A heuristic in A* for inference in nonlinear Probabilistic Classifier Chains

Deiner Mena[a,b], José Ramón Quevedo[a], Elena Montañés[a,*], Juan José del Coz[a]

[a] *Artificial Intelligence Center. University of Oviedo at Gijón, 33204 Asturias, Spain* http://www.aic.uniovi.es

[b] *Dept. de Ingeniería en Telecomunicaciones e Informática, Universidad Tecnológica del Chocó, Quibdó - Chocó, Colombia* http://www.utch.edu.co

Abstract

Probabilistic Classifier Chains (PCC) is a very interesting method to cope with multi-label classification, since it is able to obtain the entire joint probability distribution of the labels. However, such probability distribution is obtained at the expense of a high computational cost. Several efforts have been made to overcome this pitfall, proposing different inference methods for estimating the probability distribution. Beam search and the ϵ- approximate algorithms are two methods of this kind. A more recently approach is based on the A* algorithm with an admissible heuristic, but it is limited to be used just for linear classifiers as base methods for PCC. This paper goes in that direction presenting an alternative admissible heuristic for the A* algorithm with two promising advantages in comparison to the above-mentioned heuristic, namely, i) it is more dominant for the same depth and, hence, it explores fewer nodes and ii) it is suitable for nonlinear classifiers. Additionally, the paper proposes an efficient implementation for the computation of the heuristic that reduces the number of models that must be evaluated by half. The experiments show, as theoretically expected, that this new algorithm reaches Bayes-optimal predictions in terms of subset 0/1 loss and explores fewer nodes than other state-of-the-art methods

*Corresponding author

Email addresses: deiner.mena@utch.edu.co (Deiner Mena), quevedo@uniovi.es (José Ramón Quevedo), montaneselena@uniovi.es (Elena Montañés), juanjo@uniovi.es (Juan José del Coz)

that also provide optimal predictions. In spite of exploring fewer nodes, this new algorithm is not as fast as the ϵ-approximate algorithm with $\epsilon = 0$ when the search for an optimal solution is highly directed. However, it shows its strengths when the datasets present more uncertainty, making faster predictions than other state-of-the-art approaches.

Keywords: Multilabel, Classifier Chains, Inference, Heuristic Search

---

1. Introduction

In Multi-label classification (MLC), a subset from a predefined set of labels may be assigned to an instance. Several real problems fall into this kind of learning task, for instance, tag assignment to resources in social networks, object detection in pictures or medical diagnosis.

Researchers in this field coincide in arguing that one interesting quality of multi-label learning is the dependence that may be concealed among labels. In fact, this dependence has been susceptible of being exploited by algorithms already available in the literature [2, 13, 14, 18, 22, 23, 24]. In the last years, Probabilistic Classifier Chains (PCC) have been gained interest due to its promising property of obtaining the entire joint probability. In particular, PCC methods exploit the conditional dependence, widely discussed in the literature [4], instead of the marginal or unconditional dependence.

The high computational cost of the original method [3], which performs an exhaustive search (ES), is the reason why researchers have been get down to work for designing inference algorithms able to estimate such entire joint conditional probability with an non-prohibitive computational cost. One of the forefathers of these inference methods is the ϵ-approximate algorithm (ϵ-A) [5], which uses uniform-cost search to obtain optimal Bayes predictions in terms of subset 0/1 loss considerably reducing the computational cost. Another alternative also available in the literature is beam search (BS) [8], whose reduction in computational time is remarkable, in spite of not guaranteeing optimal predictions in terms of subset 0/1 loss. Also Monte Carlo approaches [5, 16] take

the classifiers to draw samples of label combination susceptible of being an optimal prediction. Even more recently, the A* algorithm has also been proposed [10, 12], which includes an admissible heuristic that guarantees reaching an optimal solution in terms of subset 0/1 loss. Despite authors include a depth parameter in order to control the number of nodes explored in regard to the computational time spent in computing the heuristic, the use of this heuristic is limited to linear classifiers as base methods for PCC, except when the depth of the heuristic is 1, for which any base classifier with probabilistic output is allowed. But this is the extreme case of spending the least possible time in computing the heuristic, but at expenses of exploring the most number of nodes.

The main contribution of this paper is an alternative heuristic to the one proposed in [10, 12] for the A* algorithm that i) it is also admissible, hence, it guarantees Bayes-optimal predictions for the subset 0/1 loss, ii) it also includes a depth parameter to balance the number of nodes explored and the computational time of computing the heuristic, but iii) it overcomes the limitation of being applicable only when linear classifiers are chosen as base methods in PCC and iv) it is more dominant than the one introduced in [10, 12] for the same depth, hence, it theoretically explores fewer nodes. The second contribution is to present an efficient algorithm of the heuristic in order to optimize its computation. This algorithm significantly reduces the number of models that must be evaluated to compute the heuristic and outperforms the approach presented in [10, 12] both in terms of number of nodes explored and computational time. The experiments also confirm that the proposed method is competitive with respect to other approaches, especially for complex problems.

The rest of the paper is organized as follows. Section 2 formally states the multi-label problem and describes the PCC method. Section 3 gives a brief overview and discussion of some state-of-the-art algorithms to perform inference in PCC. Section 4 details the proposal of this paper. Finally, Sections 5 and 6 respectively show and discuss the experiments carried out and the conclusions.

2. Multi-label classification and Probabilistic Classifier Chains

Before describing the PCC algorithm and the inference methods, let us formally introduce the multi-label classification problem. Firstly, a non-empty and finite set of m labels $\mathcal{L} = \{\ell_1, \ell_2, \ldots, \ell_m\}$ is provided. Secondly, if $\mathcal{X}$ is the instance description space and $\mathcal{Y}$ is the power set of $\mathcal{L}$, a set of instances $S = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots, (\boldsymbol{x}_n, \boldsymbol{y}_n)\}$ is drawn over $\mathcal{X} \times \mathcal{Y}$ according to an unknown probability distribution $\mathbf{P}(\mathbf{X}, \mathbf{Y})$. Notice that the output space can be redefined as $\mathcal{Y} = \{0, 1\}^m$, since $\boldsymbol{y}_i$ is, in fact, a vector $\boldsymbol{y}_i = (y_{i,1}, y_{i,2}, \ldots, y_{i,m})$ where $y_{i,j} = 1$ means a positive relevance and $y_{i,j} = 0$ means that label ℓ_j is irrelevant for $\boldsymbol{x}_i$.

Under this framework, the goal of MLC is to obtain $\boldsymbol{f} : \mathcal{X} \longrightarrow \mathcal{Y}$ from S that minimize the risk in terms of a certain loss function $L(\cdot, \cdot)$. That risk can be defined as the expected loss over the entire joint distribution $\mathbf{P}(\mathbf{X}, \mathbf{Y})$, that is,

$$r_L(\boldsymbol{f}) = \mathbb{E}_{\mathbf{X},\mathbf{Y}} L(\mathbf{Y}, \boldsymbol{f}(\mathbf{X})). \tag{1}$$

This expression can be simplified if the conditional probability distribution $\mathbf{P}(\boldsymbol{y} \,|\, \boldsymbol{x})$ of $\mathbf{Y} = \boldsymbol{y}$ conditioned by $\mathbf{X} = \boldsymbol{x}$ is taken into account, obtaining the following expression

$$r_L(\boldsymbol{f}(\boldsymbol{x})) = \arg\min_{\boldsymbol{f}} \sum_{\boldsymbol{y} \in \mathcal{Y}} \mathbf{P}(\boldsymbol{y} \,|\, \boldsymbol{x}) \cdot L(\boldsymbol{y}, \boldsymbol{f}(\boldsymbol{x})). \tag{2}$$

Among the evaluation measures suitable for studying the performance of the algorithms designed for MLC, this paper focuses its attention to the subset 0/1 loss. This loss function is an instance-oriented measure that evaluates for each instance if the set of predicted labels coincides with the set of actual labels or not. Formally, it is defined as[1]

$$L_{S_{0/1}}(\boldsymbol{y}, \boldsymbol{f}(\boldsymbol{x})) = [\![\boldsymbol{y} \neq \boldsymbol{f}(\boldsymbol{x})]\!]. \tag{3}$$

Taking into account the subset 0/1 loss as loss function, the risk minimizer is

[1] $[\![p]\!]$ equals 1 if p is true and 0 otherwise

reduced to optimize the conditional joint distribution, that is,

$$r_{S_{0/1}}(\boldsymbol{f}(\boldsymbol{x})) = \arg\max_{\boldsymbol{y}\in\mathcal{Y}} \mathbf{P}(\boldsymbol{y} \mid \boldsymbol{x}). \tag{4}$$

This simplification of the risk minimizer for the subset 0/1 loss allows providing optimal predictions just considering all the possible combinations for the values of the labels. This task continues being of exponential order, and, as we will see later on, this is the reason why it is necessary to perform inference. But obtaining optimal predictions for a generic loss function $L(\cdot,\cdot)$ means to directly use Equation (2), which involves summing over an exponential number of label combinations for all f [6]. Hamming loss is another common measure in MLC, which is easier to optimize than subset 0/1. This loss is also an instance-oriented measure that evaluates if the predicted value for each label coincides with the actual value or not. Formally, it is defined as

$$L_H(\boldsymbol{y}, \boldsymbol{f}(\boldsymbol{x})) = \frac{1}{m}\sum_{i=1}^{m} [\![y_i \neq f_i(\boldsymbol{x})]\!]. \tag{5}$$

Hence, the risk minimizer for the Hamming loss measure is

$$r_H(\boldsymbol{f}(\boldsymbol{x})) = (r_{H,1}(f_1(\boldsymbol{x})), \ldots, r_{H,m}(f_m(\boldsymbol{x}))), \tag{6}$$

where

$$r_{H,i}(f_i(\boldsymbol{x})) = \arg\max_{v\in\{0,1\}} \mathbf{P}(y_i = v \mid \boldsymbol{x}). \tag{7}$$

Therefore, providing optimal predictions in terms of Hamming loss means to obtain optimal predictions for each label independently of the rest of labels [18], so just applying the well-known Binary Relevance approach one can optimize this measure and no inference is required. Hence, this paper just focuses on subset 0/1 loss, whose optimization requieres inference to obtain Bayes-optimal predictions.

