Rapid Deformable Object Detection using Bounding-based Techniques

Iasonas Kokkinos

Rapid Deformable Object Detection using Bounding-based Techniques

Iasonas Kokkinos

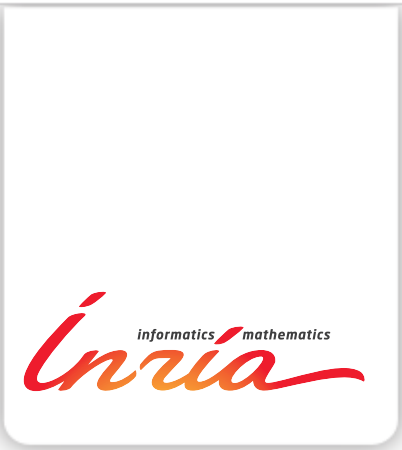
RESEARCH REPORT

N° 7940

April 2012

Project-Teams GALEN

ISSN 0249-6399 ISRN INRIA/RR--7940--FR+ENG



Rapid Deformable Object Detection using Bounding-based Techniques

Iasonas Kokkinos*

Project-Teams GALEN

Research Report n° 7940 — April 2012 — 19 pages

Abstract: In this work we use bounding-based techniques, such as Branch-and-Bound (BB) and Cascaded Detection (CD) to efficiently detect objects with Deformable Part Models (DPMs). Instead of evaluating the classifier score exhaustively over all image locations and scales, we use bounding to focus on promising image locations. The core problem is to compute bounds that accommodate part deformations; for this we adapt the Dual Trees data structure of [8] to our problem. We evaluate our approach using the DPM models of [4]. We obtain exactly the same results but can perform the part combination substantially faster; for a conservative threshold the speedup can be double, for a less conservative we can have tenfold or higher speedups. These speedups refer to the part combination process, after the unary part scores have been computed.

We also develop a multiple-object detection variation of the system, where hypotheses for 20 categories are inserted in a common priority queue. For the problem of finding the strongest category in an image this can result in more than 100-fold speedups.

Key-words: Branch-and-bound, Deformable part models

* École Centrale Paris, France and INRIA-Saclay, France.

RESEARCH CENTRE
SACLAY – ÎLE-DE-FRANCE

Parc Orsay Université
4 rue Jacques Monod
91893 Orsay Cedex

Résumé : Dans ce travail, nous utilisons des techniques qui utilisent des bornes, comme 'Branch-and-Bound' (BB) et 'Cascaded Detection' (CD) pour détecter efficacement des objets avec des modèles de pièces déformables. Au lieu d'évaluer le score classificateur d'une manière exhaustive sur tous les emplacements d'images et toutes échelles, nous utilisons BB qfine qde se concentrer sur les endroits prometteuses. Le coeur du problème est de calculer des bornes qui peuvent accueillir des déformations de pièces; pour cela nous avons adapté la structure de données 'Dual Tree' de [8] à notre problème. Nous évaluons notre approche à l'aide des les modèles de pièces déformables de [4]. Nous obtenons exactement les mêmes résultats, mais effectuons la combinaison de la pièce 10-20 fois plus rapide en moyenne. Nous développons aussi une variation de la détection de plusieurs objets du système, où les hypothèses pour 20 catégories sont insérés dans une commune file d'attente prioritaire. Pour le problème de trouver la plus forte catégorie dans une image il peut en résulter une accélération de 100 fois.

Mots-clés : Branch-and-bound, modèles de pièces déformables

1 Introduction

Deformable Part Models (DPMs) deliver state-of-the-art object detection results [4] on challenging benchmarks when trained discriminatively, and have become a standard in object recognition research. At the heart of these models lies the optimization of a merit function -the classifier score- with respect to the part displacements and the global object pose. In this work we take the classifier for granted, using the models of [4], 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 [5], 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 [16], [17] respectively. For this we exploit and adapt the Dual Tree (DT) data structure of [8] to provide the bounds required by BB/CD.

Our method is fairly generic; it applies to any star-shape 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 [4]. 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 reports provides a more extensive and updated presentation of the technique presented in [13]. The main differences are that (a) the presentation of our method has changed, opting for a 'hierarchical' description rather than the 'linear' description of [13]. (b) we now consider also Cascaded Detection, while in [13] we only used Branch-and-Bound. The former is slightly more efficient for detection with a fixed threshold, as it avoids the use of heap data structures which BB requires, and also lends itself to parallelization. (c) We provide a tighter lower bound for supporter pruning, which accelerates detection by roughly 10-20%. (d) we have revised the experimental results, after noticing that we were originally comparing to a slower variant of the algorithm in [4], but also after improving our own algorithm. In the end the results stay roughly the same.

The report is structured as follows: after briefly covering prior work in Sec. 2, in Sec. 3 we first describe the cost function used in DPMs, and then motivate the use of bounding-based techniques for efficient object detection. In Sec. 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 Sec. 4.3 we describe how we bound the DPM score and in Sec. 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 Sec. 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 [23] and coarse-to-fine detection [7] 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 [9], while [20] used A^* for detecting object instances. For categories, recent works [1, 14, 15, 22, 6, 21, 18] 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 [4] 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 -but controllable- 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 [10], 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 [16], where an upper bound of a bag-of-words classifier score delivers the bounds required by BB. Later [19] combined Graph-Cuts with BB for object segmentation, while in [17] 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 [19, 16, 17, 18], we use the DPM cost and thereby accommodate parts in a rigorous energy minimization framework. And unlike the pruning-based works [1, 6, 4, 21], 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 [8], original 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 [11] 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.

