

Accelerating Deformable Part Models with Branch-and-Bound

Iasonas Kokkinos

Abstract Deformable Part Models (DPMs) play a prominent role in current object recognition research, as they rigorously model the shape variability of an object category by breaking an object into parts and modelling the relative locations of the parts. Still, inference with such models requires solving a combinatorial optimization task. In this Chapter, we will see how Branch-and-Bound can be used to efficiently perform inference with such models. Instead of evaluating the classifier score exhaustively for all part locations and scales, such techniques allow us to quickly focus on promising image locations. The core problem that we will address is how to compute bounds that accommodate part deformations; this allows us to apply Branch-and-Bound to our problem. When comparing to a baseline DPM implementation, we obtain exactly the same results but can perform the part combination substantially faster, yielding up to tenfold speedups for single object detection, or even higher speedups for multiple objects.

1 Introduction

In computer vision the term ‘shape’ is used in a strict sense to refer to explicit geometric information, such as contours that correspond to surface boundaries, and in a broader sense to describe whatever is unaffected by appearance changes. The treatment of shape in terms of contours was the main theme of geometric 3D recognition [33, 36, 19, 53] before the advent of statistical techniques at the beginning of the previous decade. Shape has hence been used in high-level vision in its second sense, through features such as Shape Context [1], Scale-Invariant Feature Transforms (SIFT) [34] or Histograms-of-Gradients (HOG) [5], which describe shape in

Iasonas Kokkinos
Center for Visual Computing, Centrale-Supélec and INRIA-Saclay,
Grande Voie des Vignes, 92295 Chatenay-Malabry, France
e-mail: iasonas.kokkinos@ecp.fr

terms of distributions on invariant features, such as gradient histograms, or Convolutional Neural Networks [28, 15, 16, 37], which learn transformation-robust object representations. While such features provide a robust description of shape to object detection tasks, a more explicit representation of shape is desirable in tasks which require a more detailed object description, such as pose estimation.

Such a representation is currently most successfully provided by deformable part models (DPM), defined in terms of a set of parts that deform with respect to each other. Such models have been shown to largely outperform rigid detectors on challenging benchmarks when trained discriminatively [10], and have become a standard in object detection and pose estimation research [52, 50]. At the heart of these models lies the optimization of a merit function with respect to the part displacements. In this work we take the merit function for granted, using the discriminatively trained models of [10], and focus on the computational efficiency of the optimization problem.

The most common detection algorithm used in conjunction with DPMs relies on the Generalized Distance Transform (GDT) algorithm [11], whose complexity is linear in the image size. Despite the algorithm’s striking efficiency this approach still needs to thoroughly evaluate the object score everywhere in the image, which can become time demanding. In this work we introduce bounding-based techniques, which extend to part-based models the Branch-and-Bound (BB) and Cascaded Detection (CD) techniques used for Bag-of-Word classifiers in [29], [30] respectively. For this we exploit and adapt the Dual Tree (DT) data structure of [18] to provide the bounds required by BB/CD; we originally presented this technique in [24], but the current Chapter provides a more thorough presentation and evaluation.

Our method is fairly generic; it applies to any star-shaped graphical model involving continuous variables, and pairwise potentials expressed as separable, decreasing binary potential kernels. We evaluate our technique using the mixture-of-deformable part models of [10]. Our algorithm delivers *exactly the same* results, but is substantially faster. We also develop a multiple-object detection variation of the system, where all object hypotheses are inserted in the same priority queue. If our task is to find the best (or k-best) object hypotheses in an image this can result in more than a 100-fold speedup. These speedups refer to the part combination process, after the unary part scores have been computed.

This Chapter is structured as follows: after briefly covering prior work in Section 2, in Section 3 we first describe the cost function used in DPMs, and then motivate the use of bounding-based techniques for efficient object detection. In Section 4 we start with a high-level description of BB and CD in a general setting, and then proceed to describe the details of their implementation for detection with DPMs: in Section 4.3 we describe how we bound the DPM score and in Section 4.3.3 we describe how we keep the computation of the bound tractable. Qualitative results are provided throughout the text; we provide systematic experimental results on the Pascal VOC dataset in Section 5.

2 Previous Work on Efficient Detection

Cascade Detection (CD) algorithms were introduced in the beginning of the previous decade in the context of boosting [47] and coarse-to-fine detection [14] and have led to a proliferation of computer vision applications. However these works deal with ‘monolithic’ object models, i.e. there is no notion of deformable parts in the representation. Incorporating parts can make detection more challenging, since combinatorial optimization problems emerge.

The combinatorics of matching have been extensively studied for rigid objects [19], while [35] used A^* for detecting object instances. For categories, recent works [4, 27, 43, 13, 39, 31] have focused on reducing the high-dimensional pose search space during detection by initially simplifying the cost function being optimized, mostly using ideas similar to A^* and coarse-to-fine processing. In the recent work of [10] thresholds pre-computed on the training set are used to prune computation and result in substantial speedups compared to GDTs. However this approach requires tuning thresholds using the training set and comes only with approximate guarantees.

A line of work which brought new ideas into detection has been based on Branch-and-bound (BB). Even though BB was studied at least as early as [20], it was typically considered to be appropriate only for geometric matching/instance-based recognition. A most influential paper has been the Efficient Subwindow Search (ESS) technique of [29], where an upper bound of a bag-of-words classifier score delivers the bounds required by BB. Later [32] combined Graph-Cuts with BB for object segmentation, while in [30] a Cascaded Detection (CD) system for efficient detection was devised by introducing a minor variation of BB.

Our work is positioned with respect to these works as follows: unlike existing BB/CD works [32, 29, 30, 31], we use the DPM cost and thereby accommodate parts in a rigorous energy minimization framework. And unlike the pruning-based works [4, 13, 10, 39], we do not make any approximations or assumptions about when it is legitimate to stop computation; our method is exact.

We obtain the bounds required by BB/CD by adapting the Dual Tree data structure of [18], originally developed in the context of nonparametric density estimation. To the best of our knowledge, Dual Trees have been minimally used in object detection; we are only aware of the work in [21] which used Dual Trees to efficiently generate particles for Nonparametric Belief Propagation. Here we show that Dual Trees can be used for part-based detection, which is related conceptually, but entirely different technically.

A considerable body of work has been developed around the efficient approximation of the part scores of DPMs [40, 46, 7, 44, 6, 25, 26, 42, 41]. These can be understood as complementary to the work presented here, in the sense that we consider that the part scores have been computed, and tackle the remaining combinatorial problem of ‘assembling’ the object parts. We actually deal with the approximations incurred by fast part computation in [25, 26] and show that they can be seamlessly integrated into the bounding-based framework presented here.

3 Object Detection with DPMs

The state \mathbf{x} of a general DPM, e.g. [49, 12] encodes the object’s putative configurations in terms of P position vectors, $x_p, p = 1, \dots, P$:

$$\mathbf{x} = \{x_1, \dots, x_P\}, \quad x_p \in [1, K] \times [1, L], \quad (1)$$

where x_p can correspond to any of the $K \times L = N$ image pixels. For the most general graph topology N^P part combinations would need to be considered, severely raising the computational cost of DPMs. Coming up with algorithms of a smaller complexity is thus crucial for fast object detection in the presence of deformations.

Star-shaped DPMs [8, 9, 10] take a step in this direction, by constraining the model’s topology so that a single part is designated as the ‘root’ node of a graph, and the remaining parts as the leaf nodes. All leaf nodes $p = 2, \dots, P$ are connected exclusively with the root node $p = 1$, i.e. we have a star-shaped graphical model.

If the root node is placed at x , the score for a part p being placed at x' is given by $m_p(x', x) = U_p(x') + B_p(x', x)$, where the unary term $U_p(x')$ measures the fidelity of the image around position x' to the appearance model of the p -th part and the pairwise term $B_p(\mathbf{x}_p, \mathbf{x}'_p)$ measures the geometric consistency of the positions of part p with respect to the root’s position.