Among the methods to cope with MLC, let us concentrate on PCC in what follows. The interest of this approach among researches is that, in addition of considering the conditional dependence among labels, it has the ability of optimally estimating the joint probability of a set of labels and, hence, the risk

minimizer for the subset 0/1 loss (see Equation (4)). Let us now explain the method in order to easily deduce that for this purpose it is just necessary to apply the product rule of probability.

First of all, PCC [3] establishes an order of the labels, defining a chain. Secondly and following this order, PCC trains a binary and probabilistic classifier for each label ℓ_j able to estimate the probability of ℓ_j of being relevant for a given instance $\boldsymbol{x}$ and taking into account the predictions for the previous labels in the chain, that is, PCC estimates $\mathbf{P}(y_j \,|\, \boldsymbol{x}, y_1, \ldots, y_{j-1})$. Hence, the binary probabilistic classifier that estimates the relevance of the label ℓ_j is of the form

$$f_j : \mathcal{X} \times \{0,1\}^{j-1} \longrightarrow [0,1]. \tag{8}$$

The training set employed to induce f_j is $S_j = \{(\overline{\boldsymbol{x}}_1, y_{1,j}), \ldots, (\overline{\boldsymbol{x}}_n, y_{n,j})\}$ where now the instance description is $\overline{\boldsymbol{x}}_i = (\boldsymbol{x_i}, y_{i,1}, \ldots, y_{i,j-1})$, that is, the original instance description $\boldsymbol{x}_i$ together with the relevance of the labels $\ell_1, \ldots, \ell_{j-1}$ placed before ℓ_j in the chain and the class $y_{i,j}$ is the relevance of ℓ_j.

Consequently, the task of evaluating the risk minimizer for the subset 0/1 loss is simplified to just finding a combination of labels that maximizes the expression obtained after applying the product rule of probability to the conditional joint probability, that is:

$$\mathbf{P}(\boldsymbol{y} \,|\, \boldsymbol{x}) = \prod_{j=1}^{m} \mathbf{P}(y_j \,|\, \boldsymbol{x}, y_1, \ldots, y_{j-1}). \tag{9}$$

Theoretically, this expression is valid for all possible orders of the labels, but in practice, those methods based on Classifier Chains (CC) [17] are label-order dependent and, besides, some of them also present other issues related to error propagation [20, 21]. These aspects are topics of interest among researchers but they are out of the scope of this paper. Hence, let us consider an unique order of the chain and concentrate on what follows in performing inference to provide Bayes-optimal predictions in a way that the computational cost may be reduced.

Before analyzing this issue in the next section, note that from a theoretical point of view, this expression holds for any order considered for the labels. In

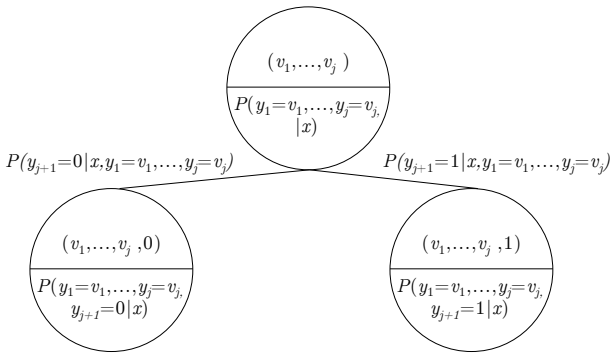


Figure 1: A generic node and its children of the probability binary tree. The top part of each node contains the combination of labels and the joint probability of such a combination. The edges are labeled with the conditional probability

any case, in this paper we assume the order of the labels in the chain to be given, since the goal is just to analyze the performance of the methods, without taking into account the effect of different orders. Hence, we do not include any study about which order can be the best.

Notice that performing inference in PCC can be seen as the different ways of exploring a probability binary tree (see Figure 1). In this tree, the root is labeled by the empty set of labels and a generic node of level $j<m$ is labeled by $(v_1, v_2, \ldots, v_j)$ with $v_i \in \{0,1\}$ for $i=1,\ldots,j$. This node has two children who are respectively labeled by $(v_1, v_2, \ldots, v_j, 0)$ and $(v_1, v_2, \ldots, v_j, 1)$ with a respective marginal joint conditional probability $\mathbf{P}(y_1=v_1, \ldots, y_j=v_j, y_{j+1}=0 \,|\, \boldsymbol{x})$ and $\mathbf{P}(y_1 = v_1, \ldots, y_j = v_j, y_{j+1} = 1 \,|\, \boldsymbol{x})$. The probability of the edges between the node and their children are respectively the conditional probabilities $\mathbf{P}(y_{j+1} = 0 \,|\, \boldsymbol{x}, y_1=v_1, \ldots, y_j=v_j)$ and $\mathbf{P}(y_{j+1}=1 \,|\, \boldsymbol{x}, y_1=v_1, \ldots, y_j=v_j)$ estimated using the trained classifiers by $1 - f_{j+1}(\boldsymbol{x}, v_1, \ldots, v_j)$ and $f_{j+1}(\boldsymbol{x}, v_1, \ldots, v_j)$. Finally, the marginal joint conditional probability of the children is estimated using the product rule of probability: $\mathbf{P}(y_1 = v_1, \ldots, y_j = v_j, y_{j+1} = v_{j+1} \,|\, \boldsymbol{x}) = \mathbf{P}(y_{j+1} = v_{j+1} \,|\, \boldsymbol{x}, y_1=v_1, \ldots, y_j=v_j) \cdot \mathbf{P}(y_1=v_1, \ldots, y_j=v_j \,|\, \boldsymbol{x})$.

3. Inference methods in PCC for multi-label classification

Several state-of-the-art methods for inference in PCC have been deeply studied so far (see [11] for an exhaustive review). Let us now briefly describe some of the most promising approaches, with which the method proposed in this paper will be compared later on.

3.1. Greedy Search

The so-called method CC [18] is the original inference method for PCC. It performs a Greedy Search (GS) over the probability binary tree, then it just explores one path. At each node, it follows the branch with the highest marginal joint conditional probability. Taking into account the product rule of probability, this branch coincides with that of the highest conditional probability estimated by f_j, since both branches among to choose are children of the same node. Each classifier f_j is successively feed by the labels previously predicted along the path. This method tries to optimize the subset 0/1 loss, but an analysis performed in this direction [5] have shown poor results and in general it does not reach an optimal solution.

3.2. ϵ-Approximate algorithm

The ϵ-*Approximate* (ϵ-A) algorithm [5] has promisingly arisen in between the high computational cost of obtaining the entire joint conditional probability by ES and the poor performance of GS. The method explores more than one path and it consists of expanding just the nodes whose marginal joint conditional probability exceeds the value of the threshold $\epsilon = 2^{-k}$, with $1 \leq k \leq m$. The values of this threshold determines the strategy of the algorithms. For instance, the particular case of $\epsilon = 0$ (or any value in the interval $[0, 2^{-m}]$, that is, $k{=}m$) means to perform a uniform-cost search (UC) that always reaches an optimal solution. Besides, the method goes towards GS as ϵ increases and equals GS when $\epsilon = 2^{-1} = 0.5$ $(k = 1)$. This method optimizes the subset 0/1 loss to a greater or lesser degree depending on the value of ϵ [5].

3.3. Beam Search

Beam Search (BS) [7, 8] also explores several paths, whose number is limited by a parameter b called *beam width*. In fact, at most b nodes are explored at each level. Hence, i) it performs an ES in the first $k^* - 1$ levels, where k^* is the lowest integer such that $b < 2^{k^*}$ and ii) it explores just b nodes from the k^*-*th* level to the leaves, those with the highest marginal joint conditional probability. BS encapsulates both GS (with $b = 1$) and ES (with $b = 2^m$), then, adjusting the beam width, this method establishes a balance between the computational cost and the performance. BS also tends to optimize the subset 0/1 loss when b is large enough. The authors of [8] state that BS converges rapidly to the optimal predictions in terms of subset 0/1 loss with $b < 15$. They also observe that with $b = 15$ a significant fraction of the actual labels are predicted as relevant even when the label combination selected is not exactly correct.

3.4. Methods based on Monte Carlo sampling

Monte Carlo sampling was also proposed [5, 16] to perform inference in PCC. In them, the classifiers f_j induce probability distributions to draw random values from which the prediction of the value of ℓ_j is obtained whereas in GS, such prediction is taken directly from the evaluation of f_j. This flexibility of Monte Carlo approaches makes possible to perform the process several times, each one offering a label combination, although the same combination may appear more than once. After generating a certain number of label combinations, they must be aggregated to provide the final prediction. In this sense, one can take the most frequent label combination (the mode) [5] or the label combination with the highest joint conditional probability [16]. In any case, Monte Carlo approaches converge to optimize the subset 0/1 loss when the number of predictions drawn is large enough, but they do not guarantee to reach optimal predictions. As final remark, the number of nodes expanded by these approaches is $m \cdot q$, where m is the number of labels and q the number of predictions drawn.

