Automatically selecting inference algorithms for discrete energy minimisation

Paul Henderson & Vittorio Ferrari

School of Informatics, University of Edinburgh
{p.m.henderson,vittorio.ferrari}@ed.ac.uk

Abstract. Minimisation of discrete energies defined over factors is an important problem in computer vision, and a vast number of MAP inference algorithms have been proposed. Different inference algorithms perform better on factor graph models (GMs) from different underlying problem classes, and in general it is difficult to know which algorithm will yield the lowest energy for a given GM. To mitigate this difficulty, survey papers [1–3] advise the practitioner on what algorithms perform well on what classes of models. We take the next step forward, and present a technique to automatically select the best inference algorithm for an input GM. We validate our method experimentally on an extended version of the OpenGM2 benchmark [3], containing a diverse set of vision problems. On average, our method selects an inference algorithm yielding labellings with 96% of variables the same as the best available algorithm.

1 Introduction

Minimisation of discrete energies defined over factors is an important problem in computer vision and other fields such as bioinformatics, with many algorithms proposed in the literature to solve such problems [3]. These models arise from many different underlying *problem classes*; in vision, typical examples are stereo matching, semantic segmentation, and texture reconstruction, each of which yields models with very different characteristics, making different choices of minimisation algorithm preferable.

We consider factor graph models (GMs) defined by sets V and F of variables and factors respectively. Each variable takes values in some discrete label-space, and each factor is a real-valued function on some subset of V, its clique, yielding an additive contribution to a global energy. In this paper, we focus on algorithms to find the labelling of variables that minimises this global energy, the so-called *MAP inference* problem. Different problem classes give rise to problem instances with different characteristics, such as size of cliques and number of variables, affecting which inference algorithms are best suited to them.

The space of published inference algorithms is vast, with methods ranging from highly specialised to very general. For example, message passing [4] is widely applicable, but takes exponential time for large cliques, and may not converge. Dual-space variants such as TRW-S [5] do guarantee convergence, but not necessarily to a global optimum. α-expansion [6] and graph-cuts [7] are better suited

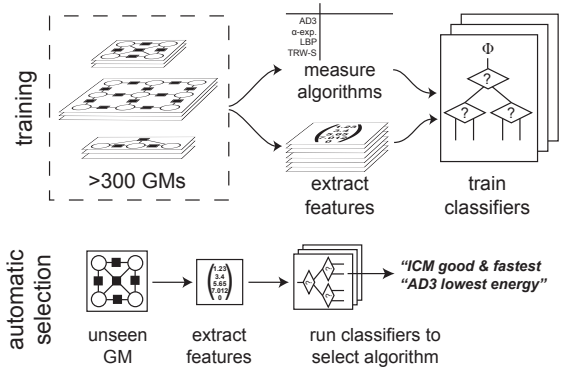


Fig. 1: Our pipeline for automatic algorithm selection

to models with dense connectivity, but require factors to take certain restricted forms, while QPBO [8] only works for binary models and may leave some variables unlabelled. Algorithms solving the Wolfe dual [9–11] such as [12, 13] are applicable to models with arbitrary factors and labels, but existing implementations for this generic setting tend to run more slowly.

Thus, when developing a new model, it may be difficult to decide what algorithm to use for inference. Selecting a good algorithm for a given model requires extensive expertise about the landscape of existing algorithms and typically involves understanding the operational details of many of them. Moreover, even for an expert who can choose which algorithm is best overall on a particular problem class, it may not be clear which is best for a particular *instance*—certain problem classes are heterogeneous enough that different instances within them may be best solved by different algorithms (sec. 3). An alternative solution would be to run many algorithms on each input model and see which one performs best empirically. However, this would be computationally very expensive.

Recently studies appeared that evaluate a number of algorithms on various problems, comparing their performance [3, 2, 14, 1, 15]. These are intended to provide a 'field guide' for the practitioner, suggesting which techniques are suited for which models. In this paper, we take the next step forward and propose a technique to *automatically* select which inference algorithm to run on an input problem instance (sec. 4). We do so without requiring the user to have any knowledge of the applicability of different inference methods, and without the computational expense of running many algorithms. Thus, our method is particularly suited for the vision practitioner with limited knowledge of inference, but who wishes to apply it to real-world problems.

Our method uses features extracted from the problem instance itself, to select inference algorithms according to two criteria relevant for the practitioner: (1) the fastest algorithm reaching the lowest energy for that instance; or (2) the fastest algorithm delivering a very similar labelling to the lowest energy one (fig. 1). The features are designed to capture characteristics of the instance

that affect algorithm applicability or performance, such as the clique sizes and connectivity structure (sec. 4.1). We train our selection models without human supervision, based on the results of running many algorithms over a large dataset of training problem instances.

We perform experiments (sec. 5) on an extended version of the OpenGM2 benchmark [3], containing 344 problem instances drawn from 32 diverse classes (sec. 2), and consider a pool of 15 inference algorithms drawn from the most prominent approaches (sec. 3). The results show that on 69% of problem instances our method selects the best algorithm. On average, the labels of 96% of variables match that returned by the algorithm achieving the lowest energy. Our automatic selector achieves these results over 88× faster than the obvious alternative of running all algorithms and retaining the best solution.