In particular, in [9] the appearance term $U_p(x') = \langle w_p, H(x') \rangle$ is formed as the inner product of a HOG feature $H(x')$ at x' with a discriminant w_p for p . This captures the local fidelity of the image to the appearance model of part p . The pairwise terms constrain the relative location x' of each part p w.r.t. the location x of the root in terms of a quadratic function of the form:

$$B_p(x', x) = - (x' - x - \mu_p)^T I_p (x' - x - \mu_p), \quad (2)$$

where $I_p = \text{diag}(H_p, V_p)$ is a diagonal ‘precision’ matrix, μ_p is the nominal relative location vector, and for the root node, $p = 1$, we consider:

$$B_1(x', x) = \begin{cases} -\infty, & x' \neq x \\ 0, & x' = x \end{cases} \quad (3)$$

for convenience, practically ensuring that the ‘root’ part is pinned at position x . We can view the expression in Eq. 2 as related to the log-likelihood of the relative locations under a diagonal-covariance Gaussian model.

A star-shaped DPM scores a configuration $\mathbf{x} = (x_1, \dots, x_P)$ by summing the merit of its parts:

$$M(\mathbf{x}) = \sum_{p=1}^P m_p(x_p, x_1). \quad (4)$$

To decide if a location x can serve as the root of an object, we maximize over all configurations that place the root at x :

$$S(x) \doteq \max_{\mathbf{x}:x_1=x} M(\mathbf{x}) \stackrel{\text{Eq. 4}}{=} \max_{\mathbf{x}} \sum_{p=1}^P m_p(x_p, x) \quad (5)$$

$$= \sum_{p=1}^P m_p(x), \quad \text{where} \quad m_p(x) \doteq \max_{x'} U_p(x') + B_p(x', x). \quad (6)$$

To go from Eq. 5 to Eq. 6 we use the fact that $M(\mathbf{x})$ factorizes over x_p ; $m_p(x)$ serves as notation for the ‘messages’ being sent from the part nodes to the root node, and is obtained by eliminating x' from $m_p(x_p, x)$. The part-to-root message passing described by Eq. 6 is identical to the leaf-to-parent message passing equations of the Max-Product algorithm [23] if we use the logarithm of the probabilities.

The flow of computation of this algorithm is illustrated in Fig. 1. As one can see, the part scores have sharply peaked responses, but tend to provide many false positives, while the result of message passing (left-to-right transition) and summation (top-to-bottom transition), performed at the root node provides a well-localized estimate of the object’s position.

3.1 Complexity of Object Detection with Star-Shaped DPMs

During detection our goal is to identify either (a) $M^* = \{\arg \max_x S(x)\}$, or (b) $M^\theta = \{x : S(x) \geq \theta\}$. We will refer to case (a) as **first-best detection** and (b) as **threshold-based detection**. Case (a) is encountered commonly in pose estimation, or during latent SVM training, when maximizing over latent variables. Case (b) corresponds to the common setup for detection, where all image positions scoring above a threshold are used as object hypotheses.

A naive approach to solve both of those cases is to consider all possible values of x , evaluate $S(x)$ on them and then recover the solutions. The complexity of this would be $O(PN^2)$, where $N = |\{x\}|$ is the cardinality of the set of possible locations considered (Eq. 6 suggests doing N maximizations per point, and we have N points and P parts).

But due to the particular form of the pairwise term in Eq. 2, the maximization within each summand $m_p(x)$ in Eq. 6 lends itself to efficient computation in batch mode for all values of x using a Generalized Distance Transform (GDT) [11], in time $O(N)$. So the standard approach taken so far is to maximize each summand separately with GDTs and then add up the scores at all image locations to obtain the overall object score; this yields an overall complexity of $O(PN)$. Even though the $O(PN)$ complexity achieved with GDTs is remarkably fast (requiring 1-2 seconds for multi-scale processing of VGA-sized images), the N factor can still slow things down for large images. This motivates an approach to detection that can potentially operate with a complexity that is sublinear in the number of pixels - which can only be accomplished if we can somehow ‘skip’ unpromising pixel positions. This is implemented in a rigorous, fail-proof manner with bounding-based techniques, as described below.

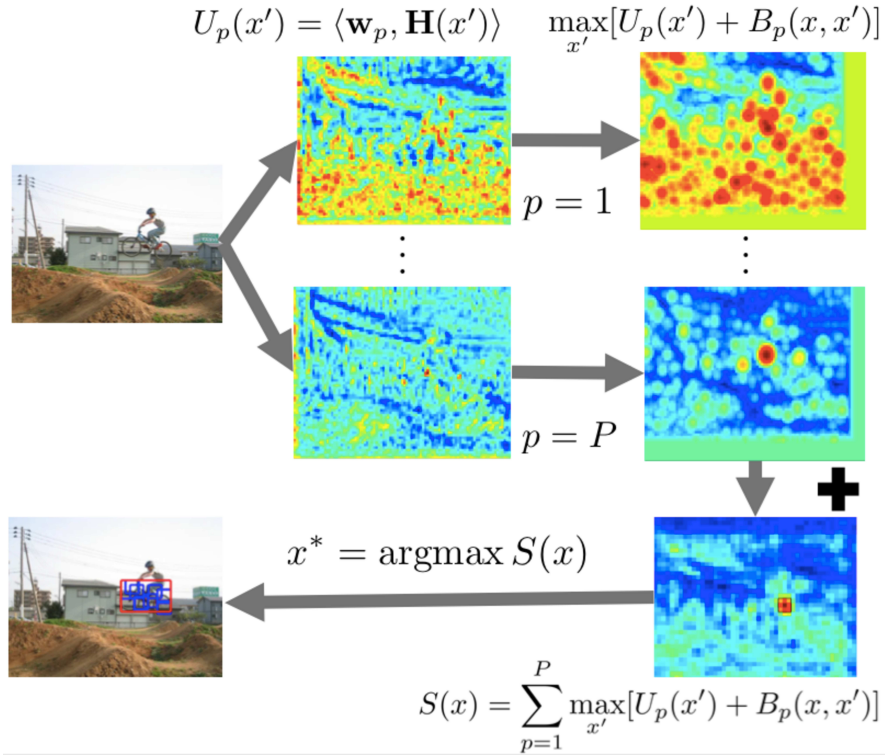


Fig. 1 Pipeline of object detection with star-shaped Deformable Part Models: the image features are filtered with a set of templates, providing part-specific unary terms. These are used to pass messages regarding the object's position to the root, where messages are summed to compute the overall score. From the maximum of this score we can obtain the best-scoring object hypothesis, as well as the position of the parts that support it.

4 Bounding-based Detection with DPMs

Our approach to accelerating detection starts from the observation that if we use a fixed threshold for detection, e.g. -1 for an SVM classifier, then the GDT-based approach outlined above can be wasteful. In particular it treats equally all image locations, even when we can quickly realize that some of them score far below the threshold. This is illustrated in Fig. 2: in (a) we show the part-root configuration that gives the maximum score, and in (b) the score of a bicycle model from [10] over the whole image domain. The tiny part of the image scoring above a conservative threshold of -1 is encircled by a black contour in (b).

