	-52	52	10	-52	66	0	-84	0	-6196	42
dd-ls0	**100**	**-4293**	100	**100**	**-3778**	100	**100**	**-67**	100	**97**	**-69**	91	**100**	**-63**	97	50	-160	43	-7414	58
dd-ls3	**100**	**-4293**	100	**100**	**-3778**	100	96	**-66**	100	47	-57	74	65	-57	87	0	-118	33	-6842	57
dd-ls4	98	-4291	100	92	-3763	99	18	-56	86	3	-49	59	30	-52	78	0	-105	24	-6332	54
fm-bca	**100**	**-4293**	100	**100**	**-3778**	100	**100**	**-67**	100	93	**-69**	92	**100**	**-63**	97	75	-170	38	-8927	62
hbp	97	—*		98	—*		**100**	**-67**	100	77	—*		95	—*		0	—*	0	—*	
mp	93	-4280	99	99	-3777	100	**100**	**-67**	100	80	**-69**	92	90	**-63**	96	0	-57	14	-7967	59
mp-fw	99	-4292	100	**100**	**-3778**	100	**100**	**-67**	100	90	**-69**	91	95	**-63**	98	0	-150	33	-8886	60
mp-mcf	90	-4245	98	31	-3542	89	**100**	**-67**	100	87	**-69**	91	90	**-63**	98	0	-57	5	-7882	60

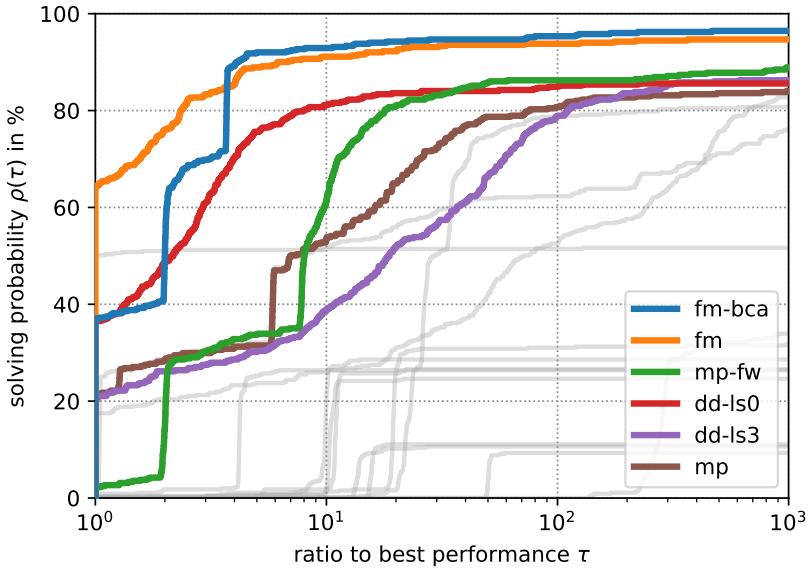
opt: optimally solved instances (%); **E:** average best objective value; **acc:** average accuracy corresponding to best objective (%)
—*: method yields no solution for at least one problem instance within the given time interval.

Table 4: Fixed-time evaluation of small problem instances. Maximal run-time per problem instance is 1 second. Boldface marks best values, except for accuracy since algorithm do not optimize it explicitly and do not have access to ground truth. Horizontal line separates purely primal from duality-based methods. Accuracy omitted for opengm as no ground truth available. Dual bounds omitted as most problems are solved optimally.

	flow (1s)			worms (1s)				caltech-large (10s)			caltech-large (100s)			pairs (10s)			pairs (100s)		
	opt	*E*	*D*	opt	*E*	*D*	acc	*E*	*D*	acc	*E*	*D*	acc	*E*	*D*	acc	*E*	*D*	acc
fm	**83**	**-2838**	-3436	**93**	-48457	-55757	89	-34117	-142829	52	-34125	-142829	52	-65625	-76418	54	-65825	-76418	55
fw	67	-2828	–	0	-46974	–	81	0	–	0	0	–	0	**-65797**	–	54	-65802	–	54
dd-ls0	33	-2345	-2968	0	60443	-163870	26	-32973	**-35007**	51	-33539	-34959	52	-61482	-73521	41	-62974	**-67306**	57
dd-ls3	17	-2059	-3030	0	64017	-160520	24	-28653	-42079	49	-33552	**-34914**	49	-61638	-73528	41	-62426	-67599	50
dd-ls4	0	-2062	-3090	0	65731	-160409	24	-25599	-62120	46	-30148	-38880	51	-61634	-74053	41	-61634	-70214	41
fm-bca	**83**	**-2838**	-2898	**93**	**-48460**	**-48514**	89	-34040	-48223	51	-34073	-48217	51	-65567	-70163	55	**-65913**	-69003	58
mp	33	-2628	**-2887**	0	—*			-32017	-46070	48	-32069	-46066	48	-64150	**-68255**	57	-64380	-68136	57
mp-fw	**83**	—*		0	—*			**-34237**	-48882	51	**-34277**	-45923	51	—*			—*		
mp-mcf	33	-2521	-2892	0	—*			-30362	-46630	47	-30737	-43833	47	-63990	-68318	56	-64174	-68053	57

opt, E, acc, —*: same as in Table 4; **D:** best attained lower bound if applicable, i.e., for dual methods, otherwise –

Table 5: Fixed-time evaluation of mid-size and large problem instances. Only the best performing algorithms are shown. Notation is the same as in Table 4. For each dataset the maximum allowed run-time per instance is given in parentheses. For flow no ground truth is available, so the column *acc* is omitted. For caltech-large and pairs no global optima are known, and the column *opt* is omitted.



Methods solving the largest number of instances, see $\rho(\tau = 10^3)$, are highlighted in color. Other methods are shown as "ghosts", i.e., unlabeled in gray. `fm` is the best solver in 65% of all cases, see $\rho(\tau = 1)$. `fm-bca` outperforms `fm` when the allowed performance ratio is increased to $\tau \geq 3.7$. Overall, `fm-bca` solves $\approx 97\%$ and `fm` solves $\approx 95\%$ of all instances. Following are duality-based methods like `mp-fw` and `dd-ls0`.

Figure 1: Run time performance profile [22] across all 451 instances.

Algorithms `dd-ls3/4` consider tighter relaxations than `dd-ls0`, but are slower, therefore lose in the competition on short time intervals. However, they have the ability to attain the best lower bounds given longer runs ($\gg$ 100s).

There is a significant performance gap between the closely related `hbp`, `mp-*` and `fm-bca` methods. Foremost, this is explained by the method for reconstructing the primal solution: The `fm` algorithm used in the `fm-bca` solver is solid also as a stand-alone technique, and significantly outperforms the `fw` and LAP heuristics used in the `mp-*` algorithms. The branch-and-bound solver used in `hbp` is quite slow and does not scale well. The second reason for different performance of these methods is the specific BCA algorithm used for the underlying discrete graphical model, c.f. Table 2. According to the recent study [61], which provides a unified treatment of the dual BCA methods for dense[6] graphical models, MPLP++ performs best, followed by anisotropic diffusion and MPLP as the slowest method. Table 5 shows that there is no solution suitable for every purpose: The speed of `fm-bca` comes at the price of a looser lower bound. Nonetheless, combining a primal heuristic with a dual optimizer consistently improves upon the results obtained by the heuristic alone. This holds for `fm`, but the effect is even more pronounced for the `fw` and LAP heuristics.

The fixed-target evaluation in Figure 1 confirms that the `fm` and `fm-bca` method are amongst the best performing solvers. While `fm-bca` uses `fm` as primal heuristics with additional dual BCA updates, the overhead of the latter is visible. After increasing the allowed performance ratio for `fm-bca` to a factor of 3.7, we can expect better solutions than `fm` alone. Other top performers are duality-based algorithms with `mp-fw` and `dd-ls0` being the closest followers.

[6] Most of the considered graphical models are dense in terms of [61].

6 Conclusions

Our evaluation shows that: **(i)** Most instances from the popular datasets `hotel`, `house`, `car` and `motor` can be solved to optimality in well below a second by several optimization techniques. `opengm` can also be solved to optimality in under a second, although it turns out to be hard for many methods. Therefore, we argue that *these datasets alone are not sufficient anymore to empirically show efficiency of new algorithms*. The most difficult in our collection are the datasets `caltech-*` and `pairs`. For a comprehensive evaluation of new methods more datasets are required. **(ii)** The most popular comparison baselines like `ipfp`, `ga`, `rrwm`, `pm`, `sm`, `smac`, `lsm`, `mpm` and `fgmd` are not competitive, and, therefore, *comparison to these alone should not anymore be considered as sufficient*. **(iii)** The most efficient methods are duality-based techniques equipped with efficient primal heuristics. In particular, the `fm`/`fm-bca` method currently attains the best or nearly best objective values for most problem instances in the shortest time. **(iv)** Although being NP-hard in general, the graph matching problem can be often efficiently solved in computer vision practice. For many of the considered datasets, including those with $|\mathcal{L}| > 1000$ and $|\mathcal{V}| > 500$, a reasonable approximate solution can be attained in less than a second.

7 Acknowlegments

This work was supported by the German Research Foundation (Unsupervised Model Discovery for Stereotypical Organisms, DFG SA 2640/2-1) and the Helmholtz Information & Data Science School for Health (HIDSS4Health). We thank the Center for Information Services and High Performance Computing (ZIH) at TU Dresden for its facilities for high throughput calculations.

References

[1] Warren P. Adams and Terri A. Johnson. Improved linear programming-based lower bounds for the quadratic assignment problem. *Discrete Mathematics and Theoretical Computer Science*, 1994.

[2] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms and Applications*. Prentice Hall, 1993.

[3] Hassan Abu Alhaija, Anita Sellent, Daniel Kondermann, and Carsten Rother. GraphFlow – 6D large displacement scene flow via graph matching. In *Proceedings of the DAGM German Conference on Pattern Recognition*, 2015.

[4] Hassan Abu Alhaija, Anita Sellent, Daniel Kondermann, and Carsten Rother. Graph matching problems for graphflow – 6D large displacement scene flow problem instances, 2018. https://research-explorer.app.ist.ac.at/record/5573.

[5] Hassan Abu Alhaija, Anita Sellent, Daniel Kondermann, and Carsten Rother. Project graphflow – 6D large displacement scene flow images, 2018. https://hci.iwr.uni-heidelberg.de/vislearn/research/image-matching/graphflow/.