1.1 Related work

MAP inference. MAP inference algorithms can be split into several broad categories. Graph-cuts [7] is very efficient, but restricted to pairwise binary GMs with submodular factors. It can be extended to more general models, such as by the move-making methods α-expansion and $\alpha\beta$-swap [6], wherein a subset of variables change label at each iteration, or by transformations introducing auxiliary variables [16–18]. Alternatively, inference is naturally formulated as an integer linear program, which can be solved directly and optimally using off-the-shelf polyhedral solvers for small problems [3]. It can also be relaxed to a non-integer linear program (LP), which can be solved faster. However, it requires rounding the solution, which does not always yield the global optimum of the original problem. Message-passing algorithms [4, 19] have each variable/factor iteratively send to its neighbours messages encoding its current belief about each neighbour's min-marginals. Tree-reweighted methods [5, 20] use a message-passing formulation, but actually solve a Lagrangian dual of the LP, and can provide a certificate of optimality where relevant. Other dual-decomposition methods [12, 13, 21] directly solve the Wolfe dual [10, 11] to the LP, but by iteratively finding the MAP state of each clique (or other tractable subgraphs) instead of passing messages. Our focus in this paper is not to introduce another inference algorithm, but to consider the meta-problem of learning to select what existing inference algorithm to apply to an input model; as such, we use many of the above algorithms in our framework (sec. 3).

Inferning. Our work is a form of *inferning* [22], as it considers interactions between inference and learning. A few such methods use learning to guide the inference process. Unlike the hard-wired algorithms mentioned above, these approaches learn to adapt to the characteristics of a particular problem class. Some operate by pruning the model during inference, by learning classifiers to remove labels from some variables [23, 24], or to remove certain factors from the model [25, 26]. Others learn an optimal sequence of operations to perform during message-passing inference [27]. Our work operates at a higher level than these approaches. Instead of incorporating learning into an algorithm to allow

adaptation to a problem class, we instead learn to predict which of a fixed set of hard-wired algorithms is best to apply to a given problem instance.

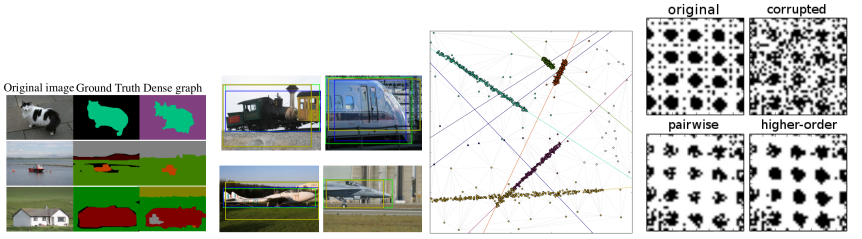
Surveys on inference. The survey papers [1, 2, 14, 15, 3] evaluate a number of algorithms on various problems, comparing their performance. [1] focuses on stereo matching and considers highly-connected grid models defined on pixels with unary and pairwise factors only. It evaluates three inference algorithms (graph-cuts, TRW-S, and belief propagation). [2] considers a wider selection of problems—stereo matching, image reconstruction, photomontaging, and binary segmentation—but with 4-connectivity only, and applies a wider range of algorithms, adding ICM and α-expansion to the above. Recently, [14, 3] substantially widened the scope of such analysis, by considering also models with higher-order potentials, regular graphs with denser connectivity, models based on superpixels with smaller number of variables, and partitioning problems without unary terms. They compare the performance of many different types of algorithms on these models, including some specialised to particular problem classes. These surveys help to understand the space of existing algorithms and provide a guide to which algorithms are suited for which models. Our work takes a natural step forward, with a technique to automatically select the best algorithm to run on an input problem instance.

Automatic algorithm selection. Automatic algorithm selection was pioneered by [28], which considered algorithms for quadrature and process scheduling. More recently, machine learning techniques have been used to select algorithms for constraint-satisfaction [29], and other combinatorial search problems [30]. However, none of these works consider selecting MAP inference algorithms.

2 Dataset of models

OpenGM2 [3]. The OpenGM2 dataset contains GMs drawn from 28 problem classes, including pairwise and higher-order models from computer vision and bioinformatics; it is the largest dataset of GMs currently available. We briefly summarize here the main kinds of problems and refer to [3] for details.

- low-level vision problems such as stereo matching [2], inpainting [31, 32], and montaging [2]. These are all locally-connected graphs with variables corresponding to pixels, and with pairwise factors only; label counts vary widely between classes, from 2–256.
- small semantic segmentation problems with up to eight classes, with labels corresponding to surface types [33] and geometric descriptions [34]. These are irregular, sparse graphs over superpixels; [33] uses pairwise factors only, while [34] has general third-order terms.
- partitioning (unsupervised segmentation by clustering) based on patch similarity, operating on superpixels and with as many labels as variables, in both 2D [35–37] and 3D [38]. Potts or generalised Potts factors are used in all cases; [35] has very large cliques with up to 300 variables, while the other classes are pairwise or third-order, just one class having dense connectivity.



msrc-segmentation	**joint-localisation**	**line-fitting**	**texture-restoration**
$\|V\| = 275$	$\|V\| = 81$	$\|V\| = 661$	$\|V\| = 1900$
$L = 21$	$L = 100$	$L = 51$	$L = 2$
$O = 2$	$O = 2$	$O = 2$	$O = 49$

Fig. 2: Our four added GM classes. Variable count $|V|$ is a mean over all instances; L is mean label count, and order O is largest factor clique size

– two problem classes from bioinformatics: protein side-chain prediction [39], and protein folding [40]; both are defined over irregular graphs, with [39] having only two labels but general third-order factors, while [40] has up to 503 labels and dense pairwise connectivity.

Below we complement the OpenGM2 dataset with four additional, interesting problem classes which arise in modern computer vision applications (fig. 2).