3 Object Detection with DPMs

3.1 DPM score function

We consider a star-shaped graphical model for objects consisting of $P+1$ nodes $\{n_0, \dots n_P\}$; n_0 is called the root and $n_1, \dots, n_P$ are the part nodes. Each node p comes with a unary observation potential $U_p(x)$, indicating the fidelity of the image at x to the local model for node p. For instance in [2] $U_p(x) = \langle w_p, H(x)\rangle$ is the inner product of a HOG feature $H(x)$ at x with a discriminant w_p for p. In this work we consider that $U_p(x)$ have been computed [1], and focus on the task of efficiently combining these unary measurements into a global decision that respects the pairwise constraints among parts.

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:

$$B_p(x', x) = -\left(x' - x - \mu_p\right)^T I_p \left(x' - x - \mu_p\right), \tag{1}$$

where $I_p = \mathrm{diag}(H_p, V_p)$ is a diagonal 'precision' matrix, μ_p is the nominal relative location vector, and we consider:

$$B_0(x', x) = \begin{cases} -\infty, & x' \neq x \\ 0, & x' = x \end{cases} \tag{2}$$

for convenience. We can view the expression in Eq. 1 as related to the log-likelihood of the relative locations under a diagonal-covariance Gaussian model.

All parts are connected exclusively with the root node, i.e. we have a star-shaped graphical model. If the root node is placed at x, the merit for a part p being placed at x' is given by $m_p(x', x) = U_p(x') +$

[1] A practical concern is that the computation of $U_p(x)$ typically takes substantially longer than their GDT-based combination. In our on-going research we have built on the work reported here to largely sidestep the unary part computation. Other relevant works include [21, 17]. See also the discussion at the end of the experimental results section.

$B_p(x', x)$. The score of a candidate object configuration $\mathbf{x} = (x_0, \ldots, x_P)$ is obtained by summing over the part merits:

$$M(\mathbf{x}) = \sum_{p=1}^{P} m_p(x_p, x_0). \tag{3}$$

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:

$$\begin{aligned} S(x) &\doteq \max_{\mathbf{x}: x_0 = x} M(\mathbf{x}) && (4)\\ &= \max_{\mathbf{x}: x_0 = x} \sum_{p=1}^{P} m_p(x_p, x) && (5)\\ &= \sum_{p=1}^{P} \max_{x_p} m_p(x_p, x) && (6)\\ &= \sum_{p=1}^{P} m_p(x), \quad \text{where} \quad m_p(x) \doteq \max_{x'} U_p(x') + B_p(x', x). && (7) \end{aligned}$$

To go from Eq. 4 to Eq. 5 we use Eq. 3, to go from Eq. 5 to Eq. 6 we use the distributive property and in Eq. 7 we introduce the notation for the 'messages' being sent from the part nodes to the root node. The part-to-root message passing described by Eq. 7 is identical to the leaf-to-parent message passing equations of the Max-Product algorithm [12] if we use the logarithm of the probabilities.

3.2 Object Detection

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 the latent variables. Case (b) corresponds to the common setup for detection, where all image positions scoring above a threshold are used as object hypotheses.

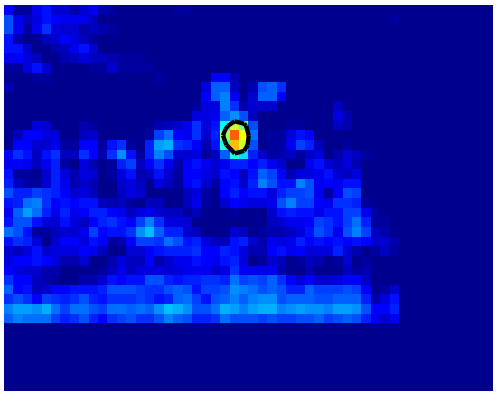
A naive approach to solve both of those cases is to consider all possible values of x, evaluate $S(x)$ on them and then recover the solutions. The complexity of this would be $O(PN^2)$, where $N = |\{x\}|$ is the cardinality of the set of possible locations considered (Eq. 7 suggests doing N maximizations per point, and we have N points and P parts).

But due to the particular form of the pairwise term, the maximization within each summand $m_p(x)$ in Eq. 7 lends itself to efficient computation in batch mode for all values of x using a Generalized Distance Transform (GDT) [5], 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, the N factor can still slow things down for large images.

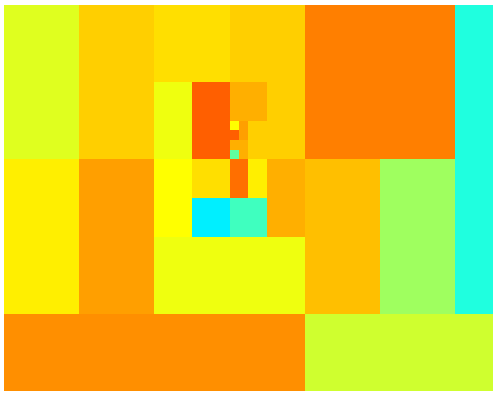
4 Bounding-based Detection with DPMs

Coming to how we can accelerate detection, our basic observation is that if we use a fixed threshold for detection, e.g. -1 for an SVM classifier, then the GDT-based approach 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

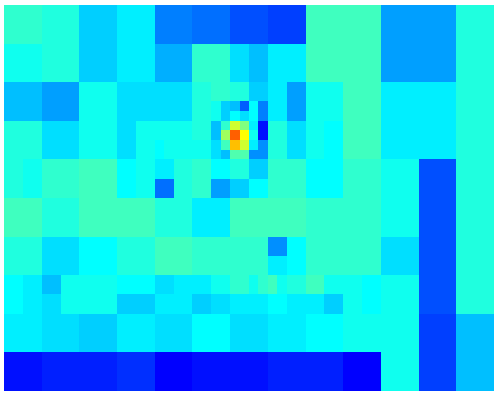
(a) Input & Detection result



(b) Detector score $S(x)$



(c) Brand and Bound for $\{\arg\max_x S(x)\}$



(d) Cascaded Detection for $\{x : S(x) \geq -1\}$.