3.5. A algorithm using an admissible heuristic designed for linear classifiers*

The A* algorithm has been also proposed for inference in PCC [10, 12] using an admissible heuristic only suitable when the classifiers f_i are linear. Since the main contribution of the paper continues this research designing an alternative heuristic with more promising properties, let us give a detailed description of this approach. The A* algorithm [15, 19] explores the best node according to an evaluation function e which can be decomposed in two other evaluation functions g (called the known part) and h (called the unknown part). In this way, for a generic node k, $g(k)$ computes the cost or gain for going from the root to the node k in a tree, whereas $h(k)$ estimates the cost or gain for going from the node k to a leave. Hence, $e(k)$ computes the cost or gain for going from the root to a leave through the node k. Usually, it is possible to obtain the exact value of $g(k)$, but $h(k)$ must be estimated using an heuristic. One of the principles of A* algorithm is that only if the heuristic is admissible[2], the A* algorithm is able to reach an optimal solution. Another crucial principle is that the A* algorithm is the most efficient algorithm for any heuristic, because there is not any other algorithm that using the same heuristic expands fewer nodes than A*. The particular case of using the A* algorithm for inference in PCC means to consider a gain rather than a cost, since the interest is to optimize the subset 0/1 loss. Besides, the function of g and h to obtain e is the product function due to the product rule of probability, that is, for a node k of the level j, $e(k) = g(k) \cdot h(k)$. Then, $e(k) = \mathbf{P}(y_1, \ldots, y_m \,|\, \boldsymbol{x})$ will be the joint conditional probability that applying the product rule of probability is decomposed into $g(k) = \mathbf{P}(y_1, \ldots, y_j \,|\, \boldsymbol{x})$, which is the marginal joint conditional probability of labels from ℓ_1 to ℓ_j in the order of the chain, and $h(k) = \mathbf{P}(y_{j+1}, \ldots, y_m \,|\, \boldsymbol{x}, y_1, \ldots, y_j)$, which is the marginal joint conditional probability of labels from ℓ_{j+1} to ℓ_m.

Notice that the above-mentioned requirement for g of being exactly computed is satisfied, since the values of $y_1, \ldots, y_j$ are known at level j. On the

[2] An heuristic h is admissible if for any node k, $h(k)$ does not overestimate (in case of $e(k)$ is a cost function) or underestimate (in case of $e(k)$ is a gain function) the actual cost or gain.

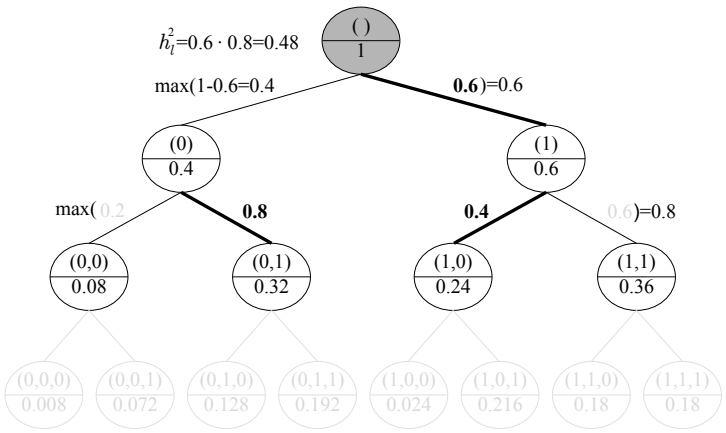


Figure 2: An example of h_l^d heuristic for $d = 2$ levels of depth, hence, only two levels of depth from there are expanded. The node in dark grey is the node for which the heuristic is computed. Bold lines indicate the evaluations of the classifiers that must be computed to obtain the value of the heuristic, which are, at most, two evaluations per classifier, that which respectively correspond to the maximum of the left branches and to the maximum of the right branches per level. Notice that only one evaluation of the classifier is performed for the contiguous level of the node for which the heuristic is computed

other hand, h must be estimated, since the values of $y_{j+1}, \ldots, y_m$ are unknown, so h will be an heuristic. Moreover, it must be an admissible heuristic to guarantee reaching an optimal solution. Also, g is, in fact, the same evaluation function that ϵ-A and BS consider in order to choose the node to be explored next. Even more, ϵ-A with $\epsilon = 0$ is equivalent to the A* algorithm with $h = 1$ as heuristic [10, 12].

The design of an admissible heuristic requires to define a function that returns an upper bound of $\mathbf{P}(y_{j+1}, \ldots, y_m \,|\, \boldsymbol{x}, y_1, \ldots, y_j)$ as close as possible to the exact value. An upper bound can be easily obtained performing the product of the highest probability for each remaining level, from label ℓ_{j+1} to label ℓ_m. Besides, the depth of this computation may be limited to d levels, from ℓ_{j+1} to ℓ_{j+d} and upperbounding the probability by 1 for the remaining levels, from ℓ_{j+d+1} to ℓ_m. This is the heuristic proposed in [10, 12] that shall be denoted here as h_l^d, in which d represents the depth and the subscript l means that the

heuristic is only valid for linear models. An example is depicted in Figure 2. There, the value of the heuristic h_l^2 for the root node is 0.48 which is an upper bound of the exact value of $\mathbf{P}(y_{j+1}, \ldots, y_m \,|\, \boldsymbol{x}, y_1, \ldots, y_j) = 0.216$. Besides, this value is also an upper bound of the highest joint probability for those nodes at the $j+d$-th level (0.36).

Therefore, the computation of h_l^d is reduced to calculate the highest probability for each level. This is not difficult for linear classifiers, since this kind of classifiers, f_i, are defined by means of $(\boldsymbol{w}^i = [\boldsymbol{w}_x^i, \boldsymbol{w}_y^i], \beta^i)$, where $\boldsymbol{w}_x^i$ and $\boldsymbol{w}_y^i$ are respectively the weight vectors for the $\boldsymbol{x}$ part (the description of the example) and the $\boldsymbol{y}$ part (the values of the previous labels in the chain for the example) and β^i is the intercept term. Mathematically, h_l^d can be computed as follows (see [12] for a complete deduction):

$$h_l^d = \prod_{i=j+1}^{j+d} \max_{v \in \{0,1\}} \{\mathbf{P}(y_i = v \,|\, \boldsymbol{x}, y_1, \ldots, y_j, y_{j+1}^{i,v}, \ldots, y_{i-1}^{i,v})\} \cdot \prod_{i=j+d+1}^{m} 1, \quad (10)$$

where $\mathbf{P}(y_i = v \,|\, \boldsymbol{x}, y_1, \ldots, y_j, y_{j+1}^{i,v}, \ldots, y_{i-1}^{i,v})$ for each i with $j+1 \leq i \leq j+d$. The maximum value for either $v = 1$ or $v = 0$ only depends on the value that takes each $y_k^{i,v}$ because the part $\langle \boldsymbol{w}_x^i, \boldsymbol{x} \rangle + \beta_i$ is constant for each instance $\boldsymbol{x}$. Hence, it is straightforward to determine $y_k^{i,v}$ looking at the corresponding value of $w_{y,k}^i$ depending on v:

1) using $f_i(\boldsymbol{x}, y_1, \ldots, y_j, y_{j+1}^{i,1}, \ldots, y_{i-1}^{i,1})$ when $v = 1$. In this case, we set $y_k^{i,1}$ for $j+1 \leq k \leq i-1$ to 1 if the corresponding $w_{y,k}^i$ of the linear classifier is positive and 0 otherwise. These values for $y_k^{i,1}$ for $j+1 \leq k \leq i-1$ make the probability $\mathbf{P}(y_i = v \,|\, \boldsymbol{x}, y_1, \ldots, y_j, y_{j+1}^{i,v}, \ldots, y_{i-1}^{i,v})$ be maximum when $v = 1$.

2) using $1 - f_i(\boldsymbol{x}, y_1, \ldots, y_j, y_{j+1}^{i,0}, \ldots, y_{i-1}^{i,0})$ when $v = 0$. In this case, we set $y_k^{i,1}$ for $j+1 \leq k \leq i-1$ to 1 if the corresponding $w_{y,k}^i$ of the linear classifier is negative and 0 otherwise. These values for $y_k^{i,1}$ for $j+1 \leq k \leq i-1$ make the probability $\mathbf{P}(y_i = v \,|\, \boldsymbol{x}, y_1, \ldots, y_j, y_{j+1}^{i,v}, \ldots, y_{i-1}^{i,v})$ be maximum when $v = 0$.

Taking the maximum of the probability $\mathbf{P}(y_i = v \,|\, \boldsymbol{x}, y_1, \ldots, y_j, y_{j+1}^{i,v}, \ldots, y_{i-1}^{i,v})$ between the two values when $v = 1$, obtained in 1), and when $v = 0$, obtained in 2), for each i with $j + 1 \leq i \leq j + d$ and performing the product of this maximum for all i such that $j + 1 \leq i \leq j + d$, the value of the heuristic h_l^d of depth d for a node of level j is provided. Remember that the values of y_i for $1 \leq i \leq j$ are known at all nodes of level j, so they depend on the node of the level j for which the heuristic is computed.

Figure 2 illustrates the computation of the heuristic h_l^2 when $d = 2$ levels of depth for the unique node (the root) of level $j = 0$ and $m = 3$. In this case i takes values $i = 1, 2$ ($1 = j + 1 \leq i \leq j + d = 2$).

1) For $i = 1$, $\mathbf{P}(y_1 = 1 \,|\, \boldsymbol{x}) = 0.6$ and $\mathbf{P}(y_1 = 0 \,|\, \boldsymbol{x}) = 0.4$, hence, the maximum is reached when $v = 1$ and is 0.6.

2) For $i = 2$, $\mathbf{P}(y_2 = 1 \,|\, \boldsymbol{x}, y_1^{2,1})$ equals 0.8 when $y_1^{2,1} = 0$ and equals 0.6 when $y_1^{2,1} = 1$, hence, for $v = 1$ the maximum is 0.8. In the same way, $\mathbf{P}(y_2 = 0 \,|\, \boldsymbol{x}, y_1^{2,0})$ equals 0.2 when $y_1^{2,0} = 0$ and equals 0.4 when $y_1^{2,0} = 1$, hence, for $v = 0$ the maximum is 0.4. Now, the maximum between 0.8 (for $v = 1$) and 0.4 (for $v = 0$) is 0.8, reached for $v = 1$.

Computing the product of the maximum reached in 1) and 2), one obtains $0.6 \cdot 0.8 = 0.48$, which is the value of the heuristic for the root.

In addition to the limitation of the heuristic h_l^d of being just valid for linear classifiers as base models, it is a quite time consuming heuristic because it requires to evaluate $2d - 1$ models. Hence, the power of exploring fewer nodes than other state-of-the-art algorithms is lessened by the time spent in computing h_l^d [12].