Semantic segmentation with context [23]. Semantic segmentation on the MSRC-21 dataset [41] with relative position factors. Variables correspond to superpixels and labels to 21 object/background classes (*e.g.* car, road, sky). Unary factors are given by appearance classifiers on features of a superpixel, while pairwise factors encode relative location in the image, to favour labellings showing classes in the expected spatial relation to one another (*e.g.* sky above road). The model is fully connected, *i.e.* there is a pairwise factor between every two superpixels in the image.

Joint localisation [23]. Joint object localisation across images on the PASCAL VOC 2007 dataset [42]. The set of images containing a certain object class form a problem instance. Variables correspond to images and labels to object proposals [43] in the images. Unary factors are given by the objectness probability of a proposal [43], while pairwise factors measure the appearance similarity between two proposals in different images. Inference on this model will select one proposal per image, so that they are likely to contain objects and to be visually similar over the images.

Line fitting [44]. Fitting of multiple lines to a set of points in $\mathbb{R}^2$. This an alternative to RANSAC [45] for fitting an unknown number of geometric models to a dataset. Variables correspond to points and labels to candidate lines from a fixed pool (sampled from the point set in a preprocessing stage). Unary factors favour labelling a point with a nearby line, while pairwise factors promote local smoothness of the labelling (*i.e.* nearby points should take the same label).

Texture restoration [46]. Binary texture restoration with pattern potentials. Given a binary image corrupted by noise, the task is to reconstruct the original noise-free image, while preserving the underlying texture regularity. Variables correspond to pixels and labels to 'on' or 'off'. Unary factors penalise deviations from the input noisy image, while pairwise factors prefer pixels at certain offsets taking certain pairs of values (learned on a training image showing a noise-free texture). Higher-order factors reward image patches for taking joint labellings which occur frequently in the training image (patterns). The pairwise and higher-order factors capture low and high order texture properties, respectively.

Data diversity. From each problem class we take all instances up to a maximum of 20. This results in a diverse dataset of 344 problem instances drawn from the 32 classes; 224 of these instances are pairwise and 120 higher-order. 21 of the problem classes have small label-spaces (< 20 labels), while the remainder vary greatly up to a maximum of 17074. Variable counts similarly cover a wide range, from 19 to 2356620, with a median of 10148. Amongst the higher-order problems, 58% of instances have arbitrary dense factor tables, while the remainder have Potts potentials [6] or generalised versions thereof [47, 48]. The problem classes also differ greatly in the degrees of homogeneity of their instances. For example, instances in the *line-fitting* class vary by an order of magnitude in variable and label counts, whereas all instances in the *inclusion* class have identical characteristics but for the factor energies themselves.

3 Inference algorithms and performance

Inference algorithms. A vast number of MAP inference algorithms have been proposed in the literature, with differing approaches, degrees of generality, and performance characteristics. We selected 15 to use in our experiments (table 1), including representative algorithms from most prominent approaches, *e.g.* move-making, message-passing, dual-decomposition, combinatorial, etc. This covers many of the most commonly used algorithms in computer vision, such as TRW-S [5], QPBO [8], and α-expansion [6]. Note however that we do not aim to form an exhaustive pool of all good algorithms; our automated selection method is agnostic to the pool of algorithms it is trained to select from, and explicitly avoids making prior assumptions on their applicability.

We also include a simple method, dubbed *unary-modes (UM)*, which labels each variable by minimizing its unary factors only; this should perform poorly on genuinely hard structured prediction problems, where the non-unary factors have a decisive impact on the MAP labelling.

Protocol for inference. We used the original authors' implementation of each algorithm where available, and the implementations in [3] otherwise. Every algorithm was run on every problem instance in our dataset, with limits of 60 minutes CPU time and 4GB RAM imposed for inference on one instance. For each successful run, we recorded the MAP labelling and time taken.

Many of the algorithms have free parameters that must be defined by the user. While it was not practical to evaluate every possible combination of parameters,

Table 1: Algorithms used in this study, including the GM orders they are applicable to (pw = pairwise), number of parameter settings included if more than one (#p), full name or description, and reference to the original work

alias	order	#p	name / description	ref.
A*	*all*		implicitly convert to shortest-path problem and apply A*	[49]
AD^3	*all*		alternating directions dual decomposition with branch and bound	[13]
α-exp	*pw*		alpha-expansion	[6]
BPS	*all*	4	sequential loopy belief propagation, implementation of [3]	[4]
DDS	*all*	2	dual decomposition with subgradient descent	[12]
FPD	*pw*	3	fast primal/dual (FastPD)	[50]
ICM	*all*		iterated conditional modes	[51]
ILP	*all*		solve as integer programming problem with Gurobi	[3]
KL	*pw*		Kernighan-Lin method for 2$^{\text{nd}}$ order partitioning problems	[52]
LBP	*all*	4	parallel loopy belief propagation, implementation of [3]	[4]
LP	*all*		solve linear programming relaxation with Gurobi	[3]
MPLP	*all*		max-product linear programming with cutting plane relaxation tightening	[21, 53]
QPBO	*pw*		quadratic pseudo-boolean optimisation	[8]
TRW-S	*pw*	3	sequential tree-reweighted message-passing	[5]
UM	*all*		take lowest-energy label according to unary factors only	-

for several of the algorithms we included multiple parameterisations where this affects their results significantly. For example, we ran four versions of loopy belief propagation, with damping set to 0.0 and 0.75, and maximum iteration counts of 50 and 250. In such cases, the different parameterisations are combined to create a meta-algorithm, which simulates the user running every parameterisation, then taking the results from that yielding lowest energy on the problem instance.