[6] Vahid Beiranvand, Warren Hare, and Yves Lucet. Best practices for comparing optimization algorithms. *Optimization and Engineering*, 2017.

[7] Florian Bernard, Johan Thunberg, Peter Gemmar, Frank Hertel, Andreas Husch, and Jorge Goncalves. A solution for multi-alignment by transformation synchronisation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[8] Dimitri P. Bertsekas. A distributed algorithm for the assignment problem. *Lab. for Information and Decision Systems Working Paper, MIT*, 1979.

[9] Dimitri P Bertsekas. *Nonlinear programming, second edition*. Athena scientific, 1999.

[10] Rainer Burkard, Mauro Dell'Amico, and Silvano Martello. *Assignment Problems*. SIAM, 2009.

[11] Rainer Burkard, Stefan Karisch, and Franz Rendl. QAPLIB – a quadratic assignment problem library. *Journal of Global Optimization*, 1997.

[12] Rainer E. Burkard, Eranda Cela, Panos M. Pardalos, and Leonidas S. Pitsoulis. *The Quadratic Assignment Problem*, pages 1713–1809. Springer, 1998.

[13] Tiberio Caetano. Data for learning graph matching, 2011. https://www.tiberiocaetano.com/data/.

[14] Tibério S. Caetano, Julian J. McAuley, Li Cheng, Quoc V. Le, and Alexander J. Smola. Learning graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2009.

[15] Eranda Cela. *The Quadratic Assignment Problem: Theory and Algorithms*, volume 1. Springer Science & Business Media, 2013.

[16] Minsu Cho, Lee Jungmin, and Mu Lee Kyoung. Reweighted random walks for graph matching: Project page, 2010. https://cv.snu.ac.kr/research/~RRWM/.

[17] Minsu Cho, Jungmin Lee, and Kyoung Mu Lee. Reweighted random walks for graph matching. In *Proceedings of the European Conference on Computer Vision*, 2010.

[18] Minsu Cho, Jian Sun, Olivier Duchenne, and Jean Ponce. Finding Matches in a Haystack: A Max-Pooling Strategy for Graph Matching in the Presence of Outliers. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[19] Minsu Cho, Jian Sun, Olivier Duchenne, and Jean Ponce. Finding matches in a haystack source code, 2014. https://www.di.ens.fr/willow/research/maxpoolingmatching/.

[20] Timothee Cour. Graph matching toolbox in matlab, 2010. http://www.timotheecour.com/software/graph_matching/graph_matching.html.

[21] Timothee Cour, Praveen Srinivasan, and Jianbo Shi. Balanced graph matching. In *Advances in Neural Information Processing Systems*, 2007.

[22] Elizabeth D Dolan and Jorge J Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 2002.

[23] Mark Everingham, Luc Van Gool, Christopher K.I. Williams, John Winn, and Andrew Zisserman. The PASCAL visual object classes challenge 2007 results, 2007. http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

[24] Pasquale Foggia, Gennaro Percannella, and Mario Vento. Graph matching and learning in pattern recognition in the last 10 years. *International Journal of Pattern Recognition and Artificial Intelligence*, 2014.

[25] Marguerite Frank, Philip Wolfe, et al. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 1956.

[26] Amir Globerson and Tommi S. Jaakkola. Fixing Max-Product: Convergent message passing algorithms for MAP LP-relaxations. In *Advances in Neural Information Processing Systems*, 2008.

[27] Steven Gold and Anand Rangarajan. A graduated assignment algorithm for graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1996.

[28] Monique Guignard and Siwhan Kim. Lagrangean decomposition: A model yielding stronger Lagrangean bounds. *Mathematical Programming*, 1987.

[29] Tobias Heimann and Hans-Peter Meinzer. Statistical shape models for 3D medical image segmentation: A review. *Medical Image Analysis*, 2009.

[30] Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Cambridge University Press, 2012.

[31] Lisa Hutschenreiter, Stefan Haller, Lorenz Feineis, Carsten Rother, Dagmar Kainmüller, and Bogdan Savchynskyy. Fusion moves for graph matching. In *Proceedings of the IEEE International Conference on Computer Vision*, 2021.

[32] Lisa Hutschenreiter, Stefan Haller, Lorenz Feineis, Carsten Rother, Dagmar Kainmüller, and Bogdan Savchynskyy. Fusion moves for graph matching website, 2021. https://vislearn.github.io/libmpopt/iccv2021/.

[33] Bo Jiang, Jin Tang, Chris Ding, and Bin Luo. A local sparse model for matching problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2015.

[34] Dagmar Kainmueller, Florian Jug, Carsten Rother, and Gene Myers. Active graph matching for automatic joint segmentation and annotation of C. elegans. In *Proceedings of the International Conference on Medical Image Computing and Computer Assisted Intervention*, 2014.

[35] Dagmar Kainmueller, Florian Jug, Carsten Rother, and Gene Myers. Graph matching problems for annotating C. elegans, 2017. https://doi.org/10.15479/AT:ISTA:57.

[36] 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*, 2015.

[37] 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. OpenGM Benchmark, 2015. http://hciweb2.iwr.uni-heidelberg.de/opengm/index.php?l0=benchmark.

[38] Vladimir Kolmogorov. Feature correspondence via graph matching source code, 2015. https://pub.ist.ac.at/~vnk/software.html#GRAPH-MATCHING.

[39] Nikos Komodakis and Nikos Paragios. Beyond loose LP-relaxations: Optimizing MRFs by repairing cycles. In *Proceedings of the European Conference on Computer Vision*, 2008.

[40] J.J. Kosowsky and A.L. Yuille. The invisible hand algorithm: Solving the assignment problem with statistical physics. *Neural Networks*, 1994.

[41] Harold W Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 1955.

[42] Eugene L. Lawler. The quadratic assignment problem. *Management Science*, 1963.

[43] M. Leordeanu and M. Hebert. A spectral technique for correspondence problems using pairwise constraints. In *Proceedings of the IEEE International Conference on Computer Vision*, 2005.

[44] Marius Leordeanu. Efficient methods for graph matching and MAP inference, 2013. https://sites.google.com/site/graphmatchingmethods/.

[45] Marius Leordeanu and Martial Hebert. Cars and motor models. https://datasets.d2.mpi-inf.mpg.de/discrete_cv_problems/car_motor_graph_matching.zip.

[46] Marius Leordeanu, Martial Hebert, and Rahul Sukthankar. An integer projected fixed point method for graph matching and MAP inference. In *Advances in Neural Information Processing Systems*, 2009.

[47] Marius Leordeanu, Rahul Sukthankar, and Martial Hebert. Unsupervised learning for graph matching. *International Journal of Computer Vision*, 2012.

[48] Eliane Maria Loiola, Nair Maria Maia de Abreu, Paulo Oswaldo Boaventura-Netto, Peter Hahn, and Tania Querido. An analytical survey for the quadratic assignment problem. *European Journal of Operational Research*, 2007.

[49] Jiayi Ma, Xingyu Jiang, Aoxiang Fan, Junjun Jiang, and Junchi Yan. Image matching from handcrafted to deep features: A survey. *International Journal of Computer Vision*, 2021.

[50] Panos M. Pardalos, Franz Rendl, and Henry Wolkowicz. The quadratic assignment problem - a survey and recent developments. *Quadratic Assignment and Related Problems*, 1993.

[51] Michal Rolínek, Paul Swoboda, Dominik Zietlow, Anselm Paulus, Vít Musil, and Georg Martius. Deep graph matching via blackbox differentiation of combinatorial solvers. In *Proceedings of the European Conference on Computer Vision*, 2020.

[52] Carsten Rother, Vladimir Kolmogorov, Victor S. Lempitsky, and Martin Szummer. Optimizing binary MRFs via extended roof duality. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2007.

[53] Sartaj Sahni. Computationally related problems. *SIAM Journal on Computing*, 1974.

[54] Bogdan Savchynskyy. Discrete graphical models – an optimization perspective. *Foundations and Trends in Computer Graphics and Vision*, 2019.

[55] Hui Sun, Wenju Zhou, and Minrui Fei. A survey on graph matching in computer vision. In *International Congress on Image and Signal Processing, BioMedical Engineering and Informatics*, 2020.

[56] Paul Swoboda. LPMP source code, 2021. https://github.com/LPMP/LPMP.

[57] 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*, 2017.

[58] Lorenzo Torresani, Vladimir Kolmogorov, and Carsten Rother. Hotel and house-sparse models. https://datasets.d2.mpi-inf.mpg.de/discrete_cv_problems/graph_matching_hotel_house.zip.

[59] Lorenzo Torresani, Vladimir Kolmogorov, and Carsten Rother. A dual decomposition approach to feature correspondence. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2013.

[60] 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*, 2018.

[61] Siddharth Tourani, Alexander Shekhovtsov, Carsten Rother, and Bogdan Savchynskyy. Taxonomy of dual block-coordinate ascent methods for discrete energy minimization. In *Proceedings of the Conference on Artifical Intelligence and Statistics*, 2020.

[62] Joshua T. Vogelstein, John M. Conroy, Vince Lyzinski, Louis J. Podrazik, Steven G. Kratzer, Eric T. Harley, Donniell E. Fishkind, R. Jacob Vogelstein, and Carey E. Priebe. Fast Approximate Quadratic Programming for Graph Matching. *PLOS ONE*, 2015.

[63] Junchi Yan, Xu-Cheng Yin, Weiyao Lin, Cheng Deng, Hongyuan Zha, and Xiaokang Yang. A short survey of recent advances in graph matching. In *Proceedings of the ACM International Conference on Multimedia Retrieval*, 2016.

[64] Alper Yilmaz, Omar Javed, and Mubarak Shah. Object tracking: A survey. *ACM Computing*, 2006.

[65] Ron Zass and Amnon Shashua. Probabilistic graph and hypergraph matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2008.

[66] Zhen Zhang. HungarianBP: Pairwise matching through max-weight bipartite belief propagation source code, 2016. https://github.com/zzhang1987/HungarianBP.

[67] 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*, 2016.

[68] Feng Zhou. Implementation of factorized graph matching, 2018. https://github.com/zhfe99/fgm.

[69] Feng Zhou and Fernando De la Torre. Factorized Graph Matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2016.

Supplementary Material – A Comparative Study of Graph Matching Algorithms in Computer Vision

A1 Equivalence Proofs

Theorem 1. *The graph matching problem and the quadratic assignment problem (QAP) are polynomially reducible to each other.*

While the formal proof can be found below, let us develop an intuition for the problem behind Theorem 1. In general, the main difference between graph matching and the QAP is that while the QAP requires complete matchings, graph matching allows for incomplete matchings, i.e., not every element of the first set has to be assigned to one of the second and vice versa. So the equivalence construction mainly has to deal with these side constraints.