4. Exhaustive admissible heuristic in the A* algorithm for nonlinear classifiers

This section details the steps followed to build the exhaustive heuristic proposed in this paper for the A* algorithm as an alternative to the one presented in [10, 12]. The goal is to design an heuristic that may be used with

both, linear and nonlinear models. First of all, such heuristic must be admissible, hence, it cannot underestimate the marginal joint conditional probability $\mathbf{P}(y_{j+1}, \ldots, y_m \,|\, \boldsymbol{x}, y_1, \ldots, y_j)$. This leads to trivially think of selecting the maximum value that this probability can take, which will be the optimal possibility h^*:

$$h^* = \max_{(y_{j+1},\ldots,y_m)\in\{0,1\}^{m-j}} \mathbf{P}(y_{j+1}, \ldots, y_m \,|\, \boldsymbol{x}, y_1, \ldots, y_j) \tag{11}$$

Applying the product rule of probability, that expression leads to

$$h^* = \max_{(y_{j+1},\ldots,y_m)\in\{0,1\}^{m-j}} \prod_{i=j+1}^{m} \mathbf{P}(y_i \,|\, \boldsymbol{x}, y_1, \ldots, y_j, y_{j+1}, \ldots, y_{i-1}) \tag{12}$$

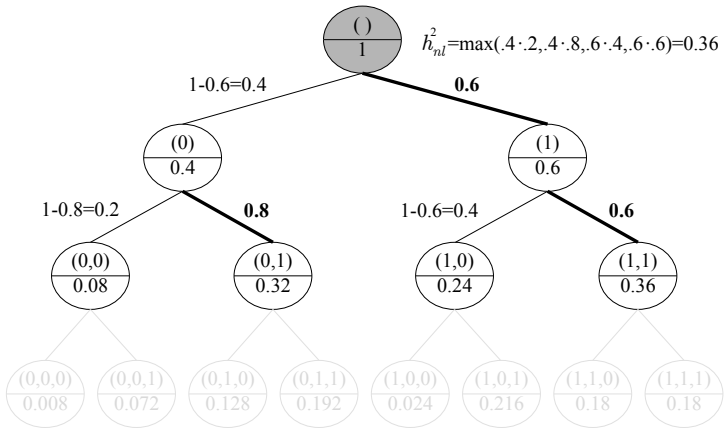
The main problem is that evaluating this maximum is not computationally acceptable, since it means to apply an ES for the subset of labels $\mathcal{L} = \{\ell_{j+1}, \ldots, \ell_m\}$.

The main idea consists in relaxing the optimality of the heuristic h^* in exchange of designing a computationally feasible heuristic. Our proposal involves including a parameter d for limiting the depth of the heuristic in order to control the computational cost. In this sense, an ES is carried out just until d levels of depth in the tree. This is possible thanks to the fact that any probability is delimited by 1. Hence, for a node of the level j and a value of d such that $0 \leq d \leq m - j$, the probabilities $\mathbf{P}(y_i \,|\, \boldsymbol{x}, y_1, \ldots, y_j, y_{j+1}, \ldots, y_{i-1})$ are computed i) applying ES from level $j + 1$ to $j + d$, that is, evaluating $\mathbf{P}(y_i \,|\, \boldsymbol{x}, y_1, \ldots, y_j, y_{j+1}, \ldots, y_{i-1})$ for these levels and taking the combination that reaches the maximum and ii) delimiting $\mathbf{P}(y_i \,|\, \boldsymbol{x}, y_1, \ldots, y_j, y_{j+1}, \ldots, y_{i-1})$ by 1 from level $j + d + 1$ to m. Hence, we define the heuristic h^d_{nl}, in which the subscript nl indicates that the heuristic is also valid for nonlinear classifiers, as:

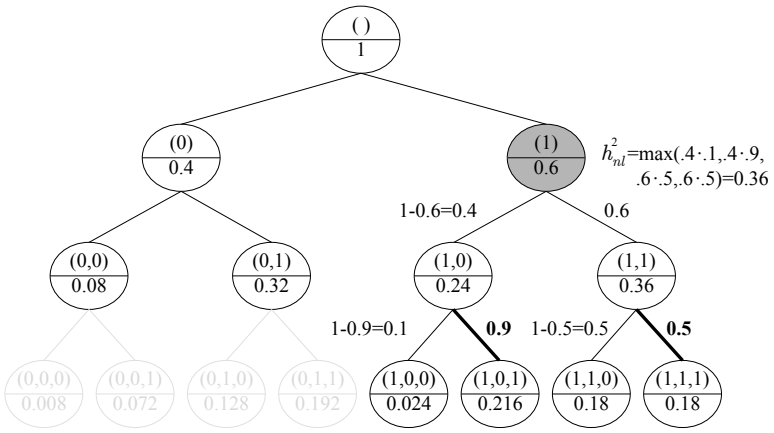
$$h^d_{nl} = \max_{(y_{j+1},\ldots,y_{j+d})\in\{0,1\}^d} \prod_{i=j+1}^{j+d} \mathbf{P}(y_i \,|\, \boldsymbol{x}, y_1, \ldots, y_j, y_{j+1}, \ldots, y_{i-1}) \cdot \prod_{i=j+d+1}^{m} 1 \tag{13}$$

or equivalently

$$h^d_{nl} = \max_{(y_{j+1},\ldots,y_{j+d})\in\{0,1\}^d} \prod_{i=j+1}^{j+d} \mathbf{P}(y_i \,|\, \boldsymbol{x}, y_1, \ldots, y_j, y_{j+1}, \ldots, y_{i-1}) \tag{14}$$



(a) A parent node



(b) A child node

Figure 3: An example of h^d_{nl} heuristic for $d = 2$ levels of depth, hence, only two levels of depth from there are expanded. The nodes in dark grey are the nodes for which the heuristic is computed: (a) Root node (unique node of level $j = 0$) (b) a child node (the right node out of two existing nodes in level $j = 1$). Bold lines indicate the evaluations of the classifiers that must be computed to obtain the value of the heuristic. Notice that those evaluations are only those corresponding to the classifier of the deepest level, except for the root node in which all possible evaluations must be computed

Notice that h^d_{nl} continues being admissible, but less dominant[3] than the optimal heuristic h^*. The particular case of $d = 0$ is the constant heuristic $h = 1$. Hence, h^d_{nl} is in turn more dominant than h^0_{nl}. In general, h^d_{nl} becomes more dominant as d increases, and then, it gets to explore fewer nodes, but at the cost of increasing the computational cost. This computational cost mainly depends on the number of models that must be evaluated to calculate the heuristic. Theoretically, h^d_{nl} requires to evaluate $2^d - 1$ models, which means to evaluate many more models than the number of models evaluated by h^d_l $(2d - 1)$. However, an appropriate implementation of h^d_{nl} (see next Section) can reduce that number to 2^{d-1}, making the computational cost of both heuristics comparable for $d \leq 4$.

Figure 3 illustrates the ways of computing the heuristic h^d_{nl} for $d = 2$ in two different cases: for a parent node and for one of its child. Looking to the former case (Figure 3(a)), the value of h^2_{nl} for the root node is 0.36, which corresponds to the highest joint conditional probability of the nodes of the 2-th level (label combination (1,1)). This value is an upper bound of the optimal value $h^* = 0.211$ which corresponds to the path of the label combination (1,0,1). But the most interesting issue of our proposal is that this bound is tighter than the one computed by h^2_l (0.48) (see Figure 2). Notice that for the latter case (Figure 3(b)), it is just necessary to evaluate the classifiers of the last level because the classifiers of previous levels were evaluated when the heuristic was computed for the parent node.

The reason why h^d_l is not able to deal with nonlinear models is because h^d_l is computed taking into account the weights of the linear models (see (10)). In case of h^d_{nl}, those weights are not required because the computation of h^d_{nl} just wisely applies the models over the label combinations at the deepest level of the corresponding subtree. For instance, to compute h^2_{nl} for the child node in Figure 3(b), we just need to apply model f_3 over the label combinations $(y_1 = 1,\ y_2 = 0)$ and $(y_1 = 1,\ y_2 = 1)$. With these two evaluations, it is possible

[3]When e is a gain function, an heuristic h_1 is more dominant than another heuristic h_2, denoted by $(h_1 \prec h_2)$, if the value of h_2 is not lower than the value of h_1 for any node

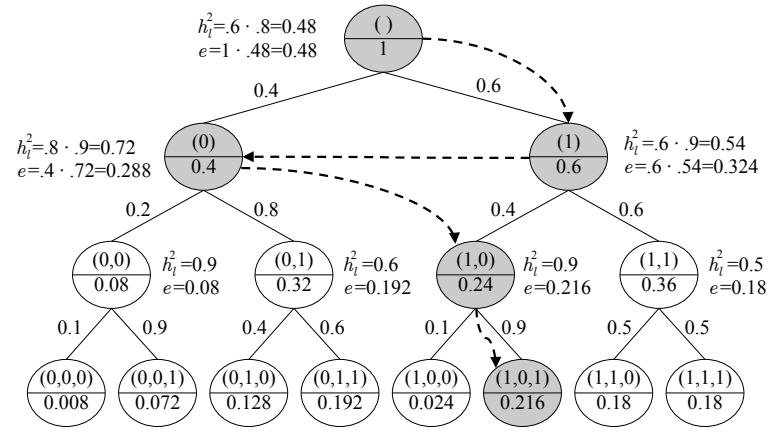
to compute the probabilities $\mathbf{P}(y_3 = 1 \,|\, \boldsymbol{x}, y_1 = 1, y_2 = 0) = 0.9$ and $\mathbf{P}(y_3 = 1 \,|\, \boldsymbol{x}, y_1 = 1, y_2 = 1) = 0.5$. The other two probabilities are trivially obtained as $\mathbf{P}(y_3 = 0 \,|\, \boldsymbol{x}, y_1 = 1, y_2 = 0) = 1 - \mathbf{P}(y_3 = 1 \,|\, \boldsymbol{x}, y_1 = 1, y_2 = 0) = 1 - 0.9 = 0.1$ and $\mathbf{P}(y_3 = 0 \,|\, \boldsymbol{x}, y_1 = 1, y_2 = 1) = 1 - \mathbf{P}(y_3 = 1 \,|\, \boldsymbol{x}, y_1 = 1, y_2 = 1) = 1 - 0.5 = 0.5$.