Several incompatible combinations of algorithms and GMs were included. When possible, we still ran the algorithm to obtain an approximate solution:

- higher-order factors are omitted when passing GMs to pairwise algorithms. However, when evaluating the algorithm's performance, the energy of the output labelling is still computed on the full model including all factors.
- non-metric pairwise factors passed to α-expansion are handled as if they were metric, sacrificing the usual correctness and optimality guarantees [6].

When it was not possible to run the algorithm, we counted this as a failure:

- QPBO aborts when presented with a GM having non-binary variables.
- FastPD aborts when presented with a GM whose pairwise factors are not all proportional to some uniform distance function on labels.
- Kernighan-Lin aborts when presented with a GM having factors that are not pairwise Potts

Performance measures. We measured three aspects of the performance of each algorithm:

- *completes:* whether the algorithm runs to completion, *i.e.* returns a solution within 60 minutes, regardless of the energy of that solution.

Table 2: Aggregate performance of each inference algorithm on our dataset; mean time is over instances for which the algorithm successfully returns a result

	% instances for which...			mean time /s
	completes	*best-&-fastest*	*good-&-fastest*	
A*	4	0	0	0.1
AD^3	52	7	1	390.2
α-exp	98	5	7	23.4
BPS	72	4	2	158.3
DDS	80	0	0	296.6
FPD	31	9	22	7.2
ILP	48	1	0	96.3
LP	52	2	1	76.8
ICM	100	30	31	60.7
KL	12	10	10	142.2
LBP	73	6	4	193.5
MPLP	56	1	1	1116.3
QPBO	12	0	2	0.1
TRW-S	94	19	10	236.4
UM	100	0	3	0.1

- *best-and-fastest:* whether the algorithm reaches the lowest energy among all algorithms, faster than any other one that does so. This is relevant for a user requiring the solution with lowest possible energy, even at high computational cost.
- *good-and-fastest:* whether the algorithm is the fastest to reach a solution with 98% of variables matching the lowest energy labelling. This is highly relevant in practice, as minor deviations from that labelling may not matter to the user, while achieving it would require a significantly slower algorithm.

Table 2 shows the performance of the algorithms with respect to these measures.

Algorithm diversity. We see that the distributions of both best-and-fastest and good-and-fastest algorithms over instances have high entropy—many different algorithms are best-and-fastest or good-and-fastest for a significant fraction of GMs. 11 of the 15 algorithms are able to return a solution for at least one instance on more than half of the problem classes; the other four are particularly restricted, such as QPBO (which only operates on binary problems). All the algorithms other than A* and DDS are the best-and-fastest for at least one problem instance. TRW-S and FastPD perform particularly well on pairwise problems, with TRW-S generally reaching slightly lower energies, but FastPD being much quicker. Kernighan-Lin outperforms all algorithms on pairwise partitioning problems. AD^3 gives low energies for high-order problems, but often takes longer than other algorithms. Only ICM and unary-modes are able to return a solution for all problem instances. Although they are fast and widely-applicable, these naïve methods are unable to return the best solution in the majority of cases. All these observations show how our goal of learning to select

the best inferencer is much harder than simply picking any algorithm that runs to completion.

4 Learning to select an algorithm

We now consider how to automatically select the best MAP inference algorithm for an input problem instance. This is the main contribution of this paper. We define two tasks: (1) predicting the best-and-fastest algorithm; and (2) predicting the good-and-fastest algorithm. To address these tasks, we design selection models that take a GM as input, and select an algorithm as output (sec. 4.2). The selection models operate on features extracted from the GMs themselves (sec. 4.1). This is different from the typical approach in computer vision of extracting features from images and using these to build a GM.

4.1 GM features

We extract the following three groups of features from each problem instance (fig. 3).

Instance size. The number of variables, $|V|$, and of factors, $|F|$, are used to indicate the overall size of the problem instance, hence whether slower algorithms are likely to be applicable. We also include the minimum, maximum and mean label count over all variables. See fig. 3b.

Structural features. We extract more sophisticated features based on the model structure, *i.e.* which do not depend on the factor values themselves. Firstly, we take a histogram and statistics (minimum, maximum, mean) of both:

- variable orders (*i.e.* for each variable, number of connected factors, fig. 3c)
- factor orders (*i.e.* for each factor, number of variables in its clique, fig. 3c)

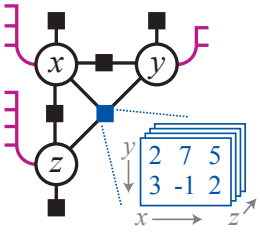
Secondly, we measure factor densities—for each factor order $M \geq 2$, the number of factors of order M divided by the binomial coefficient $\binom{|V|}{M} = \frac{|V|!}{M!(|V|-M)!}$. Intuitively, this is the number of possible M-cliques that actually have an associated factor. In fig. 3, this is 1 for third order, as there is only one possible triplet, but 2/3 for second order, as only two of the possible three pairs of variables have a pairwise factor: (x, y) and (x, z) but not (z, y)

Energy features. Our final group of features depend on the values of the factors themselves, *e.g.* the blue values in fig. 3a. To determine the influence of different orders of factors, we compute means and deviations over values they take, defined as follows:

- for each factor $f \in F$, let μ_f be the mean and σ_f the standard deviation of all unique values taken by f
- then, for each factor order $M \geq 2$, compute for factors F_M of that order, the ratio of each of the following to the same quantity for $M = 1$:

  (i) $\sum_{f \in F_M} \mu_f$ (ii) $\sum_{f \in F_M} \mu_f / |F_M|$ (iii) $\sum_{f \in F_M} \sigma_f / |F_M|$.