When transitioning from graph matching to the QAP, we go from incomplete to complete matchings. Therefore, the idea is to extend the two sets, usually called $\mathcal{V}$ and $\mathcal{L}$, by "dummy" elements to which all previously unassigned elements can be assigned without changing the total cost, see Figure A1 for an illustration.

On the other hand, when going from the QAP to graph matching, we switch from a complete to an incomplete matching. Here, the idea is to shift the cost structure in such a way that the cost difference of any two complete matchings stays the same, while at the same time guaranteeing that any incomplete matching can be improved by completing it, see Figure A2 for an illustration.

Proof. We will prove Theorem 1 by construction.

Reduction of graph matching to QAP. Consider the graph matching problem, c.f. (1),

$$\min_{x \in \{0,1\}^{\mathcal{V}\times\mathcal{L}}} \sum_{\substack{i,j \in \mathcal{V} \\ s,l \in \mathcal{L}}} c_{is,jl}\, x_{is}\, x_{jl} \tag{A1}$$

$$\text{subject to} \sum_{s \in \mathcal{L}} x_{is} \le 1 \quad \text{for all } i \in \mathcal{V}, \tag{A2}$$

$$\sum_{i \in \mathcal{V}} x_{is} \le 1 \quad \text{for all } s \in \mathcal{L}$$

for finite sets $\mathcal{V}$ and $\mathcal{L}$, and cost function $c\colon (\mathcal{V}\times\mathcal{L})^2 \to \mathbb{R}$.

Let M be the disjoint union of $\mathcal{V}$ and $\mathcal{L}$, $M := \mathcal{V} \dot\cup \mathcal{L}$, and let $w\colon M^4 \to \mathbb{R}$ be defined as

$$w_{is,jl} = w(i,s,j,l) := \begin{cases} c_{is,jl}, & \text{if } i,j \in \mathcal{V}, s,l \in \mathcal{L}, \\ 0, & \text{otherwise.} \end{cases}$$

We can then formulate the QAP

$$\min_{y \in \{0,1\}^{M\times M}} \sum_{\substack{i,j \in M \\ s,l \in M}} w_{is,jl}\, y_{is}\, y_{jl} \tag{A3}$$

$$\text{subject to} \sum_{s \in M} y_{is} = 1 \quad \text{for all } i \in M, \tag{A4}$$

$$\sum_{i \in M} y_{is} = 1 \quad \text{for all } s \in M.$$

Let $y^\star$ be a solution of (A3) satisfying (A4), i.e.,

$$y^\star \in \operatorname*{arg\,min}_{y \in \{0,1\}^{M\times M}} \sum_{\substack{i,j \in M \\ s,l \in M}} w_{is,jl}\, y_{is}\, y_{jl}$$

$$\text{s.t.} \sum_{s \in M} y_{is} = 1 \quad \text{for all } i \in M,$$

$$\sum_{i \in M} y_{is} = 1 \quad \text{for all } s \in M.$$

Now let $x^\star \in \{0,1\}^{\mathcal{V}\times\mathcal{L}}$ be defined as $x^\star_{is} = y^\star_{is}$ for all $i \in \mathcal{V}$, $s \in \mathcal{L}$. It remains to show that $x^\star$ then is a solution of (A1) satisfying (A2).

First note that $x^\star$ satisfies the constraints (A2) since for all $i \in \mathcal{V}$

$$\begin{aligned} \sum_{s \in \mathcal{L}} x^\star_{is} &= \sum_{s \in \mathcal{L}} y^\star_{is} \\ &\le \sum_{s \in M} y^\star_{is} && \text{as } \mathcal{L} \subseteq M \text{ and } y^\star \in \{0,1\}^{M\times M}, \\ &\le 1 && \text{as } y^\star \text{ satisfies (A4).} \end{aligned}$$

Analogously, $\sum_{i \in \mathcal{V}} x^\star_{is} \le 1$ for all $s \in \mathcal{L}$.

Suppose now that $x^\star$ is not a solution of (A1), i.e., there exists $x' \in \{0,1\}^{\mathcal{V}\times\mathcal{L}}$ satisfying (A2) with

$$\sum_{\substack{i,j \in \mathcal{V} \\ s,l \in \mathcal{L}}} c_{is,jl}\, x'_{is}\, x'_{jl} < \sum_{\substack{i,j \in \mathcal{V} \\ s,l \in \mathcal{L}}} c_{is,jl}\, x^\star_{is}\, x^\star_{jl}.$$

We can then define $y' \in \{0,1\}^{M\times M}$ by

$$y'_{is} := \begin{cases} x'_{is}, & \text{if } i \in \mathcal{V}, s \in \mathcal{L}, \\ x'_{si}, & \text{if } i \in \mathcal{L}, s \in \mathcal{V}, \\ 1, & \text{if } i \in \mathcal{V}, s = i, \text{ and } \sum_{l \in \mathcal{L}} x'_{il} < 1, \\ 1, & \text{if } i \in \mathcal{L}, s = i, \text{ and } \sum_{j \in \mathcal{V}} x'_{js} < 1, \\ 0, & \text{otherwise.} \end{cases}$$

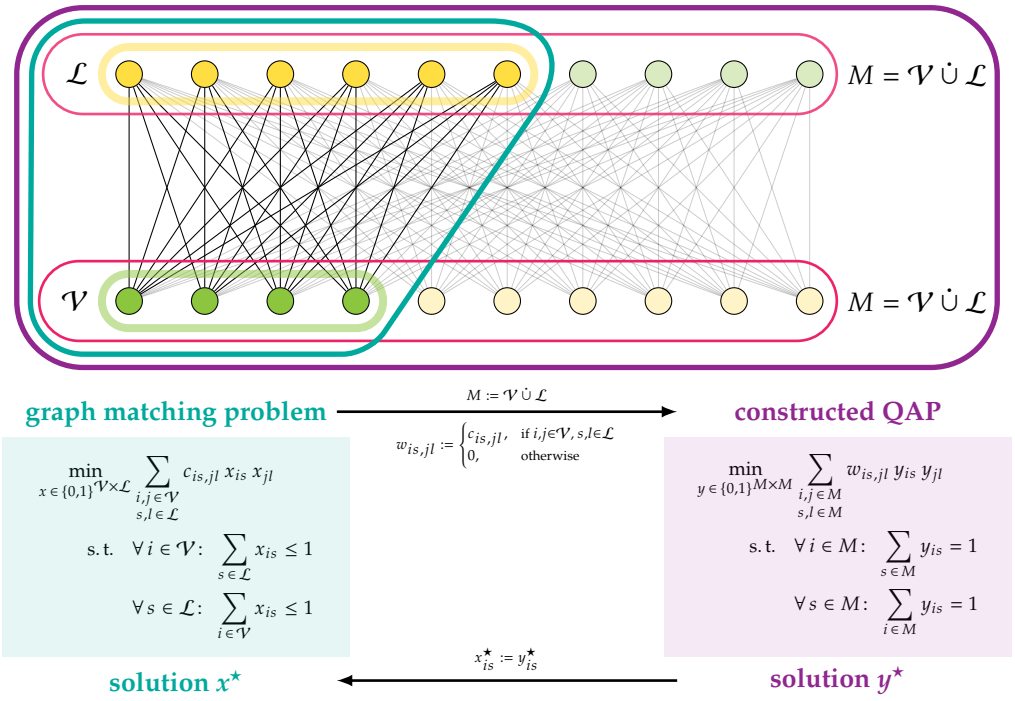


Figure A1: From graph matching to the QAP. The notation corresponds to the notation used in the proof of Theorem 1. The idea behind reducing graph matching to quadratic assignment is to enable completion of the possibly incomplete matchings feasible for graph matching. This is done by providing "dummy" nodes to which those nodes can be matched that would otherwise be left unassigned. As potentially all nodes could be unassigned, we provide a corresponding dummy for each.

Observe that y' satisfies the constraints (A4) since for all $i \in M$

$$\begin{aligned}\sum_{s\in M} y'_{is} &= \sum_{s\in\mathcal{V}} y'_{is} + \sum_{s\in\mathcal{L}} y'_{is} \\ &= \begin{cases} y'_{ii} + \sum_{s\in\mathcal{L}} x'_{is} & \text{if } i \in \mathcal{V} \\ \sum_{s\in\mathcal{V}} x'_{si} + y'_{ii} & \text{if } i \in \mathcal{L} \end{cases} \\ &= \begin{cases} 1 - \sum_{l\in\mathcal{L}} x'_{il} + \sum_{s\in\mathcal{L}} x'_{is} & \text{if } i \in \mathcal{V} \\ \sum_{s\in\mathcal{V}} x'_{si} + 1 - \sum_{j\in\mathcal{V}} x'_{ji} & \text{if } i \in \mathcal{L} \end{cases} \\ &= 1,\end{aligned}$$

and, analogously, $\sum_{i\in M} y'_{is} = 1$ for all $s \in M$. Then we obtain

$$\begin{aligned}&\sum_{\substack{i,j\in M\\ s,l\in M}} w_{is,jl}\, y'_{is}\, y'_{jl} \\ &\quad= \sum_{\substack{i,j\in\mathcal{V}\\ s,l\in\mathcal{L}}} c_{is,jl}\, y'_{is}\, y'_{jl} && \text{by definition of } w, \\ &\quad= \sum_{\substack{i,j\in\mathcal{V}\\ s,l\in\mathcal{L}}} c_{is,jl}\, x'_{is}\, x'_{jl} && \text{by definition of } y', \\ &\quad< \sum_{\substack{i,j\in\mathcal{V}\\ s,l\in\mathcal{L}}} c_{is,jl}\, x^\star_{is}\, x^\star_{jl} && \text{due to the choice of } x', \\ &\quad= \sum_{\substack{i,j\in\mathcal{V}\\ s,l\in\mathcal{L}}} c_{is,jl}\, y^\star_{is}\, y^\star_{jl} && \text{by definition of } x^\star, \\ &\quad= \sum_{\substack{i,j\in M\\ s,l\in M}} w_{is,jl}\, y^\star_{is}\, y^\star_{jl} && \text{by definition of } w, \\ &\quad\le \sum_{\substack{i,j\in M\\ s,l\in M}} w_{is,jl}\, y'_{is}\, y'_{jl} && \text{since } y^\star \text{ solution of (A3)},\end{aligned}$$

which is a contradiction. Hence, $x^\star$ is a solution of (A1).