Finally, Figure 4 shows how different are the paths followed by A* when h^2_l (top) and h^2_{nl} (bottom) are taken as heuristics. First, notice that both approaches reach a Bayes-optimal solution (unique in this case), which it is the label combination (1,0,1) that has the highest joint conditional probability. However, h^2_{nl} requires fewer steps than h^2_l, since it follows the most direct path. This is due to the fact that the estimates computed by h^2_{nl} are more accurate. In this case, this means that the estimates are smaller and closer to the optimum. Notice that this occurs for the three top-nodes of the tree in Figure 4 (0.36 vs. 0.48, 0.48 vs. 0.72 and 0.36 vs. 0.54).

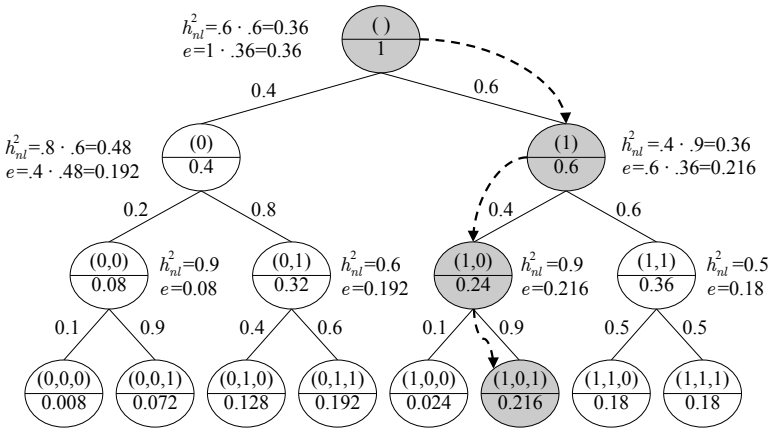
4.1. An efficient implementation of the heuristic

This section proposes a complete implementation of the A* algorithm using h^d_{nl} as heuristic, including an efficient algorithm for computing h^d_{nl} that reduces the number of models that must be evaluated from $2^d - 1$ to 2^{d-1}. The key idea is that each node stores the probabilities necessary to compute the heuristic (see Figure 5). Then, when a node is expanded, those probabilities are copied and reused to calculate the heuristic for its children. Applying this solution, only the probabilities of the *d-th* level must be computed because the rest were already calculated for the parent node. This implementation makes h^d_{nl} evaluate fewer or equal models than h^d_l when $d \leq 3$.

Algorithm 1 shows the pseudocode of the implementation of A* algorithm using h^d_{nl}. Initially, for the root, all the probabilities until level d must be computed (see *CompProbs* in line 3). When a non root node is expanded, the information of its parent is copied and reused (see $CopyHeuristicVector$ in lines 13 and 19). In this case, only the last-level classifier is evaluated for obtaining just the new needed probabilities (see $CompNewProbs$ in lines 14 and 21 respectively for the left and right children of the node expanded). Both



(a) h_l^d



(b) h_{nl}^d

Figure 4: Path (dotted arrows) followed by the algorithm A* using (a) h_l^d and (b) h_{nl}^d both with $d = 2$ levels of depth. Both approaches reach the Bayes-optimal solution but h_{nl}^2 requires fewer steps than h_l^2

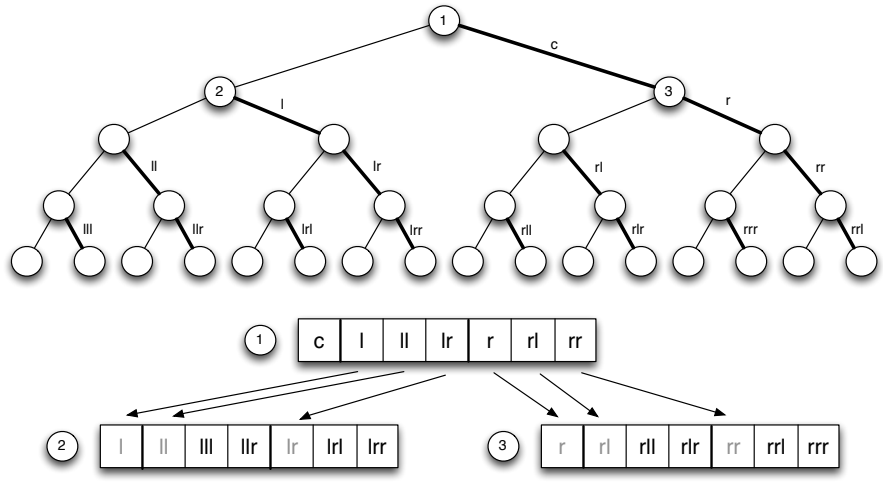


Figure 5: Optimal computation of the heuristic. The heuristic vector represents the computed probabilities of the tree using a preorder traversal. The example assumes that node 1 is expanded, so its children, nodes 2 and 3, must be added to the Q list. The computation of the heuristic for both nodes reuses the information contained in the heuristic structure of node 1. Only the probabilities of the lower level must be computed (positions in bold in vectors 2 and 3)

children of an expanded node are included in the list of candidate nodes to be expanded (see $Insert$ in lines 16 and 22), taking each time the node with the highest joint conditional probability (see line 6). The algorithm ends when the expanded node is a leaf or equivalently the level of the expanded node is m.

Algorithm 2 details the procedure to copy reusable probabilities of the vector $\boldsymbol{v}$ ($CopyHeuristicVector$). Particularly, when the heuristic must be computed for a node, it first copies the probabilities previously computed for its parent. Also, Algorithm 3 describes the recursive function which computes all the joint probabilities from the vector of probabilities ($\boldsymbol{v}$) of a node. In particular, it takes advantage of the probabilities previously saved in the way that none probability is computed twice. Finally, Algorithm 4 presents the iterative function to compute the missing probabilities of the vector $\boldsymbol{v}$ for a new node. Hence, this function evaluates the classifier of the $j+d$-th level in order to obtain the new

probabilities needed to compute the heuristic for the current node. It also stores such probabilities for being reused just in case the heuristic for any descendant must be computed later on.

Notice that all of these algorithms make sense if $d \geq 2$, since for the particular case of $d = 1$, where only one level is explored, the implementation may be considerably simplified. At sight of them, the computation of h^d_{nl} means to calculate new 2^{d-1} evaluations of the last-level $(j + d)$ classifier each time the heuristic is computed for a node, in contrast to the new $2d - 1$ evaluations of d different classifiers for h^d_l [12] (see Figure 5). Hence, both heuristics need to compute the same new classifier evaluations for $d = 1$ (1 evaluation). The number of classifier evaluations of h^d_{nl} and h^d_l are approximately equal for $2 \leq d \leq 4$ (2 vs. 3 evaluations when $d = 2$, 4 vs. 5 when $d = 3$, and 8 vs. 7 when $d = 4$). From $d = 5$, the number of classifier evaluations to compute begins to differ, being 16 evaluations for h^d_{nl} and 9 for h^d_l. This fact suggests not to take high values of d, because if not the advantage gained by the dominance of the heuristic would not offset the computational cost involved in its computation. In exchange of that, i) h^d_{nl} is more dominant than h^d_l for the same level of depth d and ii) unlike h^d_l, h^d_{nl} is suitable for nonlinear classifiers as base methods. According to the experiments performed, our advice is not to consider values of d greater than 3.

5. Experiments

The experiments were carried out over the same benchmark multi-label datasets used in [12], whose number of labels ranges from 5 to 374 (see Table 1 for the main properties of the datasets). The algorithm A* using the exhaustive heuristic introduced in this paper, h^d_{nl}, is compared to the heuristic, h^d_l, proposed in [10, 12]. As commented before, the values of d must not be so high due to the computational cost, then only values of $d \in \{1, 2, 3\}$ were considered. The algorithm A* using both heuristics are in turn compared to other state-of-the art algorithms for inference in PCC. Particularly, it is com-

Table 1: Properties of the datasets

Datasets	Instances	Attributes	Labels	Cardinality
bibtex	7395	1836	159	2.40
corel5k	5000	499	374	3.52
emotions	593	72	6	1.87
enron	1702	1001	53	3.38
flags	194	19	7	3.39
image	2000	135	5	1.24
mediamill*	5000	120	101	4.27
medical	978	1449	45	1.25
reuters	7119	243	7	1.24
scene	2407	294	6	1.07
slashdot	3782	1079	22	1.18
yeast	2417	103	14	4.24

pared with GS, ϵ-A with different values of $\epsilon \in \{.0, .25, .5\}$ and BS with several values of $b \in \{1, 2, 3\}$. Remember that GS, ϵ-A (.5) and BS with $b = 1$ explore the same path in the tree, so they attain identical results. The same occurs to ϵ-A (.0), UC and the algorithm A* with both h_l^0 and h_{nl}^0 (or equivalently whit the heuristic $h = 1$). None experiment is included for the Monte Carlo methods, since they have been shown to be less efficient than the ϵ-A algorithm [11] in addition of not guarantee to obtain an optimal prediction.

5.1. Subset 0/1 scores

First of all, we have tested the heuristic h_{nl}^d for the A* algorithm in comparison with other state-of-the-art algorithms (GS, ϵ-A with different values of $\epsilon \in \{.0, .25, .5\}$ and BS with several values of $b \in \{1, 2, 3\}$) using as base method a nonlinear classifier. Particularly, the nonlinear base classifier taken was SVM with probabilistic output and with radial basis function as kernel function. The parameters C of SVM and G of the radial basis kernel were optimized applying

Table 2: Subset 0/1 loss for the methods when a nonlinear classifier (SVM with radial basis kernel) is taken as base method for PCC. Those scores that are equal or lower than the optimal predictions are in bold

	UC/A*		GS/BS(1)	BS(2)	BS(3)
Datasets	ϵ-A(.0)	ϵ-A(.25)	ϵ-A(.5)		
bibtex	**81.20**	81.22	81.34	81.22	**81.20**
corel5k	**98.02**	98.48	98.54	98.22	98.10
emotions	**66.10**	**66.10**	69.79	66.77	**66.10**
enron	**84.20**	84.84	86.78	84.49	84.31
flags	**82.45**	83.47	83.45	82.97	**82.45**
image	**58.20**	58.45	60.20	**58.05**	**58.20**
mediamill*	**84.36**	**84.36**	85.02	84.48	84.40
medical	**28.73**	**28.73**	29.14	**28.73**	**28.73**
reuters	**21.98**	22.00	23.14	22.03	**21.98**
scene	**31.15**	**31.15**	32.40	**31.15**	**31.15**
slashdot	**51.27**	51.53	53.15	**51.24**	51.27
yeast	**77.24**	77.62	79.77	77.33	**77.24**

a grid search over the values of $C, G \in \{10^{-2}, \ldots, 10^{1}\}$ using the brier loss [1] as target function estimated through a 2-fold cross validation of five repetitions. The heuristic h_l^d is out of this comparison, since its computation is only suitable when a linear classifier is taken as base method.