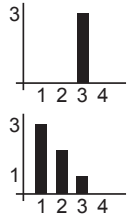
Intuitively, these capture how much influence each order of factor has on the final energy, *i.e.* how much changing the labelling will change the values of factors



(a)

$|V| = 3$
$|F| = 6$
$L_{min} = 2$
$L_{max} = 4$
$L_{mean} = 3$

(b) size



(c) variable (top) and factor (bottom) orders

order	density
2	$\frac{2}{\binom{3}{2}} = \frac{2}{3}$
3	$\frac{1}{\binom{3}{3}} = 1$
4	0

(d) factor densities

Fig. 3: Example GM structure (a) and associated features (b-e). Circles correspond to variables, and squares to factors; the label space of each variable is shown as purple dashes. Part of the value table for the third-order factor (blue) is also shown

of each order. On pairwise GMs, a large influence of the pairwise (as opposed to unary) factors makes inference harder; on higher-order GMs, pairwise algorithms should perform relatively well only when the influence of higher-order factors is small. We also count the fraction of pairwise factors f having each of the following characteristics for all labels a, b, c:

(i) $f(a,b) = 0$ iff $a = b$ (ii) $f(a,b) \geq 0$

(iii) $f(a,b) = f(b,a)$ (iv) $f(a,b) + f(b,c) \geq f(a,c)$.

Together, these are the conditions for a factor to be metric; without (iv), to be semi-metric—respectively requirements for the α-expansion and $\alpha\beta$-swap algorithms to fulfill their correctness guarantees [6]. Finally, we measure the fraction of pairwise factors which are submodular; in general pairwise submodular problems are easier to solve by LP-based methods, as their LP relaxation is tight.

4.2 Algorithm selection models and their training

Selection models. We propose two algorithm selection models. Each is a 1-of-N classifier implemented as a random forest [54], taking the features described in sec. 4.1 as input. Model *BF* is trained to predict the best-and-fastest algorithm; model *GF* is trained to predict the good-and-fastest algorithm. The random forests are trained recursively by selecting the best split from a randomly-generated pool at each step, using information gain (*i.e.* entropy decrease) as the criterion, and with outputs modelled by categorical distributions [54].

Data. We train the selection models on a subset of our dataset (sec. 2). A training sample consists of features extracted from a problem instance and a target output label denoting which algorithm works best on it. It is important to note that these training labels are automatically generated by running all algorithms on the training instances, as in sec. 3. No human annotation is required. At test time, we run the selection models on a separate subset of the dataset. The evaluation compares the algorithm selected by our model to the one known to perform best (sec. 5). Again, this test label is produced automatically.

5 Experiments

Tasks and baselines. We report results on the two tasks defined in sec. 4: (i) predicting the good-and-fastest inference algorithm for an input GM; and (ii) predicting the best-and-fastest algorithm. Task (i) is addressed by the selection model GF, and task (ii) by model BF (sec. 4.2). For both tasks, we also analyse the performance of two baseline methods that select an algorithm without looking at features of the input instance. The first baseline *NB* always selects the algorithm that is most often best over the full training set. This mimics the behaviour of a naïve user who simply chooses one commonly good algorithm to use. For the second, stronger baseline *SB*, we assign each of the problem classes to one of three superclasses:

1. pairwise—many algorithms are designed for pairwise problems only;
2. higher-order—there exist algorithms designed explicitly to handle higher-order factors, but which may be slow for pairwise instances;
3. partitioning—these are a special class which is hard for general algorithms (due to having a large label space, and being invariant to label permutations) but certain methods can exploit this structure to solve them efficiently; most partitioning problems in our dataset are pairwise, but some are third-order.

Then, at test time, each problem instance is assigned the algorithm that is most often best for training problems of its superclass. This strong baseline mimics the behaviour of a user with good working knowledge of inference—enough to recognise how her problem fits in these superclasses, and to know which algorithm will be best for each.

Experimental setup. We select half the problem instances at random to train on, and the remainder are used for testing. As discussed in sec. 3, the ground-truth labels marking which inference algorithms perform best on a problem instance are obtained automatically by running all algorithms on all instances. No human annotation is necessary for training or testing.

When training and evaluating selection models, the underlying problem classes are always treated as unknown—they are not provided as input data. We want the selection models to freely learn the optimal association between GM features and good algorithms to run. The GM features we propose are designed to enable the selection models to reason upon various properties of GMs, which can be used to characterize problem classes (*e.g.* connectivity structure and distributions of energy values in the factors). So, we might expect the selection models to learn at least some of the problem class structures, given that this often correlates with the best algorithms (sec. 3).