Therefore, graph matching is polynomially reducible to the quadratic assignment problem as the size of the constructed QAP is polynomial in the size of the initial graph matching problem, and each solution of the QAP directly induces a solution to the graph matching problem.

Reduction of QAP to graph matching. Consider the QAP

$$\min_{x\in\{0,1\}^{M\times M}} \sum_{\substack{i,j\in M\\ s,l\in M}} w_{is,jl}\, x_{is}\, x_{jl} \tag{A5}$$

$$\text{subject to } \sum_{s\in M} x_{is} = 1 \quad \text{for all } i \in M, \tag{A6}$$

$$\sum_{i\in M} x_{is} = 1 \quad \text{for all } s \in M,$$

for a finite set M and cost function $w\colon M^4 \to \mathbb{R}$.

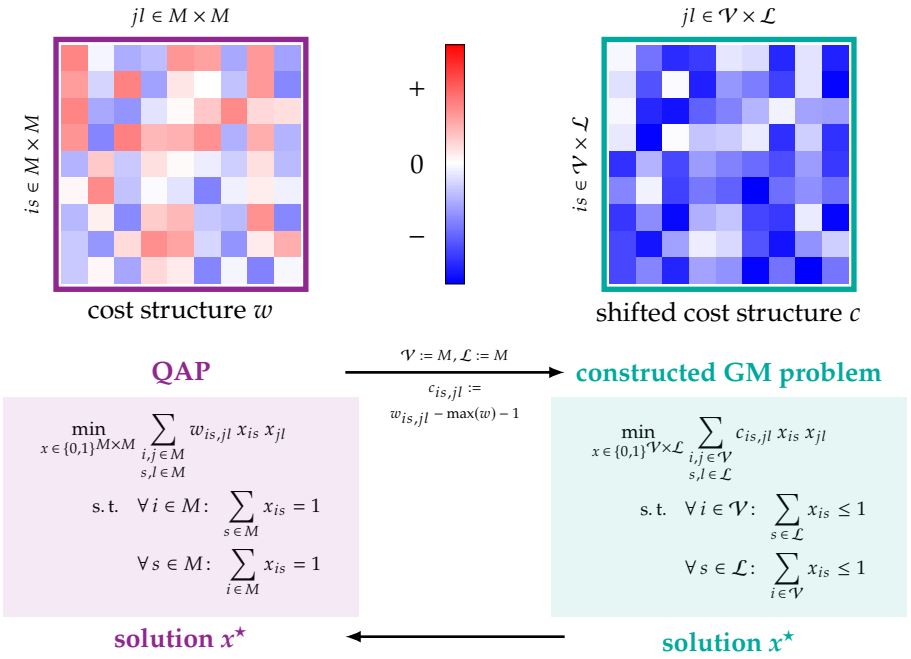


Figure A2: From the QAP to graph matching (GM). The notation corresponds to the notation used in the proof of Theorem 1. The idea behind reducing quadratic assignment to graph matching is to make sure that while the requirement for complete matchings is dropped, a better objective value can always be obtained when completing a given incomplete matching. This is guaranteed by shifting the cost structure to incur negative costs for all assignments.

We can now formulate the graph matching problem

$$\min_{x \in \{0,1\}^{M \times M}} \sum_{\substack{i,j \in M \\ s,l \in M}} c_{is,jl}\, x_{is}\, x_{jl} \tag{A7}$$

$$\text{subject to} \sum_{s \in M} x_{is} \le 1 \quad \text{for all } i \in M, \tag{A8}$$

$$\sum_{i \in M} x_{is} \le 1 \quad \text{for all } s \in M,$$

where $\mathcal{V} = \mathcal{L} = M$, and $c_{is,jl} := w_{is,jl} - \max(w) - 1$ for all $i, j, s, l \in M$. Note that by definition $c_{is,jl} < 0$ for all $i, j, s, l \in M$.

Let $x^\star$ be a solution of (A7) satisfying (A8). We now want to prove that $x^\star$ also satisfies (A6) and is a solution of (A5).

Suppose that $x^\star$ does not satisfy (A6), i.e., there exists $i' \in M$ with $\sum_{s \in M} x^\star_{i's} < 1$. Then there also exists $s' \in M$ with $\sum_{i \in M} x^\star_{is'} < 1$. Observe that $x^\star_{i's} = 0$ for all $s \in M$, and $x^\star_{is'} = 0$ for all $i \in M$ since $x^\star \in \{0,1\}^{M \times M}$.

We can now define x' as

$$x'_{is} = \begin{cases} 1, & \text{if } i = i',\ s = s', \\ x^\star_{is}, & \text{otherwise.} \end{cases}$$

Due to the choice of i' and s', x' also satisfies the constraints (A8). Furthermore,

$$\begin{aligned} &\sum_{\substack{i,j \in M \\ s,l \in M}} c_{is,jl}\, x'_{is}\, x'_{jl} \\ &= \sum_{\substack{i,j \in M \\ s,l \in M}} c_{is,jl}\, x^\star_{is}\, x^\star_{jl} + \sum_{\substack{i,s \in M \\ is \neq i's'}} c_{is,i's'}\, x^\star_{is} \\ &\quad + \sum_{\substack{j,l \in M \\ jl \neq i's'}} c_{i's',jl}\, x^\star_{jl} + c_{i's',i's'} \\ &< \sum_{\substack{i,j \in M \\ s,l \in M}} c_{is,jl}\, x^\star_{is}\, x^\star_{jl}, \end{aligned}$$

since $c_{is,jl} < 0$ for all $i, j, s, l \in M$. This contradicts $x^\star$ being a solution of (A7). Hence, $x^\star$ satisfies (A6).

Suppose now that $x^\star$ is not a solution of (A5), i.e., there exists $\overline{x} \in \{0,1\}^{M \times M}$ satisfying (A6) with

$$\sum_{\substack{i,j \in M \\ s,l \in M}} w_{is,jl}\, \overline{x}_{is}\, \overline{x}_{jl} < \sum_{\substack{i,j \in M \\ s,l \in M}} w_{is,jl}\, x^\star_{is}\, x^\star_{jl}.$$

Note that $\overline{x}$ also satisfies (A8) as it satisfies (A6).

Due to the constraints (A6), $x^\star$ and $\overline{x}$ have exactly $|M|$ nonzero entries, hence when summing over all $i, j, s, l \in M$, both, $x^\star_{is} x^\star_{jl}$ and $\overline{x}_{is}\overline{x}_{jl}$, are nonzero exactly $|M|^2$ times. Therefore, we obtain

$$\begin{aligned} &\sum_{\substack{i,j \in M \\ s,l \in M}} w_{is,jl}\, \overline{x}_{is}\, \overline{x}_{jl} - |M|^2(\max(w) + 1) \\ &\quad < \sum_{\substack{i,j \in M \\ s,l \in M}} w_{is,jl}\, x^\star_{is}\, x^\star_{jl} - |M|^2(\max(w) + 1) \end{aligned}$$

$$\sum_{\substack{i,j \in M \\ s,l \in M}} \left(w_{is,jl} - \max(w) - 1 \right) \overline{x}_{is}\, \overline{x}_{jl}$$
$$< \sum_{\substack{i,j \in M \\ s,l \in M}} \left(w_{is,jl} - \max(w) - 1 \right) x^\star_{is}\, x^\star_{jl}$$
$$\sum_{\substack{i,j \in M \\ s,l \in M}} c_{is,jl}\, \overline{x}_{is}\, \overline{x}_{jl} < \sum_{\substack{i,j \in M \\ s,l \in M}} c_{is,jl}\, x^\star_{is}\, x^\star_{jl}$$

This would contradict $x^\star$ being a solution of (A7). Thus, $x^\star$ is a solution of (A5).

This proves that the quadratic assignment problem is polynomially reducible to graph matching as the size of the constructed graph matching problem is polynomial in the size of the initial QAP. Each solution of the graph matching problem directly yields a solution of the QAP. □

Remark A1. *Note that a polynomial reduction from the QAP to graph matching can also be obtained by shifting only the unary costs by a sufficiently large constant, i.e., by defining*

$$c_{is,jl} = \begin{cases} w_{is,jl} - K, & \text{if } jl = is,\ i,j,s,l \in M \\ w_{is,jl}, & \text{otherwise.} \end{cases}$$

for sufficiently large K. This is relevant in practice as it only changes the diagonal of the cost matrix, and not the full matrix. In this way it does not as heavily influence the sparsity of the cost structure.

The statements in the proof of Theorem 1 hold in particular if all quadratic terms incur a cost of zero, so we can reduce the objective function to its linear component, i.e., the objective function

$$\sum_{\substack{i,j \in \mathcal{V} \\ s,l \in \mathcal{L}}} c_{is,jl}\, x_{is}\, x_{jl}$$

of the graph matching problem, c.f. (A1) in the proof above, can be reduced to

$$\sum_{\substack{i \in \mathcal{V} \\ s \in \mathcal{L}}} c_{is}\, x_{is},$$

which corresponds to the incomplete linear assignment problem. Similarly, we can replace the objective function

$$\sum_{\substack{i,j \in M \\ s,l \in M}} w_{is,jl}\, y_{is}\, y_{jl}$$

of the quadratic assignment problem, c.f. (A5), by

$$\sum_{i,s \in M} w_{is}\, y_{is},$$

which corresponds to the objective of the linear assignment problem. With this in mind, the corollary below directly follows from Theorem 1:

Corollary 1. *The incomplete linear assignment problem and the linear assignment problem (LAP) are polynomially reducible to each other.*

A2 Symmetry of the Formulations

Note that formulations (3) and (4)-(6) are asymmetric, since the sets $\mathcal{V}$ and $\mathcal{L}$ play different roles in the formulations, in contrast to the symmetric formulation (1). Although swapping the roles of $\mathcal{V}$ and $\mathcal{L}$ does not influence the optimal solutions, it influences the problem structure, such as the sparsity of the resulting graphical model, as well as the tightness of the corresponding LP relaxation. Symmetrized formulations based on the graphical model representation use two graphical models, with the role of $\mathcal{V}$ and $\mathcal{L}$ being swapped in one of them [57]. These graphical models, with variables denoted correspondingly by x and $\hat{x}$, are coupled by additional constraints which enforce either the unary variables to be equal in both representations, i.e., $x_{is} = \hat{x}_{si}$, or the pairwise variables, i.e., $x_{is,jl} = \hat{x}_{si,lj}$. The first type of coupling constraints is computationally cheaper, whereas the second one leads to a tighter LP relaxation equivalent to the standard one considered in operations research [1]. In our experimental evaluation the second type of constraints is implicitly used by `dd-ls*` algorithms.