Tables 2 and 3 respectively show the subset 0/1 loss estimated by a 10-fold cross validation for UC, A* with the heuristic h_{nl}^d, GS, ϵ-A algorithm and BS and the p-values of the underlying Wilcoxon signed-rank test. The approaches UC, ϵ-A(.0) and the A* algorithm using whatever admissible heuristic provide optimal solutions in terms of the subset 0/1 loss (see the scores in the first column of the table). As seen, there is four cases out of 12 for which ϵ-A(.25) equals them. None case exists where GS (or BS(1) or ϵ-A(.5)) improves the algorithms with theoretically obtain optimal solutions. BS(2) does it in 2 cases

Table 3: Wilcoxon signed-rank test for the methods when a nonlinear classifier is taken as base method for PCC

Optimal methods	others	p-value
UC/A* ϵ-A(.0)	ϵ-A(.25)	0.0078
UC/A* ϵ-A(.0)	GS/BS(1) ϵ-A(.5)	0.0005
UC/A* ϵ-A(.0)	BS(2)	0.0488
UC/A* ϵ-A(.0)	BS(3)	0.2500

(flags and slashdot) and equals them in other 2 cases. Finally, BS(3) gets equal results in 8 cases and it is never better. The p-values of the Wilcoxon signed-rank test confirm these results, namely, i) the methods that guarantee reaching Bayes-optimal predictions are significantly better than ϵ-A(.25) and GS (or BS(1) or ϵ-A(.5)) for $\alpha = 0.01$, and than BS(2) but for $\alpha = 0.05$. ii) The predictions of BS become close to optimal predictions as b increases.

Secondly, and with the aim of also including a comparison of the above methods with the A* algorithm using the heuristic h_l^d, we perform experiments taken a linear classifier as base method in PCC. Particularly, logistic regression with probabilistic output [9] was taken to build the binary classifiers f_i. In this case, only the parameter C was optimized, and it was made using a grid search over the values of $C \in \{10^{-3}, 10^{-2}, \ldots, 10^2, 10^3\}$ using the brier loss [1] as target function estimated through a 2-fold cross validation of five repetitions.

Tables 4 and 5 present the subset 0/1 loss estimated by a 10-fold cross validation for all the methods (including this time the A* with the heuristic h_l^d) and the p-values of the Wilcoxon signed-rank test. Again, the approaches UC, ϵ-A(.0) and the A* algorithm using whatever admissible heuristic provide optimal solutions in terms of the subset 0/1 loss (see the scores in the first column of the table). In this case, there is just one case (reuters) out of 12 for which ϵ-A(.25) obtains a better score than the algorithms which theoretically obtain optimal solutions and 3 cases for which it equals them. Also, GS (or BS(1)

Table 4: Subset 0/1 loss for the methods when a linear classifier (logistic regression) is taken as base method for PCC. Those scores that are equal or lower than the optimal predictions are in bold

	UC/A*		GS/BS(1)	BS(2)	BS(3)
Datasets	ϵ-A(.0)	ϵ-A(.25)	ϵ-A(.5)		
bibtex	**81.92**	81.95	82.19	**81.88**	**81.92**
corel5k	**97.48**	98.62	98.90	98.30	98.04
emotions	**71.16**	71.82	72.83	72.16	71.32
enron	**83.14**	84.26	85.43	83.43	83.37
flags	**87.13**	87.16	**86.13**	88.21	**87.13**
image	**68.35**	**68.35**	69.75	**68.35**	**68.35**
mediamill*	**83.86**	84.58	85.80	84.10	**83.86**
medical	**30.37**	**30.37**	30.67	**30.37**	**30.37**
reuters	**22.73**	**22.70**	23.60	**22.69**	**22.73**
scene	**31.86**	**31.86**	33.28	31.90	**31.86**
slashdot	**51.80**	52.22	54.49	**51.77**	**51.80**
yeast	**76.95**	77.62	79.77	**76.83**	77.08

or ϵ-A(.5)) improves the optimal algorithms in one case (reuters). BS(2) does it in 4 cases (bibtex, reuters, slashdot and yeast) and equals them in 2 cases. Notice that the wins achieved by BS(2) over optimal methods are much smaller than its losses. Interestingly, these wins, and most of the losses, disappear when $b = 3$, showing that BS converges to the optimal solution when b increases. In fact, BS(3) gets equal results in 8 cases and it is never better. The Wilcoxon signed-rank test in this case provides the same conclusions drawn when using nonlinear classifiers as base methods, but the p-values are slightly greater. The difference continues being significant for ϵ-A(.25) at level of $\alpha = 0.05$ and for GS (or BS(1) or ϵ-A(.5)) with $\alpha = 0.01$. The differences with respect to BS(2) and BS(3) are not significant.

Comparing the use of linear and nonlinear classifiers as base methods in

Table 5: Wilcoxon signed-rank test for the methods when a linear classifier (logistic regression) is taken as base method for PCC

Optimal methods	others	p-value
UC/A* ϵ-A(.0)	ϵ-A(.25)	0.0117
UC/A* ϵ-A(.0)	GS/BS(1) ϵ-A(.5)	0.0034
UC/A* ϵ-A(.0)	BS(2)	0.1309
UC/A* ϵ-A(.0)	BS(3)	0.1250

PCC, there are 8 datasets out of 12 in which using nonlinear classifiers as base methods improves the subset 0/1 scores with regard to applying linear classifiers. Therefore, as expected, this means that there are problems that require more complex models. This in turn supports the usefulness of the method proposed in this paper because it is the unique method based on the A* algorithm able to perform inference with nonlinear PCC.

5.2. Computational analysis

Let now study both heuristics (h^d_{nl} and h^d_l) in terms of number of nodes explored and computational time. In this case, we restrict the study using just linear classifiers (logistic regression) as base methods because h^d_l cannot be applied with nonlinear models. The rest of the approaches are also included to provide a broader comparison.

Table 6 shows the number of nodes explored by each method and the average time spent in performing the predictions taken in milliseconds. Obviously, GS (or ϵ-A(.5) or BS(1)) explores the least number of nodes, since it only follows one path in the tree, then, it explores many nodes as labels have the dataset (plus one if the root is also considered). Among the methods that do not guarantee to obtain optimal predictions, ϵ-A(.25) tends to expand fewer nodes that BS(2), but at expenses of offering worst scores in terms of the subset 0/1 loss. BS(3) expands the most number of nodes, in spite of getting predictions close to be optimal. Among the methods that obtain optimal solutions, the

Table 6: Averaged number of nodes explored (top) and averaged computational prediction time in milliseconds (bottom). The number in brackets is the number of labels of the dataset.

Datasets	h^1_{nl} h^1_l	h^2_{nl}	h^3_{nl}	h^2_l	h^3_l	ϵ-A(.0)	ϵ-A(.25)	GS/BS(1) ϵ-A(.5)	BS(2)	BS(3)
bibtex (159)	283.3	277.7	272.4	278.0	273.2	289.3	184.0	160.0	319.0	477.0
corel5k (374)	1456.9	1440.6	1425.4	1443.1	1431.6	1474.2	517.1	375.0	749.0	1122.0
emotions (6)	9.1	8.0	7.4	8.3	7.8	10.7	10.8	7.0	13.0	18.0
enron (53)	110.3	106.1	102.2	106.5	103.3	114.8	77.3	54.0	107.0	159.0
flags (7)	16.4	10.2	8.7	12.7	10.4	22.6	16.3	8.0	15.0	21.0
image (5)	6.6	6.1	6.0	6.3	6.1	7.3	7.7	6.0	11.0	15.0
mediamill* (101)	188.2	184.8	181.5	185.6	183.7	191.8	142.4	102.0	203.0	303.0
medical (45)	46.6	46.6	46.5	46.6	46.6	46.6	46.6	46.0	91.0	135.0
reuters (7)	8.2	8.1	8.0	8.1	8.1	8.2	8.3	8.0	15.0	21.0
scene (6)	7.2	7.1	7.0	7.2	7.1	7.2	7.3	7.0	13.0	18.0
slashdot (22)	25.0	24.7	24.4	24.9	24.9	25.3	24.9	23.0	45.0	66.0
yeast (14)	23.6	21.4	20.0	22.8	22.4	26.1	26.0	15.0	29.0	42.0
bibtex (159)	19.5	24.8	30.2	31.4	41.1	16.2	9.9	7.9	11.0	15.6
corel5k (374)	168.4	190.8	229.4	257.8	333.3	136.0	16.1	11.0	27.8	40.4
emotions (6)	0.6	0.7	0.7	0.8	0.9	0.6	0.6	0.4	0.5	0.6
enron (53)	8.5	14.5	18.4	14.1	18.5	7.2	3.0	1.9	3.9	5.5
flags (7)	1.1	0.9	0.9	1.4	1.5	1.2	0.7	0.4	0.5	0.7
image (5)	0.5	0.5	0.6	0.6	0.7	0.5	0.5	0.4	0.4	0.5
mediamill* (101)	14.4	18.9	23.3	24.2	32.6	11.5	5.5	3.7	7.2	10.5
medical (45)	3.3	4.6	5.7	5.7	7.7	2.7	2.7	2.7	3.3	4.6
reuters (7)	0.6	0.7	0.8	0.8	1	0.5	0.5	0.5	0.5	0.7
scene (6)	0.5	0.6	0.7	0.7	0.9	0.5	0.5	0.4	0.5	0.6
slashdot (22)	1.8	2.4	2.9	2.9	3.9	1.5	1.4	1.2	1.6	2.2
yeast (14)	1.6	2.0	2.3	2.6	3.4	1.5	1.0	0.6	1.0	1.4

algorithm A* using h^d_{nl} is the method that explores the fewest number of nodes for all values of d, as theoretically expected. In fact, it was shown that h^d_{nl} is the most dominant heuristic. Besides, it is more dominant as the level of depth d increases. Remarkably, the number of nodes explored by A* using h^d_{nl} is close to that of GS in some situations. However, sometimes the number of nodes explored by A* using h^d_{nl} for $d > 0$ is quite similar to that of ϵ-A(.0), UC or A* using $h = 1$. The reason may be because the probabilities provided by the classifiers f_i take values close to 0 or 1, then, there may not be so much uncertainty and the search may be highly directed.