Figure 1: 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 be positive-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.

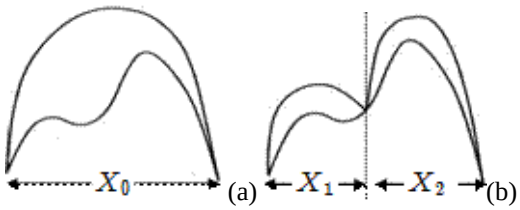


Figure 2: Illustration of how BB proceeds to maximize a complex, non-concave function within an interval by branching and bounding the function within intervals. Please see text for details.

threshold. This is illustrated in Fig. 1: in (a) we show the part-root configuration that gives the maximum score, and in (b) the score of a bicycle model from [4] 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 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. 1(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 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 Sec. 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. 2: 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 the Fig. 2 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.

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

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:

$$S(X) \doteq \max_{x \in X} S(x), \tag{8}$$

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

$$\overline{S}(X) \geq S(X) = \max_{x \in X} S(x) \quad \forall X, \quad \overline{S}(\{x\}) = S(x), \tag{9}$$

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 x, since the upper bound of their scores is below the score of $\overline{S}(\{x\}) = S(x)$.

4.2 Threshold-based detection: Cascaded Detection

The BB algorithm described above is appropriate when 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 [13] 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 [17], 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 [17] 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. 7 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:

$$\overline{S}(X) \doteq \sum_p \overline{m}_p(X) \geq \sum_p m_p(X) = \sum_p \max_{x \in X} m_p(x) \geq \max_{x \in X} \sum_p m_p(x) = S(X), \tag{10}$$

where $\overline{m}_p(X)$ 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 $\overline{m}_p(X)$ 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. 7.

As we will focus on the individual summands $\overline{m}_p(X)$, we omit the p subscript. Based on Eq. 7, $\overline{m}(X)$ should satisfy:

$$\overline{m}(X) \geq m(X) \overset{Eq.\ 8}{=} \max_{x \in X} m(x) \overset{Eq.\ 7}{=} \max_{x \in X} \left[\max_{x' \in X'} m(x', x) \right], \tag{11}$$

where X and X' do not need to be identical (by the definition of Eq. 7 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 Paritioning

We decompose the computation of the upper bound in Eq. 11 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. 3. 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. 3, the 'domain' intervals-d could be the numbers and the 'source' intervals could be the letters.

For a given partition of X, X' we can rewrite $m(X)$ in Eq. 11 as:

$$m(X) = \max_d \max_{x \in X_d} \max_s \max_{x' \in X_s} m(x', x) = \max_d \max_s \mu_d^s, \quad \text{where} \tag{12}$$

$$\mu_d^s \doteq \max_{x \in X_d} \max_{x' \in X_s} m(x', x). \tag{13}$$

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 point-interval can have to any point within any letter-interval.

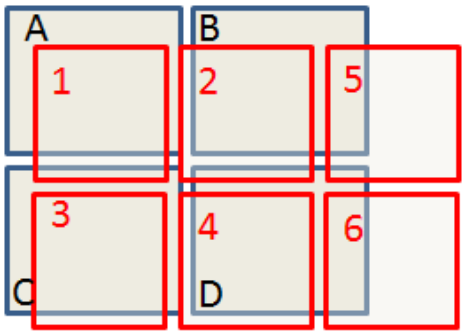


Figure 3: We rely on a partition of the 'source' (red) and 'domain' (blue) points to derive rapidly computable bounds of their 'interactions'. This could indicate for example that points lying in square 6 cannot have a large effect on points in square A, and therefore we do not need to go to a finer level of resolution to exactly estimate their interactions.

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. 5 and further detail in in Sec. 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.

4.3.2 Bounding the appearance and geometric terms

Based on Eq. 13 and the the definition of $m(x', x)$ we can upper bound μ_d^s as follows:

$$\mu_d^s = \max_{x \in X_d} \max_{x' \in X'_s} U(x') + B(x', x) \leq \max_{x' \in X'_s} U(x') + \max_{x \in X_d} \max_{x' \in X_d} B(x', x) \doteq \overline{\mu}_d^s, \tag{14}$$

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 Sec. 4.3.3, we also need to lower bound the quantity

$$\lambda_d^s = \min_{x \in X_d} \max_{x' \in X_s} U(x') + B(x', x). \tag{15}$$

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:

$$\underline{\lambda}_{d,1}^s = \max_{x' \in X_s} U(x') + \min_{x \in X_d} \min_{x' \in X_s} B(x', x) \leq \lambda_d^s \tag{16}$$

$$\underline{\lambda}_{d,2}^s = \min_{x' \in X'_s} U(x') + \min_{x \in X_d} \max_{x' \in X_s} B(x', x) \leq \lambda_d^s \tag{17}$$