Our approach instead speeds up detection by upper bounding the score of the detector within *intervals* of x . These bounds can be rapidly obtained using low-cost operations, as will be detailed in the following. Having a bound allows us to use

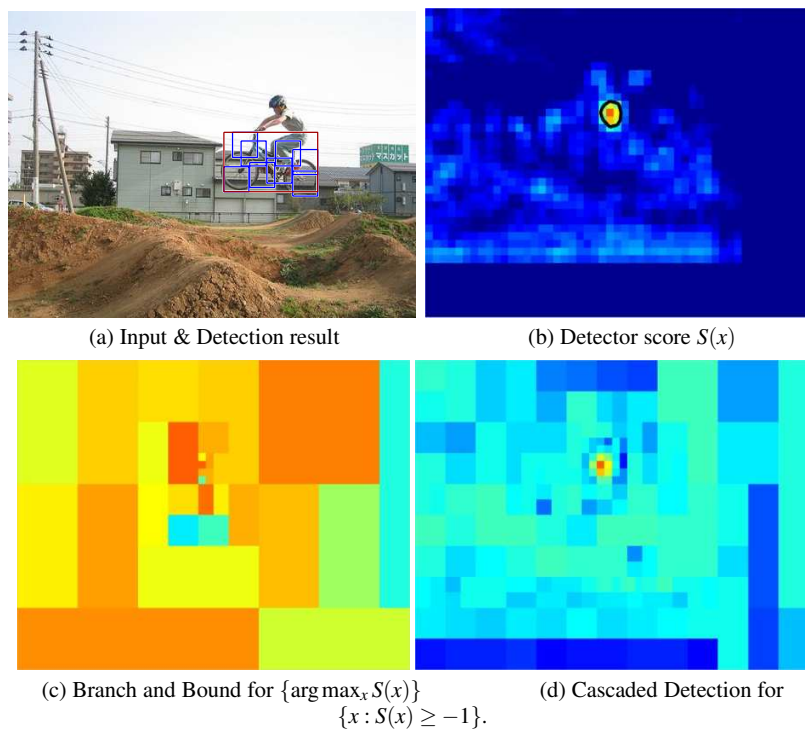


Fig. 2 Motivation for a bounding-based approach (note that the classifier is designed to ‘fire’ on the top-left corner of the object’s bounding box): standard part-based models evaluate a classifier’s score $S(x)$ over the whole image domain. Typically only a tiny portion of the image domain should have large scores - in (b) we draw a black contour around $\{x : S(x) > -1\}$ for an SVM-based classifier. Our algorithm ignores large intervals with low $S(x)$ by upper bounding their values, and postponing their exploration in favor of more promising ones. In (c) we show as heat maps the upper bounds of the intervals visited by our algorithm until the strongest location was explored, and in (d) of the intervals visited until all locations x with $S(x) > -1$ were explored.

a coarse-to-fine strategy that starts from an interval containing all possible object locations and then gradually subdivides it to refine the bounds on promising sub-intervals, while avoiding the exploration of less promising ones.

This is demonstrated in Fig. 2(c,d) where we show as heat maps the upper bounds of the intervals visited by our approach for first-best and threshold-based detection respectively. The parts of the image where the heat maps are more fine-grained correspond to image locations that seemed promising and were explored at a finer level. Coarse-grained parts correspond to intervals whose upper bound was low, and the refinement of the bound was therefore avoided.

Even though the number of operations performed by our bounding-based approach is image-dependent, we can say that it is roughly *logarithmic in the image size*, since our approach recursively subdivides the explored intervals (the best-case

complexity of our algorithm is $O(|M|P \log N)$. So rescaling an image by a factor of 2 will require roughly two more iterations for our algorithm, while for the GDT-based computation it will require four times the original number of operations (since we now have four times as many pixels).

We now make these high-level ideas more concrete by first describing Branch-and-Bound and Cascaded Detection, which respectively address the first-based and threshold-based detection problems outlined in Section 3.1, and then get into the technical details involved in the bound computation.

4.1 First-best detection with Branch and Bound

Branch and Bound (BB) can be used a generic maximization algorithm for non-convex or even non-differentiable functions. BB searches for the interval containing the function's maximum by using a prioritized search strategy; the priority of an interval is determined by the function's upper bound within it. The operation of BB for the maximization of a function over a domain X_0 is illustrated in Fig. 3: BB finds the maximum of a function by using a prioritized search strategy over intervals; at each step branching first takes place, where an interval - X_0 , here - is split into two subintervals, X_1, X_2 . Then bounding takes place, where the value of the function is upper bounded within each of the new intervals. This upper bound serves as a priority and dictates which interval is explored next.

The main hurdle in devising a BB algorithm is coming up with a bound that is relatively tight and also easy to compute. In Fig. 3 a parabola is used to upper bound a complex, non-concave function; the interval's priority can then be rapidly estimated by constructing an analytical upper bound on the parabola's value.

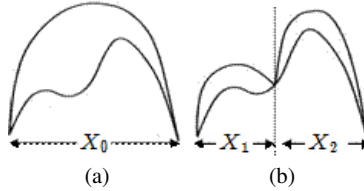


Fig. 3 Illustration of how Branch-and-Bound proceeds to maximize a complex, non-concave function within an interval by branching and bounding the function within intervals. Please see text for details.

More concretely, if the function we want to maximize is $S(x)$, BB requires that we are able to construct an upper bound of this function's value within an interval. With a slight abuse of notation we introduce:

$$S(X) \doteq \max_{x \in X} S(x), \quad (7)$$

i.e. we ‘overload’ function symbols to take intervals as arguments. Denoting the upper bound to function S as \bar{S} the requirement is that:

$$\bar{S}(X) \geq S(X) = \max_{x \in X} S(x) \quad \forall X, \quad \bar{S}(\{x\}) = S(x), \quad (8)$$

i.e. on a singleton our bound should be tight.

With such a bounding function at our disposal, BB searches for the maximum of a function using prioritized search over intervals, as illustrated by the pseudocode in Table 1. Starting from an interval corresponding to all possible object locations (X_0) the algorithm splits it into subintervals and uses the upper bounds of the latter as priorities in search. At each step the algorithm visits the most promising subinterval and the algorithm terminates when the first singleton interval, say x , is popped. This is guaranteed to be a global maximum: since the bound is tight for singletons, we know that the solutions contained in the remaining intervals of the priority queue will score below or equal to x , since the upper bound of their scores is at most $\bar{S}(\{x\}) = S(x)$.

Branch-and-Bound	Cascaded Detection
$M^* = BB(X_0, \bar{S})$	$M^\theta = CD(X, \bar{S}, \theta)$
INITIALIZE: $\mathcal{Q} = \{(X_0, \bar{S}(X_0))\}$	if $\bar{S}(X) < \theta$ then
while 1 do	RETURN $\{\}$
$X = \text{Pop}[\mathcal{Q}]$	end if
if Singleton $[X]$ then	if Singleton $[X]$ then
RETURN X { // First singleton: best	RETURN X { //Singleton with score \geq
X }	θ }
end if	end if
$[X_1, X_2] = \text{Branch}[X]$	$[X_1, X_2] = \text{Branch}[X]$
Push $[\mathcal{Q}, (X_1, \bar{S}(X_1))]$, Push $[\mathcal{Q}, (X_2, \bar{S}(X_2))]$	$M^\theta = CD(X_1, \bar{S}, \theta) \cup CD(X_2, \bar{S}, \theta)$
end while	RETURN M^θ

Table 1 Pseudocode for Brand-and-Bound (BB) and Cascaded Detection (CD). Both algorithms use a KD-tree for the image domain, where the root node, X_0 , corresponding to an interval for the whole image domain and the leaves to singletons (pixels). BB starts from the root interval and performs prioritized search to find the interval containing the best configuration. CD starts from the root node and performs a Center-Left-Right traversal of the tree to return all singletons scoring above a fixed threshold.

4.2 Threshold-based detection: Cascaded Detection

The BB algorithm described above is appropriate if we search for the first-best (or k-best) scoring configuration(s). This is typically the case for tasks such as training or pose estimation. But for detection we typically want to find all object locations that score above a threshold θ . To accommodate this in [24] we proposed to use prioritized search, but stop when the popped interval scores below θ . This will return

all singletons scoring above θ indeed, but it is more efficient to use a cascaded detection algorithm similar to [30], which avoids the overhead of inserting/removing elements from a priority queue and is also easy to parallelize.

In particular, our adaptation of the algorithm in [30] uses a tree of intervals, with the root corresponding to the whole domain and the leaves to singletons (single pixels). The algorithm, described in pseudocode in Table 1, starts from the root and recursively traverses the tree in a center-left-right manner. At the center we check if the upper bound of the current node is above threshold. If it is not, we return an empty set, meaning that none of the node’s children can contain an object above threshold. Otherwise, if the node is singleton, we return the actual location. Finally if the node is non-singleton we recurse to its left and right children (subintervals), and return the union of their outputs.