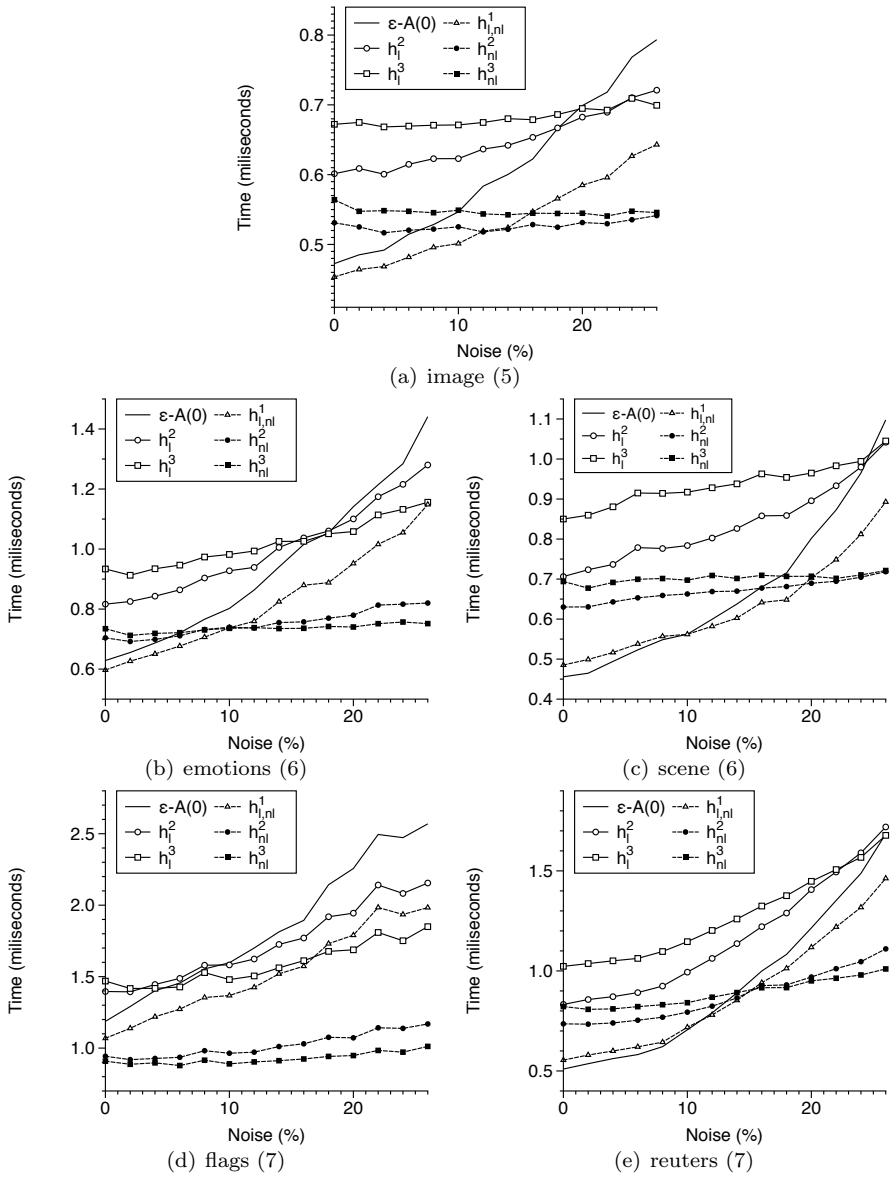
Concerning computational time, it is quite curious that ϵ-A(.0) and UC are

not so much slower than ϵ-A(.25) and ϵ-A(.5). Besides, they are quite similar to BS(2) and BS(3). Also, A* using h^d_{nl} with $d > 0$ is not as faster as ϵ-A(.0) (or A* using h^0_{nl}), in spite of exploring fewer nodes, due to the computational time spent in evaluating the heuristic. However, the difference is quite low in comparison with the heuristic h^d_l. At this respect, let us include some noise in the dataset in order to cause uncertainty and disturb the search and then, avoiding it being so direct.

Figures 6 and 7 show the number of nodes expanded and the computational time employed in milliseconds for $\epsilon-$A with $\epsilon=0$ and for A* using h^d_l and h^d_{nl} for different values of the parameter d $(1, 2, 3)$ when a percentage of noise ranging from 0% to 26% is included in the datasets. Only in case of the datasets bibtex, mediamill and enron, the percentage of noise added has had to be respectively limited to 4%, 8% and 10%, due to the high computational time involved. At sight of these figures, one can observe that using the heuristic h^d_{nl} hardly increases the computational time when the percentage of noise increases in case of datasets with few labels, being the one which presents the least increasing of computational time for datasets with high number of labels. In this last case, that is, when the increasing of the noise results in a considerable increasing in computational time because more nodes must be expanded to obtain an optimal solution is when the heuristic h^d_{nl} clearly shows its main benefit. It is quite remarkable that computational time spent by h^d_{nl} is always lower than the computational time spent by the heuristic h^d_l for the same level of depth. Hence, unlike h^d_l, the time spent in computing h^d_{nl} is not a disadvantage, but quite the opposite.

6. Conclusions

This paper proposes an admissible heuristic for the algorithm A* for performing inference in PCC either for linear and nonlinear classifiers as base methods. The heuristic is based on an exhaustive search, but including a depth parameter that establishes a balance between the number of nodes explored and the com-



(a) image (5)

(b) emotions (6)

(c) scene (6)

(d) flags (7)

(e) reuters (7)

Figure 6: Computational time employed (ms) for $\epsilon-$A with $\epsilon=0$ and for A* using h_l^d and h_{nl}^d for different values of the parameter d $(1, 2, 3)$ when certain percentage of noise is included in the datasets. Notice that for $d = 1$, the results are equal, since they follow the same path. Different percentages of noise are considered. All the datasets have few labels (from 5 to 7)

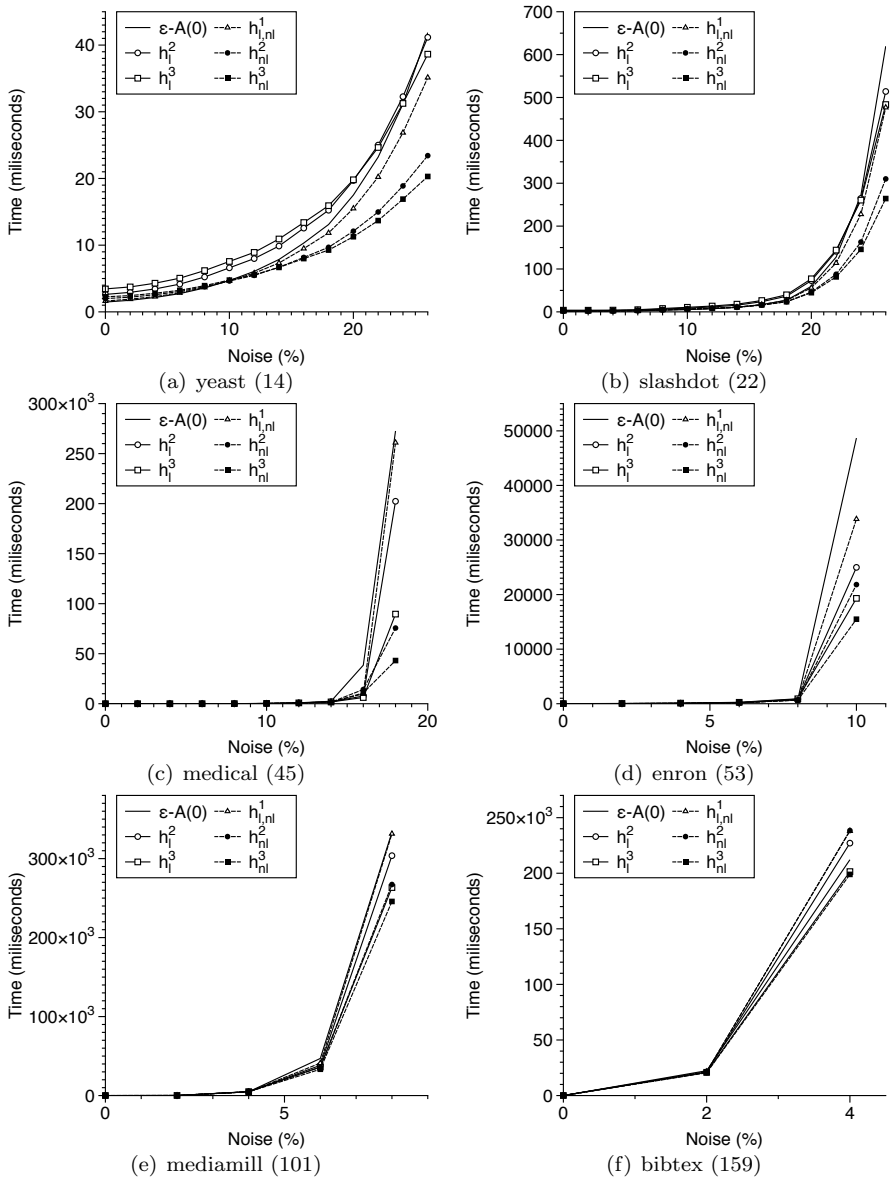


Figure 7: Computational time employed (ms) for $\epsilon-$A with $\epsilon=0$ and for A* using h_l^d and h_{nl}^d for different values of the parameter d $(1, 2, 3)$ when certain percentage of noise is included in the datasets. Notice that for $d = 1$, the results are equal, since they follow the same path. Different percentages of noise are considered. The number of labels range from 14 to 159

putational time. The method is theoretically shown to provide optimal solutions in terms of the subset 0/1 loss, since the heuristic is admissible. Besides, the heuristic is more dominant than other existing heuristics in the literature for the same purpose, then, it explores fewer nodes among the methods that guarantee to provide optimal solutions. Moreover, it is a suitable heuristic for any kind of probabilistic classifiers, including nonlinear classifiers. One of the key aspect of our proposal is to present an efficient implementation of the heuristic that significantly reduces the number of models evaluated in its computation.