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 bound into a single and tighter lower bound as:

$$\underline{\lambda}_d^s = \max\left(\underline{\lambda}_{d,1}^s, \underline{\lambda}_{d,2}^s\right). \tag{18}$$

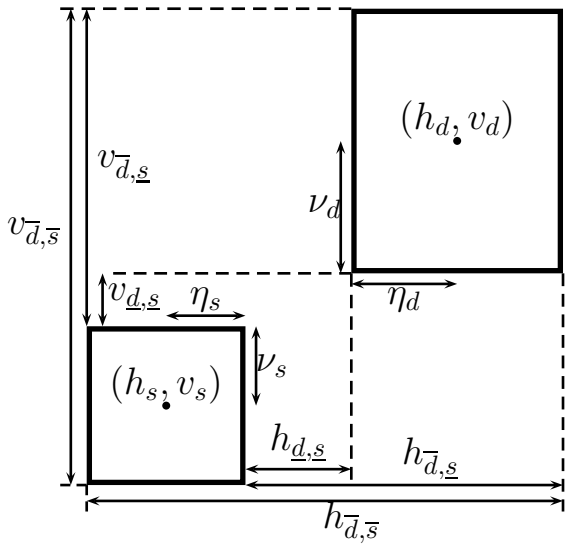


Figure 4: Illustration of the terms involved in the geometric bound computations of Eq.s 28-33. The d/s subscript indicates quantities relevant to the domain/source intervals respectively (we want to bound the score within the domain interval, using contributions from the source interval).

In [13] we had used only the first bound. Computing Eq. 18 requires some additional operations, but the bound is tighter and accelerates detection substantially.

We can rapidly compute the terms involved in the bounds of Eq.s 14–17. 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 horizonal and vertical coordinates, (h, v). Making reference to Fig. 4, 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 range of η_d/ν_d, while for X_s the respective quantities are $(h_s, v_s), \eta_s, \nu_s$. Using the (h, v) notation, we can write the pairwise term between two points, say $x \in X_d, x' \in X_s$ as:

$$\mathcal{G}_{x,x'} = -H(h - h')^2 - V(v - v')^2 \tag{19}$$

where H, V are the diagonal elements of the precision matrix showing up in Eq. 1; we omit the effect of the means μ in Eq. 1 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:

$$\begin{aligned}
\mathcal{G}_{\overline{d},\overline{s}} &\doteq \max_{x\in X_d}\max_{x'\in X_s}\mathcal{G}_{x,x'} && (20)\\
&= \max_{h\in X_d^h}\max_{h'\in X_s^h} -H(h-h')^2 + \max_{v\in X_d^v}\max_{v'\in X_s^v} -H(v-v')^2 && (21)\\
&= -Hh^2_{\underline{d},\underline{s}} - Vv^2_{\underline{d},\underline{s}}, \quad \text{where} && (22)\\
h_{\underline{d},\underline{s}} &\doteq \min_{h\in X_d^h}\min_{h'\in X_s^h}|h-h'|, \quad v_{\underline{d},\underline{s}} \doteq \min_{v\in X_d^v}\min_{v'\in X_s^c}|v-v'| && (23)
\end{aligned}$$

where we use $\overline{i}, \underline{i}$ to indicate respectively that we are max-/min-imizingwith 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

$$\begin{aligned}
\mathcal{G}_{\underline{d},\overline{s}} &\doteq \min_{x\in X_d}\max_{x'\in X_s}\mathcal{G}_{x,x'} = -Hh^2_{\overline{d},\underline{s}} - Vv^2_{\overline{d},\underline{s}}, \quad \text{where} && (24)\\
h_{\overline{d},\underline{s}} &\doteq \max_{h\in X_d^h}\min_{h'\in X_s^h}|h-h'|, \quad v_{\overline{d},\underline{s}} \doteq \max_{v\in X_d^v}\min_{v'\in X_s^v}|v-v'| && (25)
\end{aligned}$$

$$\begin{aligned}
\mathcal{G}_{\underline{d},\underline{s}} &\doteq \min_{x\in X_d}\min_{x'\in X_s}\mathcal{G}_{x,x'} = -Hh^2_{\overline{d},\overline{s}} - Vv^2_{\overline{d},\overline{s}}, \quad \text{where} && (26)\\
h_{\overline{d},\overline{s}} &\doteq \max_{h\in X_d^h}\max_{h'\in X_s^h}|h-h'|, \quad v_{\overline{d},\overline{s}} \doteq \max_{v\in X_d^v}\max_{v'\in X_s^v}|v-v'| && (27)
\end{aligned}$$

For the particular configuration shown in Fig. 4 we have:

$$\begin{aligned}
h_{\overline{d},\overline{s}} &= (h_d+\eta_d)-(h_s+\eta_s) && (28)\\
h_{\overline{d},\underline{s}} &= (h_d-\eta_d)-(h_s+\eta_s) && (29)\\
h_{\underline{d},\underline{s}} &= (h_d+\eta_d)-(h_s-\eta_s) && (30)\\
v_{\overline{d},\overline{s}} &= (v_d+\nu_d)-(v_s+\nu_s) && (31)\\
v_{\overline{d},\underline{s}} &= (v_d-\nu_d)-(v_s+\nu_s) && (32)\\
v_{\underline{d},\underline{s}} &= (v_d+\nu_d)-(v_s-\nu_s) && (33)
\end{aligned}$$