A3 Data Format (dd-format)

As a unified data format for all datasets we use the one introduced by [59]. It encodes a sparse representation of the graph matching problem (1) in plain text. Due to its origin we refer to it as *dd-* or *dual decomposition* data format. It is used by `dd-ls*`, as well as `fm`, `fm-bca` and `mp-*` algorithms as a native input. Table A1 presents an overview of the file format and shows a description of of all input line types.

With other algorithms suitable converters are used. The one for the graphical model representation (3) is readily available at [32]. For Matlab algorithms a matrix representation of (1) is required. We implemented this conversion by ourselves and will make the code available.

A4 Cost Transformations

Different methods have different requirements for the cost matrix, see columns *bijective*, *non-positive* and *zero unary* in Table 1. For datasets where the costs do not fulfill those requirements, see equivalent columns in Table 3, the costs have to be transformed to make them amenable to those algorithms. The transformations are chosen to in a way such that **(i)** the cost difference between any two matchings stays the same; and **(ii)** the

input line	description
`c comment line`	comments are ignored
`p <#V> <#L> <#A> <#E>`	prologue: $\|\mathcal{V}\|$, $\|\mathcal{L}\|$, number of assignments and edges to follow
`a <id> <i> <s> <cost>`	specifies possible assignment $i \to s$ for $i \in \mathcal{V}$, $s \in \mathcal{L}$ (unary terms $c_{is,is} x_{is}$)
`e <id1> <id2> <cost>`	specifies edge between two assignments (pairwise terms $(c_{is,jl} + c_{jl,is}) x_{is} x_{jl}$)
`i0 <i> <x> <y>`	optional: coordinate of the node $i \in \mathcal{V}$ as a point in the left image
`i1 <s> <x> <y>`	optional: coordinate of the node $s \in \mathcal{L}$ as a point in the right image
`n0 <i> <j>`	optional: specifies that points $i, j \in \mathcal{V}$ are neighbors in the left image
`n1 <s> <l>`	optional: specifies that points $s, l \in \mathcal{L}$ are neighbors in the right image

Table A1: Short specification of the dd file format. The format originated in [59]. It encodes a sparse representation of the graph matching problem (1). Unary costs are encoded as *assignments* and pairwise costs are encoded as *edges* between two assignments. Any lines marked as optional or comments are ignored when building the graph matching problem. They were introduced in [59] to ease visualization of matching problems between two images.

Algorithm 1: Transform into non-positive costs.

Input: Graph matching problem $(\mathcal{V}, \mathcal{L}, c)$
Output: Equivalent non-positive cost vector c'

`/* initialize */`
$c' \leftarrow (0 \dots 0)$

`/* first, shift unary costs */`
for $i \in \mathcal{V}$ and $s \in \mathcal{L}$ **do**
  $\alpha \leftarrow \max\{c_{is'} \mid s' \in \mathcal{L} \text{ and } c_{is'} \neq \infty\}$
  $c'_{is} \leftarrow c_{is} - \max\{0, \alpha\}$

`/* second, shift pairwise costs */`
for $i, j \in \mathcal{V}, i < j$ and $s, t \in \mathcal{L}$ **do**
  $\alpha \leftarrow \max\{c_{is',jl'} \mid s', l' \in \mathcal{L}\}$
  $c'_{is,jl} \leftarrow c_{is,jl} - \max\{0, \alpha\}$

sparsity of the cost matrix is preserved as much as possible. In the following we describe each of the three transformations. The full implementation is part of our published source code for the benchmark. Without loss of generality we assume the cost matrix to be upper triangular, i.e. $c_{is,jl} = 0$ for $i, j \in \mathcal{V}$, $s, l \in \mathcal{L}$ and $i > j$.

Bijective Costs. Non-bijective problem instances are transformed into the bijective ones in a way similar to the described in the proof of Theorem 1 and illustrated in Fig. A1.

Non-positive Costs. Here we transform the cost matrix into a form where all *finite* costs are non-positive. Note that infinite costs will stay positive, as they are identified by corresponding algorithms and prohibit selection of the associated matchings. Algorithm 1 shows the transformation which shifts the costs of all bijective assignments by a fixed constant. Therefore, for non-bijective instances we first transform them into the bijective form as described in previous paragraph.

Remove Unary Costs. The `pm` algorithm does not take into account unary costs, see column *zero unary* in Table 1. We use Algorithm 2 to transform unary costs into pairwise costs.

Algorithm 2: Transform unary into pairwise costs.

Input: Graph matching problem $(\mathcal{V}, \mathcal{L}, c)$
Output: Equivalent cost vector c' with empty diagonal

`/* initialize */`
for $i, j \in \mathcal{V}$ and $s, l \in \mathcal{L}$ **do**
  $c'_{is,jl} \leftarrow \begin{cases} c_{is,jl} & \text{if } (i,s) \neq (j,l) \\ 0 & \text{otherwise} \end{cases}$

`/* count how many additional cost entries are necessary; remember decision with least number of additional entries in array` J `*/`
for $i \in \mathcal{V}$ and $s \in \mathcal{L}$ **do**
  $J_{is} \leftarrow \arg\min_{j \in \mathcal{V}} |\{l \in \mathcal{L} \mid c_{is,jl} = 0 \text{ and } i \neq j, s \neq l\}|$

`/* distribute unaries across pairwise costs */`
for $i, j \in \mathcal{V}, i < j$ and $s, l \in \mathcal{L}$ **do**
  if $J_{is} = j$ **then**
    $c'_{is,jl} \leftarrow c'_{is,jl} + c_{is,is}$
  if $J_{jl} = i$ **then**
    $c'_{is,jl} \leftarrow c'_{is,jl} + c_{jl,jl}$

For the maximization algorithms `ipfpu`, `ipfps`, `ga`, `rrwm`, `pm`, `sm`, `smac`, `lsm`, `mpm`, `fgmd` and `hbp` the sign of the elements in the cost matrix is flipped additionally after performing all other operations.

A5 Accuracy computation

For `hotel`, `house-sparse`, `house-dense`, `car` and `motor` the complete ground truth assignments are known. The accuracy is computed as the number of correctly assigned nodes over $n_{\mathcal{V}}$. The ground truth for `caltech-small`, `caltech-large`, `worms` and `pairs` is only partial i.e. not every node has a ground truth label. This is taken into account by computing the accuracy as the number of correctly assigned nodes over

method	command line
dd-ls0	`tkrgm --linear --tree --max-iter N`
dd-ls3	`tkrgm --linear --local 2 --tree --max-iter N`
dd-ls4	`tkrgm --linear --local 3 --tree --max-iter N`
fw	`graph_matching_frank_wolfe_text_input input.dd`
mp	`graph_matching_mp -i input.dd --roundingReparametrization uniform:0.5`
mp-mcf	`graph_matching_mp_tightening -i input.dd --tighten --tightenInterval 50 \ --tightenIteration 200 --tightenConstraintsPercentage 0.01 \ --tightenReparametrization uniform:0.5 --graphMatchingRounding mcf`
mp-fw	`graph_matching_mp_tightening -i input.dd --tighten --tightenInterval 50 \ --tightenIteration 200 --tightenConstraintsPercentage 0.01 \ --tightenReparametrization uniform:0.5 --graphMatchingRounding fw`
fm	`qap_dd --max-batches N --batch-size 0 --generate 1`
fm-bca	`qap_dd --max-batches N --batch-size 10 --generate 10`

Table A2: Command line parameters for all C++ methods.

the number of nodes with available ground truth label.

A6 Run Time Measurement

To make the comparison as fair as possible, we exclude preprocessing steps and the time it takes to load the graph matching files into the solver. This means that the following steps are excluded from our run-time measurements: **(i)** Conversion process from the dd file format into the input format of the solvers; **(ii)** Loading the file from storage into memory; **(iii)** Parsing the input file; and **(iv)** Dataset transformation if needed.

Practically, we start the timer directly before the solver starts the optimization routine. For optimization methods that have not included this functionality, we have modified the methods accordingly.

A7 Reproducibility

Original Source Code. References with links to the original source code can be found in column *Matlab* and *C++* in Table 1. Note that for the `dd-ls*` methods we use the same command line wrapper as used in [31]. The wrapper allows to select the problem decomposition at run-time without need for recompilation. The specific command lines for the C++ programs (see column *C++* in Table 1) that have been used in the benchmark are listed in Table A2.

Matlab Wrappers. We incorporated the implementations of the Matlab algorithms in our own wrappers in order to be able to load the datasets and to make the output standardized. No parameters have to be specified.

Reproducible Containers. For all methods we build Linux containers so that the environment as well as the specific version of the solvers are reproducible. The container files contain instructions for downloading, processing and building a fixed version of the corresponding methods. The source code of each method is fetched from the Internet. For the case that source code is no longer online, we preserve a historical copy of the repositories. The container build scripts will pin each method to a specific version or commit id to preserve reproducibility.

HPC Scripts. For usage in high-performance compute clusters (HPC clusters) we provide a small script to transform the container images into singluarity image. Most HPC clusters provide tools to easily run these singularity images. Additionally, we provide SLURM scheduling scripts so that the benchmark can be reproduced easily and quickly.

Project Web Site. The wrappers, container files and other scripts are publicly available on the project web site, see https://vislearn.github.io/gmbench/.

A8 Detailed Evaluation Results

More detailed evaluation results are provided on the following pages. Figure A3 shows run-time performance profiles similar to Figure 1 but for each dataset separately. Tables A3 to A13 show the average results of each method on each dataset for time limits 1s, 10s, 100s and 300s similar to Table 4 and Table 5.