Evaluation measures. Our algorithm selection models are evaluated on the test set with the following measures (table 3):

- percentage of instances with the correct algorithm (best-and-fastest or good-and-fastest) selected. This is the measure for which we trained our selection models.
- mean (over instances) of fraction of variables matching the labelling returned by the best-and-fastest algorithm. This is particularly relevant in practice,

Table 3: Performance of our model GF and baselines NB and SB for selecting the good-and-fastest algorithm (first three columns), and performance of our model FG and baselines for selecting the best-and-fastest algorithm (last three columns).

	good-and-fastest			*best-and-fastest*		
algorithm selected by...	**GF**	**NB**	**SB**	**BF**	**NB**	**SB**
% instances correctly classified	69	31	28	62	30	36
mean % matching variables	96.4	75.3	87.5	97.1	75.4	95.6

Table 4: Mean times and speed-ups from using our method, versus exhaustively applying all algorithms. *matching var's* is fraction of variables whose labels match true best result; *speed-up* is ratio of time to that for exhaustive testing

mean...	**time /s**	**speed-up**	**matching var's**
exhaustive	13046.8	1.0×	100%
good-and-fastest	221.3	88.1×	96.4%
best-and-fastest	312.5	46.8×	97.1%

as users typically care about the quality of the labelling output, by an algorithm, especially in terms of how close it is to the lowest-energy labelling that could have been returned.

6 Analysis

Predicting the good-and-fastest algorithm. Model GF correctly chooses the good-and-fastest algorithm for 69% of instances, with 96.4% of variables taking the same label as in the true best labelling on average. This compares favourably to the naïve baseline NB, which correctly selects only on 31% of the instances and returns labellings that are considerably worse (75.3% correctly-labelled variables on average). Indeed, our model also substantially outperforms the strong baseline, which only achieves an average of 87.5% of correctly-labelled variables.

These results show that our selection model successfully generalises to new problem instances not seen during training. It is able to select an algorithm much better than even the strong baseline of a user who knows which algorithm performs best for similar problems in the training set.

Predicting the best-and-fastest algorithm. Model BF correctly selects the best-and-fastest algorithm on 62% of instances, exceeding the naïve baseline (32%). This results in 97.1% of variables taking the same label as in the true lowest-energy solution, greatly exceeding the naïve baseline of 75.4%. Our model performs well against even the strong baseline, which only classifies 36% of instances correctly and has a slightly lower fraction of correct variables at 95.6%.

Efficiency. As noted in sec. 1, a simple alternative to our selection method is to run every algorithm on the test problem instance, and select the lowest-

Table 5: Confusion matrix showing true (rows) and predicted (columns) good-and-fastest algorithms for pairwise problems. The table only includes those algorithms that are the true good-and-fastest for at least one problem instance.

	AD^3	**α-exp**	**BPS**	**FPD**	**ICM**	**KL**	**LBP**	**QPBO**	**TRW-S**	**UM**
AD^3	0	0	0	0	2	0	0	0	0	0
α-exp	0	5	1	1	0	0	0	0	2	0
BPS	0	0	1	0	2	0	1	0	1	0
FPD	0	0	1	19	0	0	0	0	0	0
ICM	0	0	0	0	16	1	0	0	3	0
KL	0	0	0	0	0	24	0	0	0	0
LBP	0	0	0	0	3	0	4	0	0	1
QPBO	0	0	3	0	0	0	0	3	0	0
TRW-S	0	1	0	6	1	0	1	0	6	0
UM	0	1	0	1	0	0	0	1	0	0

energy solution. However, this is computationally very expensive. To evaluate the speed-up made by our method, for each problem instance we also measured (i) the total time to run every inference algorithm; (ii) the time to predict the best-and-fastest algorithm with model BF then run it; and, (iii) the time to predict the good-and-fastest algorithm with model GF then run it. As we see in table 4, our method results in an average speed-up of 46.8× using model BF, and 88.1× using model GF, with 97.1% and 96.4% of variables correctly labelled respectively. Thus, automated selection achieves labellings very similar to running every algorithm, but at a small fraction the computational expense. Model GF yields a significantly faster-running algorithm on average than model BF, with only a small drop ($< 1\%$) in variables correctly labelled.

Algorithms selected by the strong baseline. As described in sec. 5, our strong baseline chooses the algorithm that is most often best-and-fastest or good-and-fastest over the training set, for problems in the same superclass as the test instance. For predicting the best-and-fastest algorithm, Kernighan-Lin is selected for partitioning problems, TRW-S for other pairwise instances, and AD^3 for other (higher order) instances. However, for the good-and-fastest algorithm, FastPD is selected instead of TRW-S for pairwise instances, and ICM for higher order instances, indicating that these often label 98% or more of variables correctly, while being faster to run.

Algorithms selected by our method. At a coarse level, for the task of selecting the best-and-fastest algorithm, we find that our model BF most often chooses pairwise-specific algorithms for pairwise problems, and AD^3 for higher-order problems. This agrees with intuition—pairwise algorithms are specifically designed to be faster for pairwise instances, while AD^3 is a good general-purpose algorithm for higher-order instances. Interestingly, for the good-and-fastest task, model GF correctly learns to choose ICM or a good pairwise method for higher-

Table 6: Performance of algorithm selection methods selecting good-and-fastest and best-and-fastest algorithms in the LOCO regime; see table 3 for details.

	good-and-fastest			*best-and-fastest*		
algorithm selected by...	**GF**	**NB**	**SB**	**BF**	**NB**	**SB**
% instances correctly classified	40	26	11	28	25	23
mean % matching variables	89.8	73.3	71.7	85.5	73.3	86.2

order problems in place of AD^3—for many instances, these provide solutions close in labelling to the lowest-energy, and do so much faster than AD^3.

To explore whether our method can also draw more subtle distinctions, we now examine the distribution of algorithms it selects for pairwise problems. 10 of the algorithms we consider are useful for these, in the sense of being good-and-fastest for at least one instance. Table 5 shows the confusion matrix for true and predicted good-and-fastest algorithms amongst these 10. Certain groups of problems can be distinguished based on structural properties, such as partitioning problems to be solved with KL, or very large instances that only run to completion with ICM. Our model correctly makes these distinctions. Other distinctions are even more subtle—such as whether to use α-expansion, TRW-S, or FastPD for a pairwise problem of moderate size. Our model is able to select between these three algorithms, making the correct choice for 75% of instances.