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

$$\begin{aligned}
h_{\overline{d},\overline{s}} &= \max\left(\lceil(h_d+\eta_d)-(h_s+\eta_s)\rceil, \lceil(h_s-\eta_s)-(h_d-\eta_d)\rceil\right) && (34)\\
&= \lceil|h_d-h_s|+(\eta_d-\eta_s)\rceil && (35)\\
h_{\overline{d},\underline{s}} &= \max\left(\lceil(h_d-\eta_d)-(h_s+\eta_s)\rceil, \lceil(h_s-\eta_s)-(h_d+\eta_d)\rceil\right) && (36)\\
&= \lceil|h_d-h_s|-(\eta_d+\eta_s)\rceil && (37)\\
h_{\underline{d},\underline{s}} &= \max\left(h_d+\eta_d)-(h_s-\eta_s), h_s+\eta_s)-(h_d-\eta_d)\rceil\right) && (38)\\
&= |h_d-h_s|+(\eta_d+\eta_s) && (39)
\end{aligned}$$

where $\lceil\cdot\rceil \doteq \max(\cdot,0)$; identical expressions are used for the vertical coordinate after substituting v,ν 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. 12. The range of d and s will scale inversely with the area of the intervals X_s, X_d, meaning that as the bounds get finer a larger

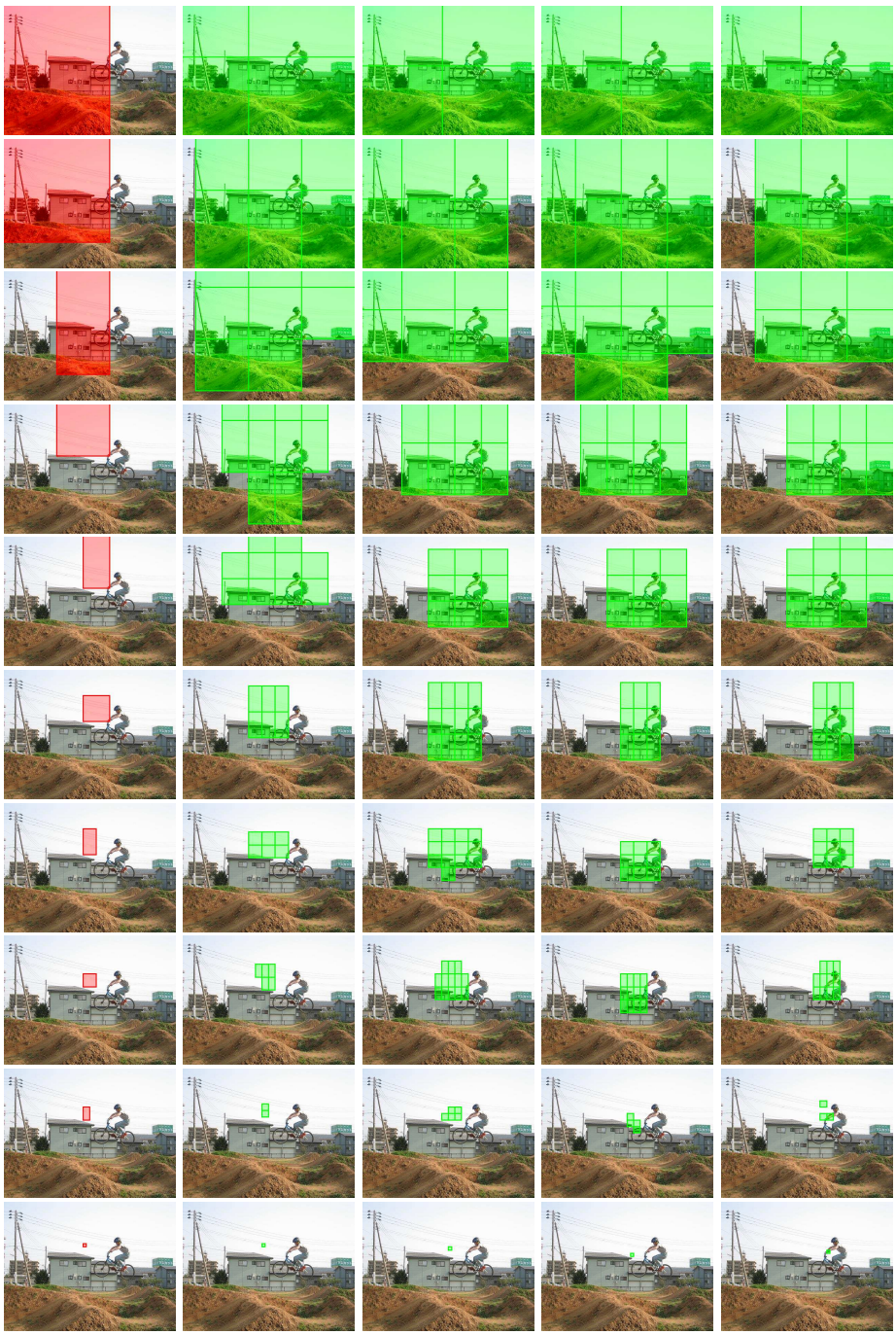


Figure 5: Illustration of supporter pruning. The left column illustrates the succession of domain intervals that leads to the optimal object configuration. The next four columns illustrate the associated 'supporters' of that interval for four distinct object parts. Our algorithm starts at the top with 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, making the overall optimization tractable. At the bottom row the part interval is a singleton, and is supporter by a single, and singleton, supporter interval. This indicates the optimal part placement for the given domain interval.