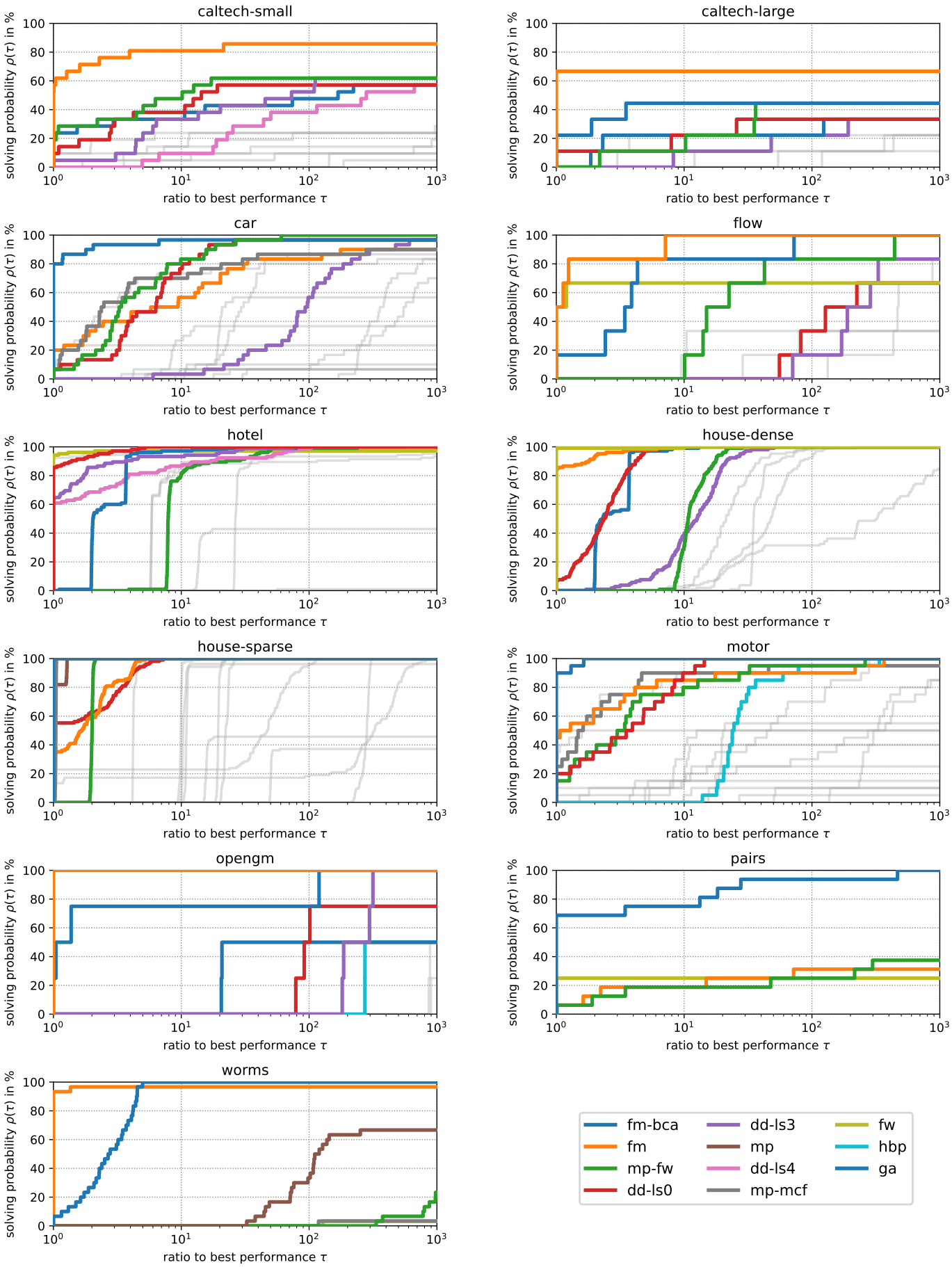


Figure A3: Run time performance profile [22] per dataset.

	caltech-small (1s)				caltech-small (10s)				caltech-small (100s)				caltech-small (300s)			
	opt	E	D	acc	opt	E	D	acc	opt	E	D	acc	opt	E	D	acc
fgmd	0		—*		14		—*		29		—*		43		—*	
fm	**52**	-8906	-43926	58	**57**	-9040	-43926	62	**57**	-9050	-43926	63	**57**	-9050	-43926	63
fw	0	0	–	0	0	0	–	0	0	0	–	0	0	0	–	0
ga	0		—*		5		—*		5		—*		5		—*	
ipfps	19	**-8983**	–	67	19	-8983	–	67	19	-8983	–	67	19	-8983	–	67
ipfpu	10	-8829	–	62	10	-8829	–	62	10	-8829	–	62	10	-8829	–	62
lsm	0		—*		0	0	–	0	0	0	–	0	0	0	–	0
mpm	0		—*		0		—*		0		—*		0		—*	
pm	0	-6510	–	51	0	-6510	–	51	0	-6510	–	51	0	-6510	–	51
rrwm	5		—*		5	-8848	–	61	5	-8848	–	61	5	-8848	–	61
sm	0	-3932	–	36	0	-3932	–	36	0	-3932	–	36	0	-3932	–	36
smac	0	-6196	–	42	0	-6196	–	42	0	-6196	–	42	0	-6196	–	42
dd-ls0	43	-7414	**-10658**	58	48	-8251	**-9502**	60	48	-8251	-9502	60	48	-8251	-9502	60
dd-ls3	33	-6842	-15159	57	52	-7862	-9904	60	52	-8357	**-9466**	61	52	-8357	-9466	61
dd-ls4	24	-6332	-18033	54	38	-6919	-13407	53	52	-7878	-9810	60	52	-8324	**-9443**	61
fm-bca	38	-8927	-12827	62	43	-8943	-12797	62	48	-8953	-12797	61	52	-8960	-12797	61
hbp	0		—*		5		—*		10		—*		10		—*	
mp	14	-7967	-12191	59	14	-8026	-12183	60	19	-8095	-12182	62	19	-8095	-12182	62
mp-fw	33	-8886	-12793	60	38	**-9052**	-12244	65	38	**-9056**	-11217	65	43	**-9057**	-10818	64
mp-mcf	5	-7882	-12380	60	10	-8030	-11737	59	14	-8324	-10987	57	14	-8435	-10731	57

opt: optimally solved instances (%); **E:** average best objective value; **D:** average best lower bound if applicable;
acc: average accuracy corresponding to best objective (%); **—*:** method yields no solution for at least one problem instance

Table A3: Detailed fixed-time evaluation for dataset caltech-small.

	caltech-large (1s)			caltech-large (10s)			caltech-large (100s)			caltech-large (300s)		
	E	D	acc	E	D	acc	E	D	acc	E	D	acc
fgmd		—*			—*			—*			—*	
fm	-33972	-142829	52	-34117	-142829	52	-34125	-142829	52	-34125	-142829	52
fw	0	–	0	0	–	0	0	–	0	0	–	0
ga		—*			—*			—*			—*	
ipfps		—*			—*		-33998	–	51	-33998	–	51
ipfpu		—*		-34216	–	52	-34216	–	52	-34216	–	52
lsm		—*			—*		0	–	0	0	–	0
mpm		—*			—*			—*			—*	
pm	-29106	–	43	-29106	–	43	-29106	–	43	-29106	–	43
rrwm		—*			—*			—*			—*	
sm		—*		-14423	–	28	-14423	–	28	-14423	–	28
smac		—*		-24183	–	39	-24183	–	39	-24183	–	39
dd-ls0	-26236	-56071	48	-32973	**-35007**	51	-33539	-34959	52	-33539	-34959	52
dd-ls3	-25226	-72766	44	-28653	-42079	49	-33552	**-34914**	49	-33557	**-34911**	49
dd-ls4	-25268	-79327	46	-25599	-62120	46	-30148	-38880	51	-32266	-35577	51
fm-bca	-33758	-48582	51	-34040	-48223	51	-34073	-48217	51	-34082	-48217	51
hbp		—*			—*			—*			—*	
mp	-31315	**-46170**	48	-32017	-46070	48	-32069	-46066	48	-32074	-46066	48
mp-fw	**-34227**	-51526	51	**-34237**	-48882	51	**-34277**	-45923	51	**-34287**	-44424	51
mp-mcf	-29813	-48168	45	-30362	-46630	47	-30737	-43833	47	-31230	-42564	48

opt: optimally solved instances (%); **E:** average best objective value; **D:** average best lower bound if applicable;
acc: average accuracy corresponding to best objective (%); **—*:** method yields no solution for at least one problem instance

Table A4: Detailed fixed-time evaluation for dataset caltech-large.

	car (1s)				car (10s)				car (100s)				car (300s)			
	opt	E	D	acc	opt	E	D	acc	opt	E	D	acc	opt	E	D	acc
fgmd	0		—*		83		—*		83	**-69**	–	89	83	**-69**	–	89
fm	77	**-69**	-107	88	90	**-69**	-107	89	90	**-69**	-107	89	90	**-69**	-107	89
fw	7	-63	–	63	7	-63	–	63	7	-63	–	63	7	-63	–	63
ga	57	-68	–	84	57	-68	–	84	57	-68	–	84	57	-68	–	84
ipfps	10	-65	–	80	10	-65	–	80	10	-65	–	80	10	-65	–	80
ipfpu	7	-60	–	69	7	-60	–	69	7	-60	–	69	7	-60	–	69
lsm	0	-51	–	52	0	-51	–	52	0	-51	–	52	0	-51	–	52
mpm	7		—*		7	-57	–	65	7	-57	–	65	7	-57	–	65
pm	0	-35	–	23	0	-35	–	23	0	-35	–	23	0	-35	–	23
rrwm	37	-68	–	87	37	-68	–	87	37	-68	–	87	37	-68	–	87
sm	7	-63	–	76	7	-63	–	76	7	-63	–	76	7	-63	–	76
smac	0	-52	–	52	0	-52	–	52	0	-52	–	52	0	-52	–	52
dd-ls0	**97**	**-69**	**-69**	91	**97**	**-69**	**-69**	91	97	**-69**	**-69**	91	97	**-69**	**-69**	91
dd-ls3	47	-57	-75	74	87	-68	**-70**	90	97	**-69**	**-69**	91	97	**-69**	**-69**	91
dd-ls4	3	-49	-78	59	57	-59	-73	77	87	-67	**-70**	89	97	**-69**	**-69**	92
fm-bca	93	**-69**	-71	92	**97**	**-69**	-71	92	97	**-69**	-71	92	97	**-69**	-71	92
hbp	77		—*		87		—*		87	**-69**	-73	91	87	**-69**	-73	91
mp	80	**-69**	-71	92	83	**-69**	-71	92	87	**-69**	-71	92	87	**-69**	-71	92
mp-fw	90	**-69**	-71	91	**97**	**-69**	-70	91	**100**	**-69**	**-70**	91	**100**	**-69**	**-70**	91
mp-mcf	87	**-69**	-70	91	90	**-69**	-70	91	93	**-69**	**-70**	91	93	**-69**	**-70**	91

opt: optimally solved instances (%); **E:** average best objective value; **D:** average best lower bound if applicable;
acc: average accuracy corresponding to best objective (%); **—*:** method yields no solution for at least one problem instance