4.3 Bounding the DPM score

Having given a high-level description of BB/CD we describe in this subsection how we compute the bounds and in the following one how we organize the computation.

The main operation required by both algorithms is to compute ‘cheap’ upper bounds of the DPM score function $S(x)$ within an interval X . From Eq. 6 we have that $S(x) = \sum_p m_p(x)$ and we are now concerned with forming an upper bound for the quantity $S(X) = \max_{x \in X} \sum_p m_p(x)$. We can upper bound $S(X)$ as follows:

$$\bar{S}(X) \doteq \sum_p \bar{m}_p(X) \geq \sum_p m_p(X) = \sum_p \max_{x \in X} m_p(x) \geq \max_{x \in X} \sum_p m_p(x) = S(X), \quad (9)$$

where $\bar{m}_p(X)$ are upper bounds on the value of $m_p(x)$ within X - we describe these below. On the left we have the construction of our upper bound and on the right the quantity we wanted to bound in the first place. The first inequality stems from the fact that $\bar{m}_p(X)$ is an upper bound for $m_p(X)$, the next equality from the definition of the ‘overloaded’ notation for $m(X)$. The second inequality stems from the fact that $\max_{x \in X} f(x) + \max_{x \in X} g(x) \geq \max_{x \in X} f(x) + g(x)$ for any two functions f, g , and any interval X . We clarify that the maximization showing up here is over the interval X for which the upper bound is computed; it is not the maximization implicit in the definition of the messages in Eq. 6.

As we will focus on the individual summands $\bar{m}_p(X)$, we omit the p subscript. Based on Eq. 6, $\bar{m}(X)$ should satisfy:

$$\bar{m}(X) \geq m(X) \stackrel{\text{Eq. 7}}{=} \max_{x \in X} m(x) \stackrel{\text{Eq. 6}}{=} \max_{x \in X} \left[\max_{x' \in X'} m(x', x) \right], \quad (10)$$

where X and X' do not need to be identical (by the definition of Eq. 6 X' is the whole image domain). We now proceed to describe how we compute the relevant bounds efficiently.

4.3.1 Dual Trees and Domain Partitioning

We decompose the computation of the upper bound in Eq. 10 into smaller parts by using the partitions $X = \cup_{d \in D} X_d$, $X' = \cup_{s \in S} X_s$ as illustrated in Fig. 4. We call points contained in X' the source locations and points in X the domain locations, with the intuition that the points in X' contribute to a score in X . Making reference to Fig. 4, the ‘domain’ intervals-d could be the letters and the ‘source’ intervals could be the numbers.

For a given partition of X, X' we can rewrite $m(X)$ in Eq. 10 as:

$$m(X) = \max_d \max_{x \in X_d} \max_s \max_{x' \in X_s} m(x', x) = \max_d \max_s \mu_d^s, \quad \text{where} \quad (11)$$

$$\mu_d^s \doteq \max_{x \in X_d} \max_{x' \in X_s} m(x', x). \quad (12)$$

The quantity μ_d^s quantifies the maximal contribution of any source-interval point X_s to any domain-interval point X_d ; and $m(X)$ expresses the maximal contribution that any point within any source-interval can have to any point within any domain-interval.

In order to compute $m(X)$ we have at our disposal a range of partitions for the domain and source points to choose from, represented using separate KD-trees (hence the ‘Dual Tree’ term). As we illustrate in Fig. 6 and further detail in Section 4.3.3, we start from coarse partitions of X, X' and iteratively refine and prune both. To describe how exactly this takes place we first provide bounds for the associated terms.

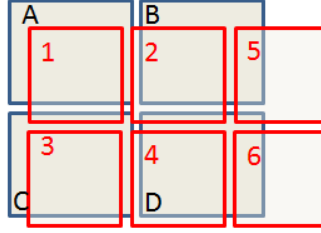


Fig. 4 We rely on a partition of the ‘source’ (red) and ‘domain’ (blue) points to derive rapidly computable bounds of their ‘interactions’. This could indicate for example that points lying in square 6 cannot have a large effect on points in square A, and therefore we do not need to go to a finer level of resolution to exactly estimate their interactions.

4.3.2 Bounding the appearance and geometric terms

Based on Eq. 12 and the the definition of $m(x', x)$ we can upper bound μ_d^s as follows:

$$\mu_d^s = \max_{x \in X_d} \max_{x' \in X_s'} (U(x') + B(x', x)) \leq \max_{x' \in X_s'} U(x') + \max_{x \in X_d} \max_{x' \in X_d} B(x', x) \doteq \overline{\mu}_d^s, \quad (13)$$

where again we use the fact that $\max_{x \in X} f(x) + \max_{x \in X} g(x) \geq \max_{x \in X} (f(x) + g(x))$.

For reasons that will become clear in Section 4.3.3, we also need to lower bound the quantity

$$\lambda_d^s = \min_{x \in X_d} \max_{x' \in X_s} (U(x') + B(x', x)). \quad (14)$$

This provides the weakest contribution to a domain point in X_d by any source point in X_s . To bound λ_d^s we have two options:

$$\underline{\lambda}_{d,1}^s = \max_{x' \in X_s} U(x') + \min_{x \in X_d} \min_{x' \in X_s} B(x', x) \leq \lambda_d^s, \quad (15)$$

$$\underline{\lambda}_{d,2}^s = \min_{x' \in X_s'} U(x') + \min_{x \in X_d} \max_{x' \in X_s} B(x', x) \leq \lambda_d^s. \quad (16)$$

The first bound corresponds intuitively to placing the point of X_s with the best unary score, say x_b to the worst location within X_s and then evaluating the support that it lends to the ‘hardest’ point of X_s . This is a lower bound since x_b will actually be in at least as good a position with respect to the hardest point. The second bound corresponds to taking the point of X_s with the worst unary score, say x_w and placing it at the location in X_s that supports the hardest point of X_d . This again is a lower bound since in practice the point of X_s supporting the hardest point in X_d will have at least as good a unary score as x_w does.

We combine these two bounds into a single and tighter lower bound as:

$$\underline{\lambda}_d^s = \max(\underline{\lambda}_{d,1}^s, \underline{\lambda}_{d,2}^s). \quad (17)$$

In [24] we had used only the first bound. Computing Eq. 17 requires some additional operations, but the bound is tighter and accelerates detection by a factor of 10% – 20%.

We can rapidly compute the terms involved in the bounds of Eq.s 13–16. First, the appearance-based terms, $\max_{x \in X_s} U(x)$ and $\min_{x \in X_s} U(x)$, can be computed with fine-to-coarse max-/min-imization through the KD-tree data structures. The overall complexity of computing all of the relevant terms turns out to be linear in the image size but with a particularly low constant, equal to the cost of the max/min operation.