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 [8]. An illustration of how the algorithm works is provided in Fig. 5: 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. 12 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. $\mathcal{S}_k \subset \cup_{\mathcal{S}_{\mathrm{pa}(k)}}\{\mathrm{ch}(s_i)\}$, where pa, ch denote the parent and child operators. If any other points where necessary, these should have been included in the domain $\mathcal{S}_d$, by the definition of the 'supporter' intervals. Second, we can remove some elements of $\cup_{\mathcal{S}_{\mathrm{pa}(k)}}\{\mathrm{ch}(s_i)\}$ when forming $\mathcal{S}_k$, if we know that these cannot contribute to the optimal score at a domain node. This requires combining $\overline{\mu}_d^s$ and the lower bounds of $\underline{\lambda}_d^s$, and relies on the following rationale, illustrated in Fig. 6: consider that a node d has supporters l, m, o. If two nodes l and m support a node d and their bounds are related by $\overline{\mu}_d^l < \underline{\lambda}_d^m$, the descendants of interval l 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.

Concretely, denote by n_1, n_2 the two children of node n. We have that

$$\overline{\mu}_d^s \geq \mu_d^s \geq \mu_{d_j}^{s_i} \quad \forall i \in \{1,2\}, \forall j \in \{1,2\} \tag{40}$$

The first inequality holds from the fact that $\overline{\mu}$ upper bounds μ. The second inequality holds because according to Eq. 13, $\mu_d^s \doteq \max_{x \in X_s} \max_{x' \in D_s} m(x,x')$ while $X_s' \subset X_s, D_s' \subset D_s$; so maximizing a function over a smaller set will leader to a smaller quantity. In words, Eq. 40 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 $\overline{\mu}_d^s$ to the contribution of s to d.

We also have that:

$$\underline{\lambda}_d^s \leq \lambda_d^s \leq \lambda_{d_i}^s = \max(\lambda_{d_i}^{s_1}, \lambda_{d_i}^{s_2}), \quad i \in \{1,2\}. \tag{41}$$

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. 15, 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. 41 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 latter is guaranteed to get support, at its worst point, at least equal to $\underline{\lambda}_d^s$.

By putting Eq. 40 and Eq. 41 together we obtain that if $\overline{\mu}_d^l \leq \underline{\lambda}_d^n$ it follows that

$$\mu_{d_j}^{l_i} \leq \max(\lambda_{d_j}^{n_1}, \lambda_{d_j}^{n_2}), \quad j \in \{1,2\}, i \in \{1,2\} \tag{42}$$

This tells us that within the domain interval d_j any point will be getting a support from n_1, n_2 that will be at least as good as the best support it can get from l_1 or l_2. Therefore the intervals l_1, l_2 do not need to be considered anymore.

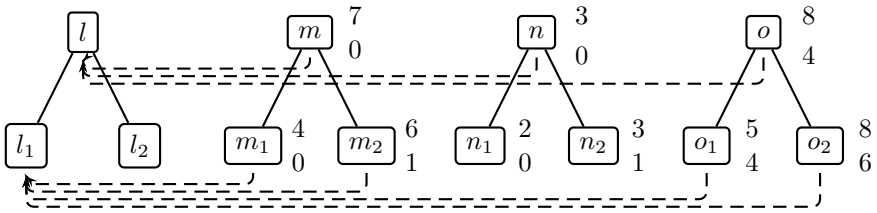


Figure 6: Supporter pruning: source nodes $\{m, n, o\}$ are among the possible supporters of domain-node l. Their upper and lower bounds (shown as numbers to the right of each node) are used to prune them. Here, the upper bound for n (3) is smaller than the maximal lower bound among supporters (4, from 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.

Concisely, we prune the children of supporter l to node d if $\overline{\mu}_d^l < \max_{j \in \mathcal{S}_d} \underline{\lambda}_d^j$. This allows us to keep the maximization over d in Eq. 12 manageable at any point. In practice less than 15 supporters are typically involved at any point of the computation, as also shown in Fig. 5.

4.4 Pseudocode

Pseudocode summarizing the Branch-and-Bound version of our algorithm is provided in Table 2. A minimal modification yields the Cascaded Detection variant. The algorithm uses a priority queue for Domain tree nodes, initialized with the root of the Domain tree (i.e. the whole range of possible locations x). At each iteration we pop a Domain tree node from the queue, compute upper bounds and supporters for its children, which are then pushed in the priority queue. The first leaf node that is popped contains the best domain location: its upper bound equals its lower bound, and all other nodes in the priority queue have smaller upper bounds, therefore cannot result in a better solution.

We note that each part has its own KDtree (SourcT[p]): we build a separate Source-tree per part using the part-specific coordinates (x^p) and weights $w_{p,i}$. Each part's contribution to the score is computed using the supporters it lends to the node; the total bound is obtained by summing the individual part bounds.

5 Results - Application to Deformable Object Detection

To estimate the merit of BB we first compare with the mixtures-of-DPMs developed and distributed by [3]. 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 [4, 24], by inserting all object hypotheses into the same queue. In the Cascaded Detection case we simply do a for-loop over scales and components.