Table A5: Detailed fixed-time evaluation for dataset car.

	flow (1s)			flow (10s)			flow (100s)			flow (300s)		
	opt	E	D	opt	E	D	opt	E	D	opt	E	D
fgmd	0	—*		0	—*		0	—*		0	—*	
fm	**83**	**-2838**	-3436	**100**	**-2840**	-3436	**100**	**-2840**	-3436	**100**	**-2840**	-3436
fw	67	-2828	–	67	-2828	–	67	-2828	–	67	-2828	–
ga	0	—*		0	—*		0	-2469	–	0	-2469	–
ipfps	0	-766	–	0	-766	–	0	-766	–	0	-766	–
ipfpu	0	-512	–	0	-512	–	0	-512	–	0	-512	–
lsm	0	—*		0	—*		0	31042	–	0	31042	–
mpm	0	—*		0	—*		0	—*		0	—*	
pm	0	-32	–	0	-32	–	0	-32	–	0	-32	–
rrwm	0	—*		0	-226	–	0	-226	–	0	-226	–
sm	0	-139	–	0	-139	–	0	-139	–	0	-139	–
smac	0	0	–	0	0	–	0	0	–	0	0	–
dd-ls0	33	-2345	-2968	67	-2819	-2854	67	-2819	-2854	67	-2819	-2854
dd-ls3	17	-2059	-3030	83	-2821	**-2847**	83	-2834	-2844	83	-2834	-2844
dd-ls4	0	-2062	-3090	67	-2767	-2863	83	-2835	**-2843**	83	-2835	**-2843**
fm-bca	**83**	**-2838**	-2898	**100**	**-2840**	-2879	**100**	**-2840**	-2878	**100**	**-2840**	-2878
hbp	0	—*		0	—*		0	—*		0	—*	
mp	33	-2628	**-2887**	33	-2674	-2882	50	-2690	-2881	50	-2690	-2881
mp-fw	**83**	—*		83	—*		83	-2838	-2870	83	-2838	-2862
mp-mcf	33	-2521	-2892	33	-2719	-2869	50	-2749	-2854	50	-2789	-2851

opt: optimally solved instances (%); **E:** average best objective value; **D:** average best lower bound if applicable;
acc: average accuracy corresponding to best objective (%); **—*:** method yields no solution for at least one problem instance

Table A6: Detailed fixed-time evaluation for dataset flow.

	hotel (1s)				hotel (10s)				hotel (100s)				hotel (300s)			
	opt	E	D	acc	opt	E	D	acc	opt	E	D	acc	opt	E	D	acc
fgmd	0		—*		0		—*		96	-4283	–	98	96	-4283	–	98
fm	97	-4292	-4406	100	97	-4292	-4406	100	97	-4292	-4406	100	97	-4292	-4406	100
fw	97	-4288	–	99	97	-4288	–	99	97	-4288	–	99	97	-4288	–	99
ga	0	947	–	15	0	947	–	15	0	947	–	15	0	947	–	15
ipfps	0	1051	–	14	0	1051	–	14	0	1051	–	14	0	1051	–	14
ipfpu	0	1062	–	15	0	1062	–	15	0	1062	–	15	0	1062	–	15
lsm	0		—*		0	1729	–	12	0	1729	–	12	0	1729	–	12
mpm	43	-2585	–	78	43	-2585	–	78	43	-2585	–	78	43	-2585	–	78
pm	0	775	–	33	0	775	–	33	0	775	–	33	0	775	–	33
rrwm	0	744	–	15	0	744	–	15	0	744	–	15	0	744	–	15
sm	0	1086	–	13	0	1086	–	13	0	1086	–	13	0	1086	–	13
smac	1	-1571	–	61	1	-1571	–	61	1	-1571	–	61	1	-1571	–	61
dd-ls0	**100**	**-4293**	**-4294**	100	**100**	**-4293**	-4294	100	**100**	**-4293**	-4294	100	**100**	**-4293**	-4294	100
dd-ls3	**100**	**-4293**	**-4294**	100	**100**	**-4293**	**-4293**	100	**100**	**-4293**	**-4293**	100	**100**	**-4293**	**-4293**	100
dd-ls4	98	-4291	-4297	100	**100**	**-4293**	**-4293**	100	**100**	**-4293**	**-4293**	100	**100**	**-4293**	**-4293**	100
fm-bca	**100**	**-4293**	-4300	100	**100**	**-4293**	-4300	100	**100**	**-4293**	-4300	100	**100**	**-4293**	-4300	100
hbp	97		—*		**100**	**-4293**	-4305	100	**100**	**-4293**	-4305	100	**100**	**-4293**	-4305	100
mp	93	-4280	-4299	99	96	-4285	-4299	99	98	-4289	-4299	100	98	-4289	-4299	100
mp-fw	99	-4292	-4306	100	**100**	**-4293**	-4299	100	**100**	**-4293**	-4296	100	**100**	**-4293**	-4295	100
mp-mcf	90	-4245	-4303	98	93	-4264	-4298	99	95	-4274	-4296	99	97	-4277	-4295	99

opt: optimally solved instances (%); **E:** average best objective value; **D:** average best lower bound if applicable;
acc: average accuracy corresponding to best objective (%); **—*:** method yields no solution for at least one problem instance

Table A7: Detailed fixed-time evaluation for dataset hotel.

	house-dense (1s)				house-dense (10s)				house-dense (100s)				house-dense (300s)			
	opt	E	D	acc	opt	E	D	acc	opt	E	D	acc	opt	E	D	acc
fgmd	0		—*		0		—*		77	-3542	–	89	77	-3542	–	89
fm	**100**	**-3778**	-4078	100	**100**	**-3778**	-4078	100	**100**	**-3778**	-4078	100	**100**	**-3778**	-4078	100
fw	**100**	**-3778**	–	100	**100**	**-3778**	–	100	**100**	**-3778**	–	100	**100**	**-3778**	–	100
ga	0	3491	–	8	0	3491	–	8	0	3491	–	8	0	3491	–	8
ipfps	0	3654	–	8	0	3654	–	8	0	3654	–	8	0	3654	–	8
ipfpu	0	3659	–	8	0	3659	–	8	0	3659	–	8	0	3659	–	8
lsm	0		—*		0	2391	–	18	0	2391	–	18	0	2391	–	18
mpm	0	1260	–	53	0	1260	–	53	0	1260	–	53	0	1260	–	53
pm	0	3262	–	18	0	3262	–	18	0	3262	–	18	0	3262	–	18
rrwm	0	2895	–	10	0	2895	–	10	0	2895	–	10	0	2895	–	10
sm	0	3789	–	9	0	3789	–	9	0	3789	–	9	0	3789	–	9
smac	0	2817	–	31	0	2817	–	31	0	2817	–	31	0	2817	–	31
dd-ls0	**100**	**-3778**	-3779	100	**100**	**-3778**	-3779	100	**100**	**-3778**	-3779	100	**100**	**-3778**	-3779	100
dd-ls3	**100**	**-3778**	**-3779**	100	**100**	**-3778**	**-3778**	100	**100**	**-3778**	**-3778**	100	**100**	**-3778**	**-3778**	100
dd-ls4	92	-3763	-3795	99	**100**	**-3778**	**-3778**	100	**100**	**-3778**	**-3778**	100	**100**	**-3778**	**-3778**	100
fm-bca	**100**	**-3778**	**-3779**	100	**100**	**-3778**	**-3778**	100	**100**	**-3778**	**-3778**	100	**100**	**-3778**	**-3778**	100
hbp	98		—*		**100**	**-3778**	-3806	100	**100**	**-3778**	-3806	100	**100**	**-3778**	-3806	100
mp	99	-3777	-3782	100	99	-3777	-3780	100	99	-3777	-3780	100	99	-3777	-3780	100
mp-fw	**100**	**-3778**	-3843	100	**100**	**-3778**	-3791	100	**100**	**-3778**	-3779	100	**100**	**-3778**	**-3778**	100
mp-mcf	31	-3542	-3825	89	85	-3732	-3783	98	99	-3776	**-3778**	100	**100**	**-3778**	**-3778**	100

opt: optimally solved instances (%); **E:** average best objective value; **D:** average best lower bound if applicable;
acc: average accuracy corresponding to best objective (%); **—*:** method yields no solution for at least one problem instance

Table A8: Detailed fixed-time evaluation for dataset house-dense.

	house-sparse (1s)				house-sparse (10s)				house-sparse (100s)				house-sparse (300s)			
	opt	E	D	acc	opt	E	D	acc	opt	E	D	acc	opt	E	D	acc
fgmd	0		—*		**100**	**-67**	–	100	**100**	**-67**	–	100	**100**	**-67**	–	100
fm	**100**	**-67**	-79	100	**100**	**-67**	-79	100	**100**	**-67**	-79	100	**100**	**-67**	-79	100
fw	0	0	–	0	0	0	–	0	0	0	–	0	0	0	–	0
ga	**100**	**-67**	–	100	**100**	**-67**	–	100	**100**	**-67**	–	100	**100**	**-67**	–	100
ipfps	**100**	**-67**	–	100	**100**	**-67**	–	100	**100**	**-67**	–	100	**100**	**-67**	–	100
ipfpu	**100**	**-67**	–	100	**100**	**-67**	–	100	**100**	**-67**	–	100	**100**	**-67**	–	100
lsm	46	-65	–	96	46	-65	–	96	46	-65	–	96	46	-65	–	96
mpm	0	-60	–	90	0	-60	–	90	0	-60	–	90	0	-60	–	90
pm	0	-54	–	83	0	-54	–	83	0	-54	–	83	0	-54	–	83
rrwm	**100**	**-67**	–	100	**100**	**-67**	–	100	**100**	**-67**	–	100	**100**	**-67**	–	100
sm	96	**-67**	–	100	96	**-67**	–	100	96	**-67**	–	100	96	**-67**	–	100
smac	37	-44	–	63	37	-44	–	63	37	-44	–	63	37	-44	–	63
dd-ls0	**100**	**-67**	**-67**	100	**100**	**-67**	**-67**	100	**100**	**-67**	**-67**	100	**100**	**-67**	**-67**	100
dd-ls3	96	**-66**	**-67**	100	**100**	**-67**	**-67**	100	**100**	**-67**	**-67**	100	**100**	**-67**	**-67**	100
dd-ls4	18	-56	-72	86	**100**	**-67**	**-67**	100	**100**	**-67**	**-67**	100	**100**	**-67**	**-67**	100
fm-bca	**100**	**-67**	**-67**	100	**100**	**-67**	**-67**	100	**100**	**-67**	**-67**	100	**100**	**-67**	**-67**	100
hbp	**100**	**-67**	**-67**	100	**100**	**-67**	**-67**	100	**100**	**-67**	**-67**	100	**100**	**-67**	**-67**	100
mp	**100**	**-67**	**-67**	100	**100**	**-67**	**-67**	100	**100**	**-67**	**-67**	100	**100**	**-67**	**-67**	100
mp-fw	**100**	**-67**	**-67**	100	**100**	**-67**	**-67**	100	**100**	**-67**	**-67**	100	**100**	**-67**	**-67**	100
mp-mcf	**100**	**-67**	**-67**	100	**100**	**-67**	**-67**	100	**100**	**-67**	**-67**	100	**100**	**-67**	**-67**	100

opt: optimally solved instances (%); **E:** average best objective value; **D:** average best lower bound if applicable;
acc: average accuracy corresponding to best objective (%); **—*:** method yields no solution for at least one problem instance