Second, the geometric terms $\min_{x \in X_d} \max_{x' \in X_s} B(x', x)$, $\max_{x \in X_d} \max_{x' \in X_d} B(x', x)$ can be rapidly computed by exploiting the fact that X_d and X_s are rectangular. For clarity’s sake we now abandon the x notation for coordinates and switch to horizontal and vertical coordinates, (h, v) . Making reference to Fig. 5, we consider two 2D intervals, one for the domain-node X_d and one for the domain-node X_s ; X_d is centered at (h_d, v_d) , and has an horizontal/vertical half-range of η_d/v_d , while for X_s the respective quantities are $(h_s, v_s), \eta_s, v_s$. Using the (h, v) notation, we can write the pairwise term between two points, say $x \in X_d, x' \in X_s$ as:

$$\mathcal{G}_{x,x'} = -H(h - h')^2 - V(v - v')^2 \quad (18)$$

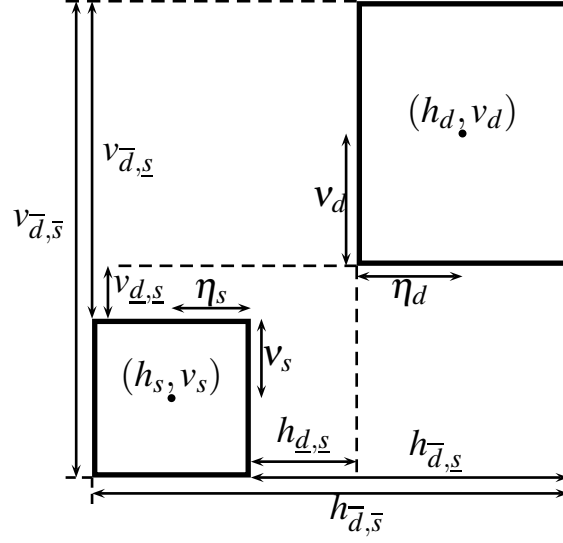


Fig. 5 Illustration of the terms involved in the geometric bound computations of Eq.s 24-26. The d/s subscript indicates quantities relevant to the domain/source intervals respectively (we want to bound the score within the domain interval, using contributions from the source interval).

where H, V are the diagonal elements of the precision matrix showing up in Eq. 2; we omit the effect of the means μ in Eq. 2 for simplicity, but they can be trivially incorporated in what follows.

Since the pairwise cost is separable in the horizontal and vertical dimensions, we can use distributivity to break the max-/min-imization operations along separate axes. In particular, we have to compute:

$$\mathcal{G}_{\bar{d},\bar{s}} \doteq \max_{x \in X_d} \max_{x' \in X_s} \mathcal{G}_{x,x'} = \max_{h \in X_d^h} \max_{h' \in X_s^h} -H(h-h')^2 + \max_{v \in X_d^v} \max_{v' \in X_s^v} -V(v-v')^2 \quad (19)$$

$$= -Hh_{\bar{d},\bar{s}}^2 - Vv_{\bar{d},\bar{s}}^2, \quad (20)$$

$$\text{where } h_{\underline{d},\underline{s}} \doteq \min_{h \in X_d^h} \min_{h' \in X_s^h} |h-h'|, \quad v_{\underline{d},\underline{s}} \doteq \min_{v \in X_d^v} \min_{v' \in X_s^v} |v-v'| \quad (21)$$

where we use \bar{i}, \underline{i} to indicate respectively that we are max-/min-imizing with respect to the points belonging to domain i , and denote by X^v, X^h the projections of a 2D interval X on the horizontal and vertical axes respectively. Similarly we get:

$$\mathcal{G}_{\underline{d},\bar{s}} \doteq \min_{x \in X_d} \max_{x' \in X_s} \mathcal{G}_{x,x'} = -Hh_{\underline{d},\bar{s}}^2 - Vv_{\underline{d},\bar{s}}^2, \quad \mathcal{G}_{\bar{d},\underline{s}} \doteq \min_{x \in X_d} \min_{x' \in X_s} \mathcal{G}_{x,x'} = -Hh_{\bar{d},\underline{s}}^2 - Vv_{\bar{d},\underline{s}}^2,$$

$$\text{where } h_{\underline{d},\bar{s}} \doteq \max_{h \in X_d^h} \min_{h' \in X_s^h} |h-h'|, \quad v_{\underline{d},\bar{s}} \doteq \max_{v \in X_d^v} \min_{v' \in X_s^v} |v-v'| \quad (22)$$

$$h_{\bar{d},\underline{s}} \doteq \max_{h \in X_d^h} \max_{h' \in X_s^h} |h-h'|, \quad v_{\bar{d},\underline{s}} \doteq \max_{v \in X_d^v} \max_{v' \in X_s^v} |v-v'| \quad (23)$$

For the particular configuration shown in Fig. 5 we have:

$$\begin{aligned}
 h_{\overline{d,s}} &= (h_d + \eta_d) - (h_s + \eta_s), v_{\overline{d,s}} = (v_d + v_s) - (v_s + v_d), \\
 h_{\overline{d,\underline{s}}} &= (h_d - \eta_d) - (h_s + \eta_s), v_{\overline{d,\underline{s}}} = (v_d - v_s) - (v_s + v_d), \\
 h_{\underline{d,s}} &= (h_d + \eta_d) - (h_s - \eta_s), \\
 v_{\underline{d,s}} &= (v_d + v_s) - (v_s - v_d)
 \end{aligned} \tag{26}$$

If we consider all possible relative placements of the two rectangles we obtain the following forms for the horizontal coordinate:

$$h_{\overline{d,s}} = \lceil |h_d - h_s| + (\eta_d - \eta_s) \rceil \tag{27}$$

$$h_{\overline{d,\underline{s}}} = \lceil |h_d - h_s| - (\eta_d + \eta_s) \rceil \tag{28}$$

$$h_{\underline{d,s}} = |h_d - h_s| + (\eta_d + \eta_s) \tag{29}$$

where $\lceil \cdot \rceil \doteq \max(\cdot, 0)$; similar expressions are used for the vertical coordinate after substituting v, v for h, η respectively.

4.3.3 Dual recursion and supporter pruning

We now describe how to control the complexity of maximizing over d and s in Eq. 11. The range of d and s will scale inversely with the cardinality of the intervals X_s, X_d , meaning that as the bounds get finer a larger number of terms will be involved; in the limit of singletons X_s, X_d we have a quadratic complexity in the number of pixels. We now describe how we use a coarse-to-fine algorithm to quickly prune the range of s involved for every d *without sacrificing accuracy*.

For this we use a Dual Recursion algorithm akin to the one originally introduced for Dual Trees by [18]. An illustration of how the algorithm works is provided in Fig. 6: starting from the root and going to the leaves, we recursively prune the range of source (s) intervals that should be used to bound the value at any domain (d) interval. In particular we ‘descend’ simultaneously on the source and domain trees; at the beginning (top) the root node of the source tree is used to bound the score of the root node of the domain tree and at the end the leaves of the source tree are used to compute the exact score of the leaves of the domain tree.

We use a recursive algorithm to limit the number of operations involved until getting to the leaves. Consider that in Eq. 11 we know that only a set of ‘supporter’ intervals $\mathcal{S}_d = \{s_i\}$ should be used in the bound computation relevant to a domain node-interval d . This means that all other source intervals cannot contribute something to any of the points contained in d . To reduce the number of operations when refining these domain and source intervals there are two observations that allow us to speed things up.

First, the children (sub-intervals) of d need to use only the children (sub-intervals) of \mathcal{S} , i.e. $\mathcal{S}_d \subset \cup_{\text{pa}(d)} \{\text{ch}(s_i)\}$, where pa, ch denote the parent and child operators. If any other points were necessary, these should have been included in the