LOCO regime. We also tested our models and baselines in an even harder 'leave one class out' (LOCO) regime, where for each problem class C in turn, we train on all instances from classes other than C, and test on those in C; the final performance is given by a weighted mean over classes. This tests generalisation to classes absent from the training set, which is relevant when the user does not wish to train our model on her classes. The results are presented in table 6.

For selecting the good-and-fastest algorithm, model GF still performs well in LOCO regime, selecting algorithms labelling 89.8% of variables correctly, and exceeding both naïve and strong baselines by over 15%. Moreover, we correctly choose the good-and-fastest algorithm 14% more often than the baselines.

For the best-and-fastest task, model BF results in 85.5% of variables being correctly labelled, significantly exceeding the naïve baseline at 73.3% and comparable with the strong baseline at 86.2%. These results demonstrate that our selection models are strong enough to generalise *across* the hidden problem classes, going beyond discovering and recalling distinguishing features of these.

7 Conclusions

We have presented a method to automatically choose the best inference algorithm to apply to an input problem instance. It selects an inference algorithm that labels 96% of variables the same as the best available algorithm for that instance. Our method is over 88× faster than exhaustively trying all algorithms. The experiments show that our automated selection methods successfully generalise across problem instances and importantly, even across problem classes.

References

1. Kolmogorov, V., Rother, C.: Comparison of energy minimization algorithms for highly connected graphs. In: ECCV. (2006) 1–15
2. Szeliski, R., Zabih, R., Scharstein, D., Veksler, O., Kolmogorov, V., Agarwala, A., Tappen, M., Rother, C.: A comparative study of energy minimization methods for markov random fields with smoothness-based priors. IEEE Trans. on PAMI **30**(6) (2008) 1068–1080
3. Kappes, J., Andres, B., Hamprecht, F., Schnörr, C., Nowozin, S., Batra, D., Kim, S., Kausler, B., Kröger, T., Lellmann, J., Komodakis, N., Savchynskyy, B., Rother, C.: A comparative study of modern inference techniques for structured discrete energy minimization problems. IJCV (2015) 1–30
4. Bishop, C.: Pattern recognition and machine learning. Springer (2006)
5. Kolmogorov, V.: Convergent tree-reweighted message passing for energy minimization. IEEE Trans. on PAMI **28**(10) (2006) 1568 – 1583
6. Boykov, Y., Veksler, O., Zabih, R.: Fast approximate energy minimization via graph cuts. IEEE Trans. on PAMI **23**(11) (2001) 1222–1239
7. Greig, D.M., Porteous, B.T., Seheult, A.H.: Exact maximum a posteriori estimation for binary images. Journal of the Royal Statistical Society **51**(2) (1989) 271–279
8. Rother, C., Kolmogorov, V., Lempitsky, V., Szummer, M.: Optimizing binary MRFs via extended roof duality. In: CVPR. (2007)
9. Storvik, G., Dahl, G.: Lagrangian-based methods for finding MAP solutions for MRF models. IEEE Transactions on Image Processing **9** (2000) 469–479
10. Guignard, M., Kim, S.: Lagrangean decomposition: a model yielding stronger Lagrangean bounds. Math. Prog. (1987) 215–228
11. Komodakis, N., Paragios, N., Tziritas, G.: Mrf optimization via dual decomposition: Message-passing revisited. In: ICCV. (2007) 1–8
12. Kappes, J., Savchynskyy, B., Schnörr, C.: A bundle approach to efficient MAP-inference by lagrangian relaxation. In: CVPR. (2012)
13. Martins, A.F.T., Figueiredo, M.A.T., Aguiar, P.M.Q., Smith, N.A., Xing, E.P.: AD3: Alternating directions dual decomposition for MAP inference in graphical models. JMLR **16** (2015) 495–545
14. Andres, B., Kappes, J.H., Köthe, U., Schnörr, C., Hamprecht, F.A.: An empirical comparison of inference algorithms for graphical models with higher order factors using OpenGM. In: DAGM. Number 6376 (2010) 353–362
15. Alahari, K., Kohli, P., Torr, P.: Dynamic hybrid algorithms for discrete map mrf inference. IEEE Trans. on PAMI **32**(10) (2010) 1846–1857
16. Ishikawa, H.: Higher-order clique reduction in binary graph cut. In: CVPR. (2009) 2993–3000
17. Ishikawa, H.: Transformation of general binary MRF minimization to the first order case. IEEE Trans. on PAMI **33**(6) (2011) 1234–1249
18. Fix, A., Gruber, A., Boros, E., Zabih, R.: A graph cut algorithm for higher-order markov random fields. In: ICCV. (2011)
19. Kschischang, F.R., Frey, B.J.: Factor graphs and the sum-product algorithm. IEEE Trans. on Information Theory **47**(2) (2001) 498–519
20. Wainwright, M.J., Jaakkola, T.S., Willsky, A.S.: MAP estimation via agreement on (hyper)trees: Message-passing and linear-programming approaches. IEEE Transactions on Information Theory **51**(11) (2005) 3697–3717