Table A9: Detailed fixed-time evaluation for dataset house-sparse.

	motor (1s)				motor (10s)				motor (100s)				motor (300s)			
	opt	E	D	acc	opt	E	D	acc	opt	E	D	acc	opt	E	D	acc
fgmd	0		—*		85	**-63**	–	97	85	**-63**	–	97	85	**-63**	–	97
fm	90	**-63**	-94	93	**100**	**-63**	-94	97	**100**	**-63**	-94	97	**100**	**-63**	-94	97
fw	20	-58	–	70	20	-58	–	70	20	-58	–	70	20	-58	–	70
ga	55	-62	–	90	55	-62	–	90	55	-62	–	90	55	-62	–	90
ipfps	25	-61	–	85	25	-61	–	85	25	-61	–	85	25	-61	–	85
ipfpu	15	-58	–	77	15	-58	–	77	15	-58	–	77	15	-58	–	77
lsm	10	-52	–	64	10	-52	–	64	10	-52	–	64	10	-52	–	64
mpm	5		—*		5	-50	–	56	5	-50	–	56	5	-50	–	56
pm	0	-35	–	32	0	-35	–	32	0	-35	–	32	0	-35	–	32
rrwm	50	-62	–	89	50	-62	–	89	50	-62	–	89	50	-62	–	89
sm	40	-60	–	87	40	-60	–	87	40	-60	–	87	40	-60	–	87
smac	10	-52	–	66	10	-52	–	66	10	-52	–	66	10	-52	–	66
dd-ls0	**100**	**-63**	**-63**	97	**100**	**-63**	**-63**	97	**100**	**-63**	**-63**	97	**100**	**-63**	**-63**	97
dd-ls3	65	-57	-65	87	**100**	**-63**	**-63**	97	**100**	**-63**	**-63**	97	**100**	**-63**	**-63**	97
dd-ls4	30	-52	-68	78	70	-60	-64	93	**100**	**-63**	**-63**	97	**100**	**-63**	**-63**	97
fm-bca	**100**	**-63**	**-63**	97	**100**	**-63**	**-63**	97	**100**	**-63**	**-63**	97	**100**	**-63**	**-63**	97
hbp	95		—*		**100**	**-63**	-65	97	**100**	**-63**	-65	97	**100**	**-63**	-65	97
mp	90	**-63**	**-63**	96	95	**-63**	**-63**	98	95	**-63**	**-63**	98	95	**-63**	**-63**	98
mp-fw	95	**-63**	**-63**	98	**100**	**-63**	**-63**	97	**100**	**-63**	**-63**	97	**100**	**-63**	**-63**	97
mp-mcf	90	**-63**	**-63**	98	95	**-63**	**-63**	99	**100**	**-63**	**-63**	97	**100**	**-63**	**-63**	97

opt: optimally solved instances (%); **E:** average best objective value; **D:** average best lower bound if applicable;
acc: average accuracy corresponding to best objective (%); **—*:** method yields no solution for at least one problem instance

Table A10: Detailed fixed-time evaluation for dataset motor.

	opengm (1s)			opengm (10s)			opengm (100s)			opengm (300s)		
	opt	E	D	opt	E	D	opt	E	D	opt	E	D
fgmd	0	—*		50	—*		75	-166	–	75	-166	–
fm	**100**	**-171**	-192	**100**	**-171**	-192	**100**	**-171**	-192	**100**	**-171**	-192
fw	0	-152	–	0	-152	–	0	-152	–	0	-152	–
ga	50	-167	–	50	-167	–	50	-167	–	50	-167	–
ipfps	0	-95	–	0	-95	–	0	-95	–	0	-95	–
ipfpu	0	-86	–	0	-86	–	0	-86	–	0	-86	–
lsm	0	-67	–	0	-67	–	0	-67	–	0	-67	–
mpm	0	-94	–	0	-94	–	0	-94	–	0	-94	–
pm	0	-83	–	0	-83	–	0	-83	–	0	-83	–
rrwm	0	-154	–	0	-154	–	0	-154	–	0	-154	–
sm	0	-101	–	0	-101	–	0	-101	–	0	-101	–
smac	0	-84	–	0	-84	–	0	-84	–	0	-84	–
dd-ls0	50	-160	**-172**	75	-161	**-171**	75	-161	**-171**	75	-161	**-171**
dd-ls3	0	-118	-185	**100**	**-171**	**-171**	**100**	**-171**	**-171**	**100**	**-171**	**-171**
dd-ls4	0	-105	-214	25	-141	-178	**100**	**-171**	**-171**	**100**	**-171**	**-171**
fm-bca	75	-170	-177	**100**	**-171**	-177	**100**	**-171**	-177	**100**	**-171**	-177
hbp	0	—*		50	-164	-185	50	-164	-185	50	-164	-185
mp	0	-57	-192	0	-57	-192	0	-57	-192	0	-57	-192
mp-fw	0	-150	-192	0	-150	-192	0	-150	-192	0	-150	-192
mp-mcf	0	-57	-192	0	-57	-192	0	-57	-192	0	-57	-192

opt: optimally solved instances (%); **E:** average best objective value; **D:** average best lower bound if applicable;
acc: average accuracy corresponding to best objective (%); **—*:** method yields no solution for at least one problem instance

Table A11: Detailed fixed-time evaluation for dataset opengm.

	pairs (1s)			pairs (10s)			pairs (100s)			pairs (300s)		
	E	D	acc	E	D	acc	E	D	acc	E	D	acc
fgmd		—*			—*			—*			—*	
fm	-64812	-76418	51	-65625	-76418	54	-65825	-76418	55	-65870	-76418	56
fw	**-65722**	–	54	**-65797**	–	54	-65802	–	54	-65802	–	54
ga		—*			—*			—*			—*	
ipfps		—*			—*		-35115	–	7	-35115	–	7
ipfpu		—*			—*		-35666	–	7	-35666	–	7
lsm		—*			—*			—*			—*	
mpm		—*			—*			—*			—*	
pm		—*			—*			—*			—*	
rrwm		—*			—*			—*			—*	
sm		—*			—*			—*		-196	–	0
smac		—*			—*			—*			—*	
dd-ls0	-61482	-76497	41	-61482	-73521	41	-62974	**-67306**	57	-63454	-67114	60
dd-ls3	-61638	**-75656**	41	-61638	-73528	41	-62426	-67599	50	-64556	**-66704**	60
dd-ls4	-61634	-75852	41	-61634	-74053	41	-61634	-70214	41	-62894	-67352	53
fm-bca		—*		-65567	-70163	55	**-65913**	-69003	58	**-65958**	-68909	58
hbp		—*			—*			—*			—*	
mp		—*		-64150	**-68255**	57	-64380	-68136	57	-64418	-68130	57
mp-fw		—*			—*			—*		-65794	-69623	55
mp-mcf		—*		-63990	-68318	56	-64174	-68053	57	-64208	-67748	57

opt: optimally solved instances (%); **E:** average best objective value; **D:** average best lower bound if applicable;
acc: average accuracy corresponding to best objective (%); **—*:** method yields no solution for at least one problem instance

Table A12: Detailed fixed-time evaluation for dataset pairs.

	worms (1s)				worms (10s)				worms (100s)				worms (300s)			
	opt	E	D	acc	opt	E	D	acc	opt	E	D	acc	opt	E	D	acc
fgmd	0		—*		0		—*		0		—*		0		—*	
fm	**93**	-48457	-55757	89	**93**	-48458	-55757	89	**93**	-48461	-55757	89	**93**	-48461	-55757	89
fw	0	-46974	–	81	3	-48032	–	85	3	-48038	–	85	3	-48038	–	85
ga	0		—*		0		—*		0		—*		0		—*	
ipfps	0		—*		0		—*		0	-1147	–	1	0	-1147	–	1
ipfpu	0		—*		0		—*		0	0	–	0	0	0	–	0
lsm	0		—*		0		—*		0		—*		0		—*	
mpm	0		—*		0		—*		0		—*		0		—*	
pm	0		—*		0		—*		0		—*		0		—*	
rrwm	0		—*		0		—*		0		—*		0		—*	
sm	0		—*		0		—*		0	-6453	–	6	0	-6453	–	6
smac	0		—*		0		—*		0		—*		0		—*	
dd-ls0	0	60443	-163870	26	0	50517	-148830	25	0	-3982	-58449	58	0	-43824	-48682	87
dd-ls3	0	64017	-160520	24	0	49257	-144842	24	0	11744	-71523	46	0	-40882	-48943	85
dd-ls4	0	65731	-160409	24	0	58300	-153566	25	0	31066	-109434	33	0	12795	-75088	44
fm-bca	**93**	**-48460**	**-48514**	89	**93**	**-48464**	**-48498**	89	**93**	**-48464**	-48498	89	**93**	**-48464**	-48498	89
hbp	0		—*		0		—*		0		—*		0		—*	
mp	0		—*		67	-48391	**-48498**	89	70	-48393	**-48497**	89	70	-48393	**-48496**	89
mp-fw	0		—*		3		—*		57	-48402	-48759	88	83	-48435	-48631	88
mp-mcf	0		—*		3	-47942	-48588	88	3	-48027	-48558	88	3	-48057	-48552	88

opt: optimally solved instances (%); **E:** average best objective value; **D:** average best lower bound if applicable;
acc: average accuracy corresponding to best objective (%); **—*:** method yields no solution for at least one problem instance

Table A13: Detailed fixed-time evaluation for dataset worms.