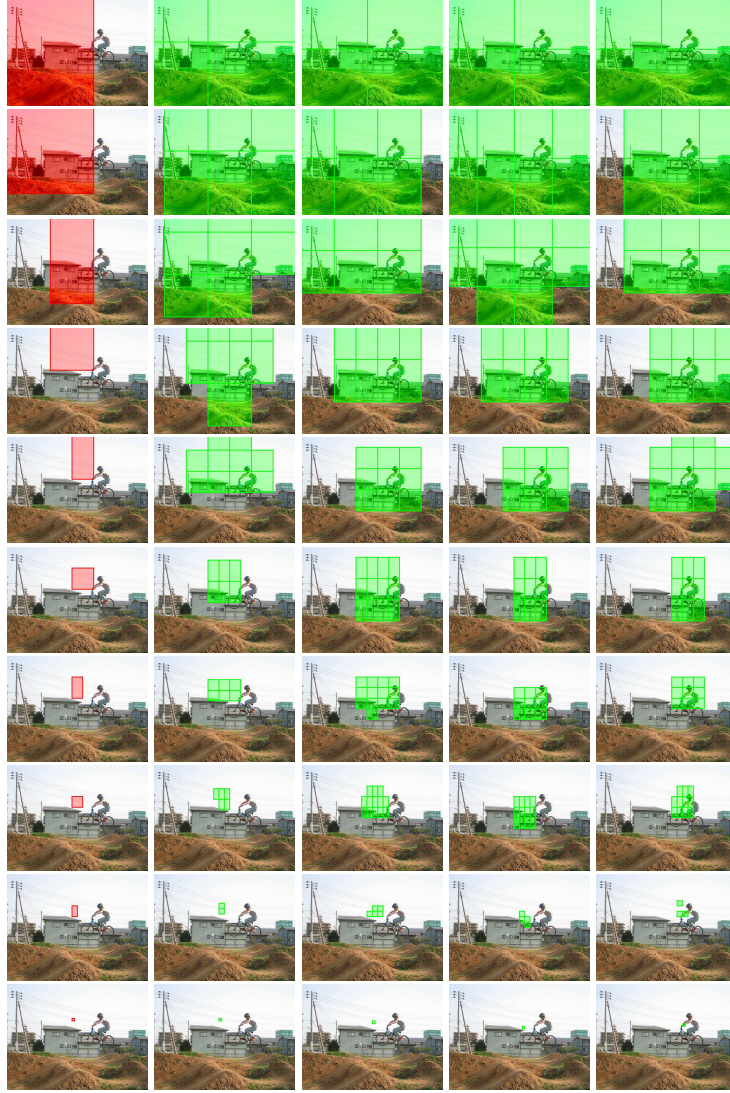


Fig. 6 Illustration of supporter pruning. The left column illustrates the succession of domain intervals that leads to the optimal object configuration. The next four columns illustrate the associated ‘supporters’ of that interval for four distinct object parts. Our algorithm starts at the top with a large interval that is supported by equally large intervals. On the way the domain and supporter intervals get refined. For each part the supporter intervals are also pruned in every step, making the overall optimization tractable. At the bottom row the domain interval is a singleton, and is supported by a single, and singleton, supporter interval. This indicates the optimal part placement for the given domain interval.

domain \mathcal{S}_d , by the definition of the ‘supporter’ intervals. Second, we can remove some elements of $\cup_{\mathcal{S}_{\text{pa}(d)}} \{\text{ch}(s_i)\}$ when forming \mathcal{S}_d , if we know that these cannot contribute to the optimal score at a domain node. This requires combining $\bar{\mu}_d^s$ and the lower bounds of $\underline{\lambda}_d^s$, and relies on the following rationale, illustrated in Fig. 7: consider that a node d has supporters m, n, o . If two nodes n and m support a node d and their bounds are related by $\bar{\mu}_d^n < \underline{\lambda}_d^m$, the descendants of interval n can be ignored from the following maximization. This is intuitively so because the bounds become tighter as the intervals become smaller, namely lower bounds increase and upper bounds decrease.

Below we provide a more concrete statement of this result, while making reference to Fig. 7: denote by s_1, s_2 the two children of a supporter node s , with s being one of the three right-most nodes and by d_1, d_2 the two children of node d , on the left. We have that

$$\bar{\mu}_d^s \geq \mu_d^s \geq \mu_{d_j}^{s_i} \quad \forall i \in \{1, 2\}, \forall j \in \{1, 2\} \quad (30)$$

The first inequality holds from the fact that $\bar{\mu}$ is an upper bound to μ . The second inequality holds because according to Eq. 12, $\mu_d^s \doteq \max_{x \in X_s} \max_{x' \in X_d} m(x, x')$ while $X_{s'} \subset X_s, X_{d'} \subset X_d$; so maximizing a function over a smaller domain will lead to a smaller quantity. In words, Eq. 30 tells us that the contribution $\mu_{d_j}^{s_i}$ of any child of s to any child of d cannot be larger than the upper bound $\bar{\mu}_d^s$ to the contribution of s to d .

We also have that:

$$\underline{\lambda}_d^s \leq \lambda_d^s \leq \lambda_{d_i}^s = \max(\lambda_{d_i}^{s_1}, \lambda_{d_i}^{s_2}), \quad i \in \{1, 2\}. \quad (31)$$

The first inequality holds from the fact that $\underline{\lambda}_d^s$ lower bounds λ_d^s . The second from the definition of $\lambda_d^s = \min_{x \in X_d} \max_{x' \in X_s} m(x, x')$ in Eq. 14, and the fact that $X_{d_i} \subset X_d$: since for $\lambda_{d_i}^s$ we are minimizing over a smaller set, it follows that $\lambda_{d_i}^s \geq \lambda_d^s$. Finally the last equality stems from the definition of λ_d^s and the fact that $X_s = X_{s_1} \cup X_{s_2}$. In words, Eq. 31 tells us that if we include both children, s_1, s_2 of s as potential supporters of a child d_i of d , the support at the worst point of d_i will be at least equal to $\underline{\lambda}_d^s$.

Having obtained the expressions in Eq. 30 and Eq. 31 for an arbitrary source node s , we now turn to how these expressions can be used in order to prune the children of a node, say n , in the light of the support delivered by another node, say m . The condition for doing this is that $\bar{\mu}_d^n \leq \underline{\lambda}_d^m$. If this holds, then it follows that

$$\mu_{d_j}^{n_i} \leq \max(\lambda_{d_j}^{m_1}, \lambda_{d_j}^{m_2}), \quad j \in \{1, 2\}, i \in \{1, 2\} \quad (32)$$

This tells us that within the domain interval d_j any point will be getting a support from m_1, m_2 that will be at least as good as the best support it can get from n_1 or n_2 . Therefore the intervals n_1, n_2 do not need to be considered anymore, and node n can be safely pruned - this is illustrated in Fig. 6 by the lack of connections between n_1, n_2 and the children of d . Concisely, we prune the children of supporter l to node

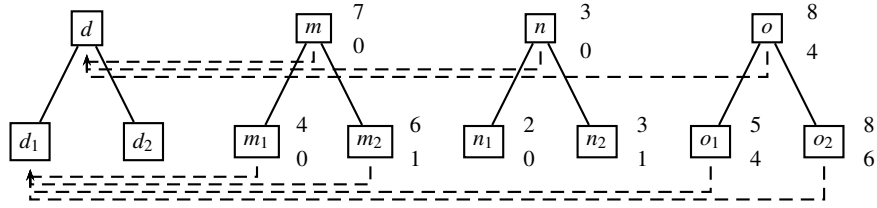


Fig. 7 Supporter pruning: source nodes $\{m, n, o\}$ are among the possible supporters of domain-node l . Their upper and lower bounds (shown as numbers to the right of each node) are used to prune them. Here, the upper bound for n (3) is smaller than the maximal lower bound among supporters (4, from o): this implies the upper bound of n 's children contributions to l 's children (shown here for l_1) will not surpass the lower bound of o 's children. We can thus safely remove n from the supporters. Please see text for details.

d if $\bar{\mu}_d^l < \max_{j \in \mathcal{S}_d} \underline{\lambda}_d^j$. This allows us to keep the maximization over d in Eq. 11 manageable at any point. In practice less than 15 supporters are typically involved at any point of the computation, as also shown in Fig. 6.

5 Results - Application to Deformable Object Detection

To estimate the merit of BB we first compare with the mixtures-of-DPMs developed and distributed in [17]. We directly extend the Branch-and-Bound technique that we developed for a single DPM to deal with multiple scales and mixtures ('ORs') of DPMs [10, 51], by inserting all object hypotheses into the same queue. In the Cascaded Detection case we simply do a for-loop over scales and components.