Our technique delivers essentially the same results as [4]. 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 [4] for all 20 object categories. As a first experiment we consider the standard detection scenario where we want to detect all objects in

$X^* = \mathrm{DualTreeBranchandBound}(\mathrm{SourceTrees}, \mathrm{DomainTree})$

X_0.DomainNode = DomainTree.Root; {Setup 'root' domain node}
for part = 1 to NParts **do**
  X_0.Supporters[part] = { SourceTrees[part].Root };
end for
$\mathcal{Q}$ = Push({,},{X_0,∞}); {Initialization of priority queue}
while 1 **do**
  $X = \mathrm{Pop}(\mathcal{Q})$;
  if Singleton[X] **then**
    RETURN X;
  end if
  $[X_1, X_2] = \mathrm{Branch}[X]$ {Descend on the domain KD-tree}
  for i = 1:2 **do**
    $\mathrm{UB} = 0$;
    $\mathrm{DomainInterval} = \mathrm{DomainTree}.[X_i.\mathrm{DomainNode}]$ {Get the domain interval}
    for part = 1:P **do**
      $S = \mathrm{Branch}(X_i.\mathrm{Supporters}[\mathrm{part}])$ {Descend on the source KD-tree}
      $[\mathrm{UP}, \mathrm{S}'] = \mathrm{Prune}(\mathrm{DomainInterval}, \mathrm{S}, \mathrm{SourceTree}[\mathrm{p}])$;
      $X_i.\mathrm{Supporters}[\mathrm{part}] = \mathrm{S}'$;
      $\mathrm{UB} = \mathrm{UB} + \mathrm{UP}$;
    end for
    $\mathrm{Push}(\mathcal{Q}, \{X_i, UB\})$;
  end for
end while

Pruning Routine

$[\mathrm{UP}, \mathrm{S}'] = \mathrm{Prune}(\mathrm{DomainInterval}, \mathrm{S}, \mathrm{SourceTree})$;
for n $\in$ S **do**
  $\mathrm{UB}[n] = \mathrm{UpperBound}(\mathrm{DomainInterval}, \mathrm{SourceTree.node}[n])$; {$\overline{\mu}_d^n$}
  $\mathrm{LB}[n] = \mathrm{LowerBound}(\mathrm{DomainInterval}, \mathrm{SourceTree.node}[n])$; {$\underline{\lambda}_d^n$}
end for
$\mathrm{MLB} = \max_n \mathrm{LB}[n]$;
$S' = \{S_i : \mathrm{UB}[i] \geq \mathrm{MLB}\}$;
$\mathrm{UP} = \max_n \mathrm{UB}[n]$;

Table 2: Pseudocode for the Dual-Tree Branch-and-Bound algorithm.

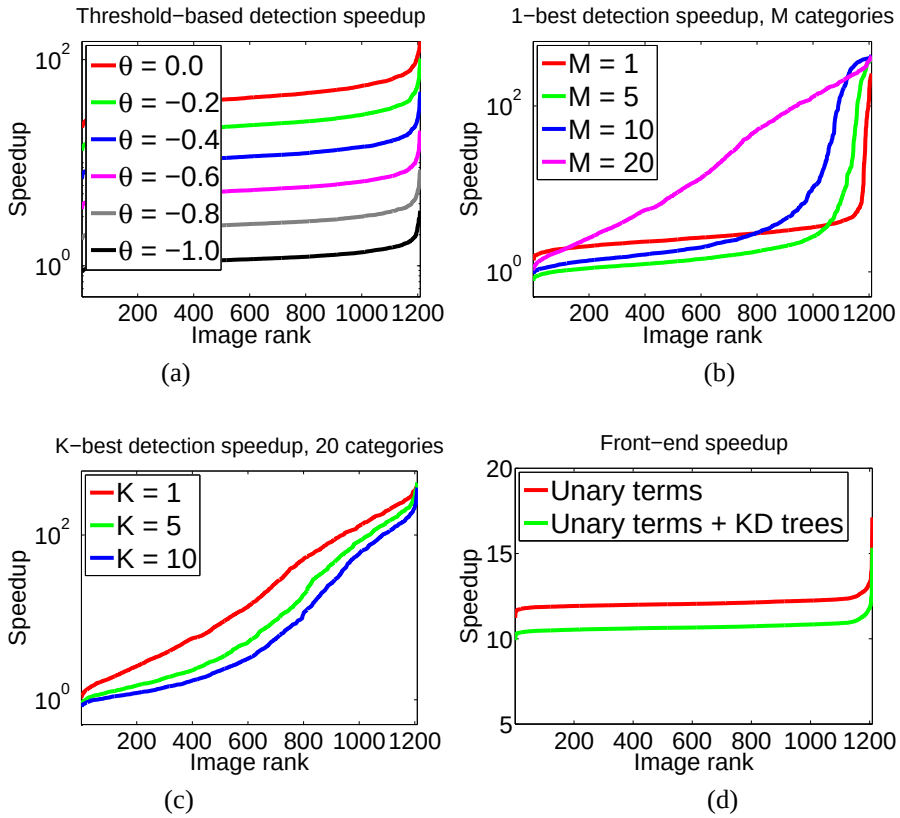


Figure 7: (a) Single-object speedup of Cascaded Detection over GDTs on images from the Pascal dataset, (b,c) Multi-object speedup. (d) Speedup due to the front-end computation of the unary potentials. Please see text for details.

	Our algorithm	[4]
Unary terms	13.20 ± 1.49	159.41 ± 15.82
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 3: Comparative timings, in seconds, of our approach compared to [4]. Please see text for details.

an image having score above a certain threshold. We show in Fig. 7 (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 the 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. In specific, 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 faster 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.

In plots Fig. 7 (b),(c) we 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 refer to the 'message passing' part implemented with GDT and not the computation of unary potentials, which is common for both models, and is currently the bottleneck, or the KD-tree construction, which is linear in the image size. Even though it is tangential to our contribution in this paper, we mention that as shown in plot (d) we compute unary potentials approximately five times faster than the single-threaded convolution provided by [3] by exploiting Matlab's optimized matrix multiplication routines.