21. Sontag, D., Meltzer, T., Globerson, A., Weiss, Y., Jaakkola, T.: Tightening LP relaxations for MAP using message-passing. In: Proc. UAI. (2008) 503–510
22. Doppa, J.R., Kumar, P., Wick, M., Singh, S., Salakhutdinov, R.: ICML 2013 workshop on inferning. http://inferning.cs.umass.edu/ (2013)
23. Guillaumin, M., Van Gool, L., Ferrari, V.: Fast energy minimization using learned state filters. In: CVPR. (2013)
24. Conejo, B., Komodakis, N., Leprince, S., Avouac, J.P.: Inference by learning: Speeding-up graphical model optimization via a coarse-to-fine cascade of pruning classifiers. In: NIPS. (2014) 1–9
25. Stoyanov, V., Eisner, J.: Fast and accurate prediction via evidence-specific MRF structure. In: ICML Workshop on Inferning. (2012)
26. Roig, G., Boix, X., De Nijs, R., Ramos, S., Kuhnlenz, K., Van Gool, L.: Active map inference in crfs for efficient semantic segmentation. In: ICCV. (2013) 2312–2319
27. Jiang, J., Moon, T., Daumé III, H., Eisner, J.: Prioritized asynchronous belief propagation. In: ICML Workshop on Inferning. (2013)
28. Rice, J.R.: The algorithm selection problem. Adv. Comps. **15** (1976) 65–118
29. Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: SATzilla: Portfolio-based algorithm selection for SAT. J. Artif. Intel. Res. **32** (2008) 565–606
30. Kotthoff, L., Gent, I.P., Miguel, I.: A preliminary evaluation of machine learning in algorithm selection for search problems. In: Symp. Comb. Search. (2011)
31. Lellmann, J., Schnörr, C.: Continuous multiclass labeling approaches and algorithms. SIAM J. Im. Sci. **4**(4) (2011) 1049–1096
32. Nowozin, S., Rother, C., Bagon, S., Sharp, T., Yao, B., Kohli, P.: Decision tree fields. In: ICCV. (2011)
33. Gould, S., Fulton, R., Koller, D.: Decomposing a scene into geometric and semantically consistent regions. In: ICCV. (2009)
34. Hoiem, D., Efros, A.A., Hebert, M.: Recovering occlusion boundaries from an image. IJCV **91**(3) (2011) 328–346
35. Kim, S., Nowozin, S., Kohli, P., Yoo, C.D.: Higher-order correlation clustering for image segmentation. In: NIPS. (2011)
36. Andres, B., Kappes, J.H., Beier, T., Köthe, U., Hamprecht, F.A.: Probabilistic image segmentation with closedness constraints. In: ICCV. (2011)
37. Brandes, U., Delling, D., Gaertler, M., Görke, R., Hoefer, M., Nikoloski, Z., Wagner, D.: On modularity clustering. IEEE Trans. on KDE **2**(2) (2008) 172–188
38. Andres, B., Kroeger, T., Briggman, K.L., Denk, W., Korogod, N., Knott, G., Koethe, U., Hamprecht, F.A.: Globally optimal closed-surface segmentation for connectomics. In: ECCV. (2012)
39. Jaimovich, A., Elidan, G., Margalit, H., Friedman, N.: Towards an integrated protein-protein interaction network: a relational markov network approach. J. Comp. Biology **13**(2) (2006) 145–164
40. Yanover, C., Schueler-Furman, O., Weiss, Y.: Minimizing and learning energy functions for side-chain prediction. J. Comp. Biology **15**(7) (2008) 899–911
41. Shotton, J., Winn, J., Rother, C., Criminisi, A.: TextonBoost for image understanding: Multi-class object recognition and segmentation by jointly modeling appearance, shape and context. IJCV **81**(1) (2009) 2–23
42. Everingham, M., Van Gool, L., Williams, C.K.I., Winn, J., Zisserman, A.: The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html (2007)
43. Alexe, B., Deselaers, T., Ferrari, V.: What is an object? In: CVPR. (2010)

44. Isack, H., Boykov, Y.: Energy-based geometric multi-model fitting. IJCV **97**(2) (2012) 123–147
45. Fischler, M.A., Bolles, R.C.: Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. Comm. ACM **24**(6) (1981) 381–395
46. Rother, C., Kohli, P., Feng, W., Jia, J.: Minimizing sparse higher order energy functions of discrete variables. In: CVPR. (2009)
47. Kohli, P., Kumar, M., Torr, P.: P3 & beyond: Solving energies with higher order cliques. In: CVPR. (2007)
48. Kohli, P., Ladicky, L., Torr, P.: Robust higher order potentials for enforcing label consistency. In: CVPR. (2008)
49. Bergtholdt, M., Kappes, J., Schnörr, C.: Learning of graphical models and efficient inference for object class recognition. In: DAGM. (2008)
50. Komodakis, N., Tziritas, G., Paragios, N.: Performance vs computational efficiency for optimizing single and dynamic MRFs: Setting the state of the art with primal-dual strategies. CVIU **112**(1) (2008) 14–29
51. Besag, J.: On the statistical analysis of dirty pictures. Journal of the Royal Statistical Society **48**(3) (1986) 48–259
52. Kernighan, B.W., Lin, S.: An efficient heuristic procedure for partitioning graphs. Bell Sys. Tech. J. **49**(2) (1970) 291–307
53. Sontag, D., Choe, D.K., Li, Y.: Efficiently searching for frustrated cycles in MAP inference. In: Proc. UAI. (2012) 795–804
54. Criminisi, A., Shotton, J., Konukoglu, E.: Decision forests for classification, regression, density estimation, manifold learning and semi-supervised learning. Microsoft Research Cambridge, Tech. Rep. MSRTR-2011-114 (2011)