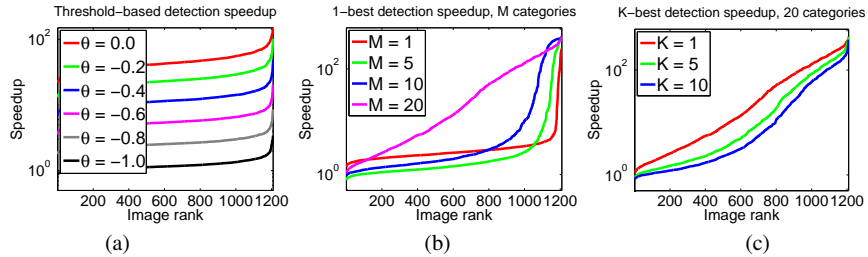


Fig. 8 (a) Single-object speedup of Cascaded Detection over GDTs on images from the Pascal dataset, (b,c) Multi-object speedup. Please see text for details.

Our technique delivers the same results as [17]: other than differences due to floating/double point arithmetic the results are identical. We therefore do not provide any detection performance curves, but only timing results.

Coming to time efficiency we compare the results of the original DPM mixture model and our implementation, using 1200 images from the Pascal dataset and the models of [17] for all 20 object categories. As a first experiment we consider the standard detection scenario where we want to detect all objects in an image that score above a certain threshold. We show in Fig. 8 (a) how the threshold affects the speedup we obtain: for a conservative threshold the speedup is typically tenfold, but as we become more aggressive it doubles.

As a second application, we consider the problem of identifying the ‘dominant’ object present in the image, i.e. the category that gives the largest score. Typically simpler models, like bag-of-words classifiers are applied to this problem, based on the understanding that part-based models can be time-consuming, therefore applying a large set of models to an image would be impractical.

Our claim is that Branch-and-Bound allows us to pursue a different approach, where in fact having more object categories can *increase* the speed of detection, if we leave the unary potential computation aside. Specifically, our approach can be directly extended to the multiple-object detection setting; as long as the scores computed by different object categories are commensurate, they can all be inserted in the same priority queue. In our experiments we observed that we can get a response with less computation per model by introducing more models. The reason for this is that including into our object repertoire a model giving a large score helps BB stop; otherwise BB keeps searching for another object.

The plots in Fig. 8 (b),(c) show systematic results for this experiment on the Pascal dataset. We compare the time that would be required by GDT to perform detection of all multiple objects considered in Pascal, to that of a model simultaneously exploring all categories. In (b) we show how finding the first-best result is accelerated as the number of objects (M) increases; while in (c) we show how increasing the ‘ k ’ in ‘ k -best’ affects the speedup. For small values of k the gains become more pronounced. Of course if we use Cascaded Detection the speedup does not change for multiple categories when compared to plot (a), since essentially the objects do not ‘interact’ in any way (we do not use nonmaximum suppression). But as we turn to the best-first problem, the speedup becomes dramatic, and can often be more than 100-fold.

We note that the timings in these plots refer to the ‘message passing’ part implemented with GDT and not the computation of unary potentials, which is common for both models, or the KD-tree construction, which is linear in the image size.

A more thorough breakdown of all costs can be found in Table 2. These results are obtained by summing over all 20 categories, and averaging over 1200 images from the Pascal VOC dataset; we report mean and standard deviation.

The first two rows report the cost of computing unary terms - these costs are common to the two methods being compared. The first method uses the BLAS-accelerated implementation of inner products provided in [17], while the second

	Our work	GDT-DPMs [17]
Unary terms (BLAS)	23.20 ± 1.49	23.20 ± 1.49
Unary terms (FFT) [7]	9.20 ± 1.21	9.20 ± 1.21
KD-trees	1.72 ± 0.21	0.00 ± 0.00
Detection, $\theta = 0.0$	0.25 ± 0.07	10.74 ± 1.02
Detection, $\theta = -.2$	0.47 ± 0.12	10.74 ± 1.02
Detection, $\theta = -.4$	0.93 ± 0.22	10.74 ± 1.02
Detection, $\theta = -.6$	1.95 ± 0.42	10.74 ± 1.02
Detection, $\theta = -.8$	4.17 ± 0.84	10.74 ± 1.02
Detection, $\theta = -1$	9.14 ± 1.79	10.74 ± 1.02
Detection, 1-best	0.41 ± 0.08	10.74 ± 1.02
Detection, 5-best	0.47 ± 0.09	10.74 ± 1.02
Detection, 10-best	0.48 ± 0.10	10.74 ± 1.02

Table 2 Timings for the treatment of all 20 categories per image on a single core, in seconds. Please see text for details

method uses the FFT in conjunction with the patchwork technique of [7] in order to accelerate computations. The method of [7] has a clear advantage.

The next row reports the time required to construct the KD-trees for the part and root intervals, alongside with the associated fine-to-coarse max-/min-imization operations; these are unique to our method, and of linear complexity, but we observe that their overall cost is negligible with respect to the overall computation costs.

The next six rows compare the cost of Cascaded Detection for a range of thresholds, with the linear-complexity GDT¹. The last three rows compare the cost of Branch-and-Bound for K-best detection with GDT.

Apart from the clear relative improvements with respect to GDTs, we observe that by combining the FFT-based unary term computation with our Branch-and-Bound implementation of DPM inference we can detect 20 objects in potentially less than ten seconds per image; these computation costs can be easily reduced further with multi-threaded computation and of course also by porting part of the computation to GPUs.

We are working on incorporating such aspects into our existing implementation, which is available from <http://cvn.ecp.fr/personnel/iasonas/dpms.html>

6 Conclusions and Discussion

In this work we have introduced Dual-Tree Branch-and-Bound for efficient part-based detection. We have used Dual Trees to compute upper bounds on the cost function of a part-based model and thereby derived Branch-and-Bound and Cascaded Detection algorithms for detection. Our algorithm is exact and makes no approximations, delivering identical results with the DPMs used in [10], but substan-

¹ In our comparisons we use the original- and faster- GDT algorithm of [11] instead of the one provided in [17]

tially smaller time. Further, we have shown that the flexibility of prioritized search allows us to consider new tasks, such as multiple-object detection, which yielded speedups by two orders of magnitude or more in certain cases.

The work presented in this Chapter focuses on the combinatorial optimization problem related to the ‘assembly’ of a deformable object from its parts; we have thus only treated one of the many aspects of deformable part modelling. In parallel works we have been exploring complementary aspects, including (i) the acceleration of the part score computations [25, 26], see also [40, 46, 7, 44, 6, 42, 41] (ii) extensions of Dual-Tree Branch-and-Bound to 3D [3] (iii) the treatment of better training criteria and richer graph topologies for shape segmentation with DPMs [2] (iv) the incorporation of segmentation information in DPMs [45] and, most recently (v) the use of Deep Convolutional Neural Network (DCNN) features in DPMs [37, 38] - see also [16, 48] for parallel works. These advances hint at the breadth of problems accommodated by DPMs, and shape modelling in general. We anticipate that properly treating the combinatorial structure of the optimization problem underlying DPMs will help further fuel progress across all these problems.

Acknowledgements I thank the two anonymous reviewers and Stefan Kinauer for feedback that helped improve this manuscript. I am grateful to the authors of [10, 29, 30, 22, 7] for making their code available. This work was funded by grants ANR-10-JCJC-0205 and FP7-Reconfig.