A summary of our results can be found in Table 3, where we compare average timings our approach to the one of [4]. The 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 top two rows compare the front-end efficiency: in the first row we compare our single-threaded Matlab-based convolution code with the single-threaded, BLAS-free convolution of [4]. In the second row we report 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. The next six rows compare the cost of Cascaded Detection for a range of thresholds, with the linear-time GDT complexity. The last three rows compare the cost of Branch-and-Bound for K-best detection with GDT. In our comparisons we use the original- and faster- GDT algorithm of [5] instead of the one provided in [4].

6 Conclusions

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 [4], but substantially 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.

Our main ongoing research direction is to reduce the unary term computation cost, while the longer-term goal of our research is to scale up recognition to hundreds or even thousands of object categories.

7 Acknowledgements

We are grateful to the authors of [3, 16, 17, 11] for making their code available. This work was funded by grant ANR-10-JCJC -0205.

References

[1] Y. Chen, L. Zhu, C. Lin, A. L. Yuille, and H. Zhang. Rapid inference on a novel and/or graph for object detection, segmentation and parsing. In *NIPS*, 2007.

[2] P. Felzenszwalb, D. McAllester, and D. Ramanan. A discriminatively trained, multiscale, deformable part model. In *CVPR*, 2008.

[3] P. F. Felzenszwalb, R. B. Girshick, and D. McAllester. Discriminatively trained deformable part models, release 4. http://www.cs.brown.edu/ pff/latent-release4/.

[4] P. F. Felzenszwalb, R. B. Girshick, and D. A. McAllester. Cascade object detection with deformable part models. In *CVPR*, 2010.

[5] P. F. Felzenszwalb and D. P. Huttenlocher. Distance transforms of sampled functions. Technical report, Cornell CS, 2004.

[6] V. Ferrari, M. J. Marin-Jimenez, and A. Zisserman. Progressive search space reduction for human pose estimation. In *CVPR*, 2008.

[7] F. Fleuret and D. Geman. Coarse-to-fine face detection. *IJCV*, 2001.

[8] A. G. Gray and A. W. Moore. Nonparametric density estimation: Toward computational tractability. In *SIAM International Conference on Data Mining*, 2003.

[9] E. Grimson. *Object Recognition by Computer*. MIT Press, 1991.

[10] D. Huttenlocher, G. Klanderman, W., and Rucklidge. Comparing Images Using the Hausdorff Distance. *IEEE T. PAMI*, 15(9):850–863, 1993.

[11] A. T. Ihler, E. B. Sudderth, W. T. Freeman, and A. S. Willsky. Efficient multiscale sampling from products of gaussian mixtures. In *NIPS*, 2003.

[12] M. Jordan. Graphical Models. *Statistical Science*, 19:140–155, 2004.

[13] I. Kokkinos. Rapid deformable object detection using dual tree branch and bound. In *NIPS*, 2011.

[14] I. Kokkinos and A. Yuille. HOP: Hierarchical Object Parsing. In *CVPR*, 2009.

[15] I. Kokkinos and A. L. Yuille. Inference and learning with hierarchical shape models. *International Journal of Computer Vision*, 93(2):201–225, 2011.

[16] C. Lampert, M. Blaschko, and T. Hofmann. Beyond sliding windows: Object localization by efficient subwindow search. In *CVPR*, 2008.

[17] C. H. Lampert. An efficient divide-and-conquer cascade for nonlinear object detection. In *CVPR*, 2010.

[18] A. Lehmann, B. Leibe, and L. V. Gool. Fast PRISM: Branch and Bound Hough Transform for Object Class Detection. *International Journal of Computer Vision*, 94(2):175–197, 2011.

[19] V. Lempitsky, A. Blake, and C. Rother. Image segmentation by branch-and-mincut. In *ECCV*, 2008.

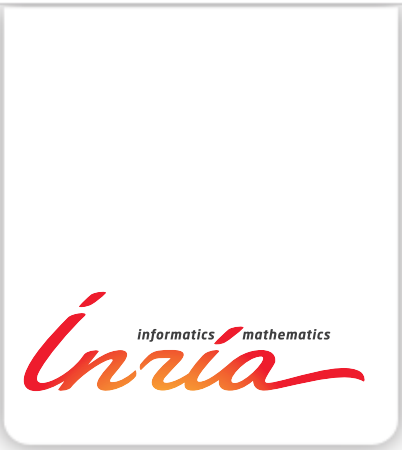
[20] P. Moreels, M. Maire, and P. Perona. Recognition by probabilistic hypothesis construction. In *ECCV*, page 55, 2004.

[21] M. Pedersoli, A. Vedaldi, and J. Gonzàlez. A coarse-to-fine approach for fast deformable object detection. In *CVPR*, 2011.

[22] B. Sapp, A. Toshev, and B. Taskar. Cascaded models for articulated pose estimation. In *ECCV*, 2010.

[23] P. Viola and M. Jones. Rapid Object Detection using a Boosted Cascade of Simple Features. In *CVPR*, 2001.

[24] S. C. Zhu and D. Mumford. Quest for a Stochastic Grammar of Images. *Foundations and Trends in Computer Graphics and Vision*, 2(4):259–362, 2007.



RESEARCH CENTRE
SACLAY – ÎLE-DE-FRANCE

Parc Orsay Université
4 rue Jacques Monod
91893 Orsay Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399