References

1. Belongie, S., Malik, J., Puzicha, J.: Shape Matching and Object Recognition Using Shape Contexts. *IEEE Trans. PAMI* **24**(4), 509–522 (2002)
2. Boussaid, H., Kokkinos, I.: Fast and exact: Admm-based discriminative shape segmentation with loopy part models. In: *CVPR* (2014)
3. Boussaid, H., Kokkinos, I., Paragios, N.: Rapid Mode Estimation for 3D Brain MRI Tumor Segmentation. In: *Energy Minimization Methods in Computer Vision and Pattern Recognition*. Lund, Sweden (2013)
4. Chen, Y., Zhu, L., Lin, C., Yuille, A.L., Zhang, H.: Rapid inference on a novel AND/OR graph for object detection, segmentation and parsing. In: *Proc. NIPS* (2007)
5. Dalal, N., Triggs, B.: Histograms of oriented gradients for human detection. In: *Proc. CVPR* (2005)
6. Dean, T., Ruzon, M., Segal, M., Shlens, J., Vijayanarasimhan, S., Yagnik, J.: Fast, accurate detection of 100,000 object classes on a single machine. In: *Proc. CVPR* (2013)
7. Dubout, C., Fleuret, F.: Exact acceleration of linear object detectors. In: *ECCV* (3) (2012)
8. Felzenszwalb, P., Huttenlocher, D.: Pictorial Structures for Object Recognition. *Int'l Journal of Computer Vision* **61**(1), 55–79 (2005)
9. Felzenszwalb, P., McAllester, D., Ramanan, D.: A discriminatively trained, multiscale, deformable part model. In: *Proc. CVPR* (2008)
10. Felzenszwalb, P.F., Girshick, R.B., McAllester, D.A.: Cascade object detection with deformable part models. In: *Proc. CVPR* (2010)
11. Felzenszwalb, P.F., Huttenlocher, D.P.: Distance transforms of sampled functions. Tech. rep., Cornell CS (2004)
12. Fergus, R., Perona, P., Zisserman, A.: Object class recognition by unsupervised scale-invariant learning. In: *Proc. CVPR* (2003)

13. Ferrari, V., Marin-Jimenez, M.J., Zisserman, A.: Progressive search space reduction for human pose estimation. In: Proc. CVPR (2008)
14. Fleuret, F., Geman, D.: Coarse-to-fine face detection. *Int.l Journal of Computer Vision* (2001)
15. Girshick, R., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. In: Proc. CVPR (2014)
16. Girshick, R., Iandola, F., Darrell, T., Malik, J.: Deformable part models are convolutional neural networks. arXiv preprint arXiv:1409.5403 (2014)
17. Girshick, R.B., Felzenszwalb, P.F., McAllester, D.: Discriminatively trained deformable part models, release 5. <http://people.cs.uchicago.edu/~rbg/latent-release5/>
18. Gray, A.G., Moore, A.W.: Nonparametric density estimation: Toward computational tractability. In: SIAM International Conference on Data Mining (2003)
19. Grimson, W.E.L.: *Object Recognition by Computer*. MIT Press (1991)
20. Huttenlocher, D., Klanderman, G., W., Rucklidge: Comparing Images Using the Hausdorff Distance. *IEEE Trans. PAMI* **15**(9), 850–863 (1993)
21. Ihler, A., Sudderth, E., Freeman, W., Willsky, A.: Efficient Multiscale Sampling from Products of Gaussian Mixtures. In: Proc. NIPS (2003)
22. Ihler, A., Sudderth, E., Freeman, W., Willsky, A.: Efficient Sampling of Gaussian Distributions. In: Proc. NIPS (2004)
23. Jordan, M.: Graphical Models. *Statistical Science* **19**, 140–155 (2004)
24. Kokkinos, I.: Rapid deformable object detection using dual-tree branch-and-bound. In: Proc. NIPS (2011)
25. Kokkinos, I.: Bounding part scores for rapid detection with deformable part models. In: 2nd Parts and Attributes Workshop, in conjunction with ECCV 2012 (2012)
26. Kokkinos, I.: Shufflets: Shared Mid-level Parts for Fast Multi-Category Detection. In: ICCV - International Conference on Computer Vision (2013)
27. Kokkinos, I., Yuille, A.: Inference and Learning with Hierarchical Shape Models. *Int.l Journal of Computer Vision* **93**, 201–225 (2011)
28. Krizhevsky, A., Sutskever, I., G., Hinton: ImageNet Classification with Deep Convolutional Neural Networks. In: Proc. NIPS (2012)
29. Lampert, C., Blaschko, M., Hofmann, T.: Beyond sliding windows: Object localization by efficient subwindow search. In: Proc. CVPR (2008)
30. Lampert, C.H.: An efficient divide-and-conquer cascade for nonlinear object detection. In: Proc. CVPR (2010)
31. Lehmann, A., Leibe, B., Gool, L.V.: Fast PRISM: Branch and Bound Hough Transform for Object Class Detection. *International Journal of Computer Vision* **94**(2), 175–197 (2011)
32. Lempitsky, V., Blake, A., Rother, C.: Image segmentation by branch-and-mincut. In: Proc. ECCV (2008)
33. Lowe, D.: *Perceptual Organization and Visual Recognition*. Kluwer (1985)
34. Lowe, D.: Object Recognition from Local Scale-Invariant Features. In: Proc. ICCV (1999)
35. Moreels, P., Maire, M., Perona, P.: Recognition by probabilistic hypothesis construction. In: Proc. ECCV, p. 55 (2004)
36. Mundy, J.L., Zisserman, A. (eds.): *Geometric invariance in computer vision*. MIT Press, Cambridge, MA, USA (1992)
37. P.-A. Savalle and S. Tsogkas and G. Papandreou and I. Kokkinos: Deformable part models with cnn features. In: 3rd Parts and Attributes Workshop, ECCV. (2014)
38. Papandreou, G., Kokkinos, I., Savalle, P.A.: Untangling local and global deformations in deep convolutional networks for image classification and sliding window detection. arXiv (2014)
39. Pedersoli, M., Vedaldi, A., González, J.: A coarse-to-fine approach for fast deformable object detection. In: Proc. CVPR (2011)
40. Pirsivash, H., Ramanan, D.: Steerable part models. In: CVPR (2012)
41. Sadeghi, M.A., Forsyth, D.A.: Fast template evaluation with vector quantization. In: NIPS (2013)
42. Sadeghi, M.A., Forsyth, D.A.: 30hz object detection with dpm v5. In: ECCV (2014)
43. Sapp, B., Toshev, A., Taskar, B.: Cascaded models for articulated pose estimation. In: Proc. ECCV (2010)

44. Song, H.O., Zickler, S., Althoff, T., Girshick, R.B., Fritz, M., Geyer, C., Felzenszwalb, P.F., Darrell, T.: Sparselet models for efficient multiclass object detection. In: Proc. ECCV (2012)
45. Trulls, E., Tsogkas, S., Kokkinos, I., Sanfeliu, A., Moreno-Noguer, F.: Segmentation-aware deformable part models. In: 2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014, pp. 168–175 (2014)
46. Vedaldi, A., Zisserman, A.: Sparse kernel approximations for efficient classification and detection. In: Proc. CVPR (2012)
47. Viola, P., Jones, M.: Rapid object detection using a boosted cascade of simple features. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2001)
48. Wan, L., Eigen, D., Fergus, R.: End-to-end integration of a convolutional network, deformable parts model and non-maximum suppression. arXiv (2014)
49. Weber, M., Welling, M., Perona, P.: Unsupervised learning of models for recognition. In: Proc. ECCV (2000)
50. Yang, Y., Ramanan, D.: Articulated human detection with flexible mixtures of parts. *IEEE Trans. Pattern Anal. Mach. Intell.* **35**(12), 2878–2890 (2013)
51. Zhu, S.C., Mumford, D.: Quest for a Stochastic Grammar of Images. *Foundations and Trends in Computer Graphics and Vision* **2**(4), 259–362 (2007)
52. Zhu, X., Ramanan, D.: Face detection, pose estimation, and landmark localization in the wild. In: 2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, June 16-21, 2012, pp. 2879–2886 (2012)
53. Zisserman, A., Forsyth, D.A., Mundy, J.L., Rothwell, C.A., Liu, J., Pillow, N.: 3D object recognition using invariance. *Artificial Intelligence* **78**, 239–288 (1995)

Index

- Branch-and-Bound, 8
- Cascaded Detection, 9
- Complexity, 5
- Deformable Models, 2
- Deformable Part Models, 2, 4
- Dual Recursion, 14
- Dual Trees, 11
- KD Trees, 11
- Object detection, 17
- PASCAL dataset, 17
- Shape recognition, 1
- Star-shaped DPMs, 4
- Supporter Pruning, 14
- Upper bounds to DPMs, 10