The experiments confirm the theoretical properties of the proposed heuristic. However, the time spent in evaluating the heuristic can be sometimes a disadvantage. This is, for instance, for datasets that cause few uncertainty in the search and, hence, such search is quite directed because the computational time is slightly higher than that of the $\epsilon - A$ with $\epsilon = 0$ and hence, the number of nodes explored are not few enough to compensate that time. Conversely, when the search is not so directed, for instance, because the prediction problem is more uncertain, using the heuristic proposed in this paper makes the A* algorithm provide faster predictions than other state-of-the-art methods that also produce Bayes-optimal predictions, like the $\epsilon - A$ algorithm.

Acknowledgments

This research has been funded by MINECO (the Spanish Ministerio de Economía y Competitividad) and FEDER (Fondo Europeo de Desarrollo Regional), grant TIN2015-65069-C2-2-R (MINECO/FEDER).

References

[1] G.W. Brier, Verification of forecasts expressed in terms of probability, Monthly weather review 78 (1950) 1–3.

[2] W. Cheng, E. Hüllermeier, Combining instance-based learning and logistic regression for multilabel classification, Machine Learning 76 (2009) 211–225.

[3] K. Dembczyński, W. Cheng, E. Hüllermeier, Bayes Optimal Multilabel Classification via Probabilistic Classifier Chains, in: ICML, 2010, pp. 279–286.

[4] K. Dembczyński, W. Waegeman, W. Cheng, E. Hüllermeier, On label dependence and loss minimization in multi-label classification, Machine Learning 88 (2012) 5–45.

[5] K. Dembczynski, W. Waegeman, E. Hüllermeier, An analysis of chaining in multi-label classification., in: ECAI, volume 242 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2012, pp. 294–299.

[6] K. Dembczyński, W. Waegeman, E. Hüllermeier, An analysis of chaining in multi-label classification, in: ECAI 2012, pp. 294–299.

[7] A. Kumar, S. Vembu, A.K. Menon, C. Elkan, Learning and inference in probabilistic classifier chains with beam search, in: ECML/PKDD 2012, pp. 665–680.

[8] A. Kumar, S. Vembu, A.K. Menon, C. Elkan, Beam search algorithms for multilabel learning, Machine Learning 92 (2013) 65–89.

[9] C.J. Lin, R.C. Weng, S.S. Keerthi, Trust region Newton method for logistic regression, Journal of Machine Learning Research 9 (2008) 627–650.

[10] D. Mena, E. Montañés, J.R. Quevedo, J.J. del Coz, Using a* for inference in probabilistic classifier chains, in: IJCAI 2015, pp. 3707–3713.

[11] D. Mena, E. Montañés, J.R. Quevedo, J.J. del Coz, An overview of inference methods in probabilistic classifier chains for multilabel classification, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 6 (2016) 215–230.

[12] D. Mena, E. Montañés, J.R. Quevedo, J.J. del Coz, A family of admissible heuristics for a* to perform inference in probabilistic classifier chains, Machine Learning 106 (2017) 143–169.

[13] E. Montañés, J. Quevedo, J.J. del Coz, Aggregating independent and dependent models to learn multi-label classifiers, in: ECML'11, pp. 484–500.

[14] E. Montañés, R. Senge, J. Barranquero, J. Quevedo, J.J. del Coz, E. Hüllermeier, Dependent binary relevance models for multi-label classification, Pattern Recognition 47 (2014) 1494 – 1508.

[15] J. Pearl, Heuristics: Intelligent Search Strategies for Computer Problem Solving, Addison-Wesley Longman, Boston, MA, USA, 1984.

[16] J. Read, L. Martino, D. Luengo, Efficient monte carlo methods for multi-dimensional learning with classifier chains, Pattern Recognition 47 (2014) 1535 – 1546.

[17] J. Read, B. Pfahringer, G. Holmes, E. Frank, Classifier chains for multi-label classification, in: ECML/PKDD'09, LNCS, Springer, 2009, pp. 254–269.

[18] J. Read, B. Pfahringer, G. Holmes, E. Frank, Classifier chains for multi-label classification, Machine Learning 85 (2011) 333–359.

[19] S.J. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Pearson Education, 2 edition, 2003.

[20] R. Senge, J.J. del Coz, E. Hüllermeier, On the problem of error propagation in classifier chains for multi-label classification, in: Conference of the German Classification Society on Data Analysis, Machine Learning and Knowledge Discovery, 2012.

[21] R. Senge, J.J. del Coz, E. Hüllermeier, Rectifying classifier chains for multi-label classification, in: Learning, Knowledge, Adaptation 2013.

[22] G. Tsoumakas, I. Vlahavas, Random k-Labelsets: An Ensemble Method for Multilabel Classification, in: ECML/PKDD'07, Springer, 2007, pp. 406–417.

[23] Q. Wu, M.K. Ng, Y. Ye, X. Li, R. Shi, Y. Li, Multi-label collective classification via markov chain based learning method, Knowledge-Based Systems 63 (2014) 1–14.

[24] J.J. Zhang, M. Fang, J.Q. Wu, X. Li, Robust label compression for multi-label classification, Knowledge-Based Systems 107 (2016) 32 – 42.

Algorithm 1 Pseudocode of the implementation of A* algorithm using h^d_{nl}

1: **function** A*

Input: m, d, $\boldsymbol{x}$, $\{[\mathbf{W} = \{\boldsymbol{w}^i\}, \boldsymbol{\beta} = \{\beta^i\}], i = 1, \ldots, m\}$ a CC Linear Model

Output: Label combination with highest probability for $\boldsymbol{x}$

2: $\boldsymbol{v} \leftarrow Inic(\boldsymbol{x}, [\mathbf{W}, \boldsymbol{\beta}], d)$ // Probabilities of the h_{nl} for the root node

3: $e \leftarrow Max(CompProbs(\mathbf{1}_{2^d}, \boldsymbol{v}, d))$

4: $Q[1] \leftarrow \{[\,], 0, 1, e, \boldsymbol{v}\}$ // root, empty label set, level 0, $g = 1$ and $\boldsymbol{v}$

5: **while** true **do**

6: $Node \leftarrow Max(Q)$

7: **if** $Node.Level = m$ **then**

8: **return** $Node.Labels$ // Leaf node

9: $level \leftarrow Node.Level + 1$

10: $P \leftarrow Node.v[1]$ // Probability that the label is 1

11: // Left child

12: $g \leftarrow Node.g * (1 - P)$

13: $\boldsymbol{vl} \leftarrow CopyHeuristicVector(Left, \boldsymbol{v}, d)$

14: $CompNewProbs(\boldsymbol{vl}, d, \boldsymbol{x}, [\mathbf{W}, \boldsymbol{\beta}], [Node.Labels\ 0], level)$

15: $h^d_{nl} \leftarrow Max(JointProbs(\mathbf{1}_{2^d}, \boldsymbol{vl}, d))$

16: $Insert(Q, \{[Node.Labels\ 0], level, g, g * h^d_{nl}, \boldsymbol{vl}\})$

17: // Right child

18: $g \leftarrow Node.g * P$

19: $\boldsymbol{vr} \leftarrow CopyHeuristicVector(Right, \boldsymbol{v}, d)$

20: $CompNewProbs(\boldsymbol{vr}, d, \boldsymbol{x}, [\mathbf{W}, \boldsymbol{\beta}], [Node.Labels\ 1], level)$

21: $h^d_{nl} \leftarrow Max(JointProbs(\mathbf{1}_{2^d}, \boldsymbol{vr}, d))$

22: $Insert(Q, \{[Node.Labels\ 1], level, g, g * h^d_{nl}, \boldsymbol{vr}\}$

Algorithm 2 Copy reusable probabilities of the vector $\boldsymbol{v}$

1: **function** $CopyHeuristicVector$

Input: $Child$, $\boldsymbol{v}$, d

Output: An incomplete heuristic vector, copy corresponding values from $\boldsymbol{v}$

2: $\boldsymbol{v}' \leftarrow AllocMemory(2^d - 1)$

3: $l \leftarrow 2$ // Beginning of left part

4: $r \leftarrow 1 + 2^{d-1}$ // Beginning of right part

5: **if** $Child = Left$ **then** // Beginning of the part to be copied

6: $i \leftarrow l$

7: **else**

8: $i \leftarrow r$

9: $\boldsymbol{v}'[1] \leftarrow \boldsymbol{v}[i]$ // Probability of the top level

10: $i \leftarrow i + 1$

11: $ElementsPerLevel \leftarrow 1$ // the next level has 1 element

12: $level \leftarrow 3$

13: **while** $level \leq d$ **do**

14: **for** $j = [1 : ElementsPerlevel]$ **do**

15: $\boldsymbol{v}'[l] \leftarrow \boldsymbol{v}[i]$

16: $l \leftarrow l + 1, i \leftarrow i + 1$

17: **for** $j = [1 : ElementsPerlevel]$ **do**

18: $\boldsymbol{v}'[r] \leftarrow \boldsymbol{v}[i]$

19: $r \leftarrow r + 1, i \leftarrow i + 1$

20: $ElementsPerLevel \leftarrow ElementsPerLevel * 2$

21: $level = level + 1$

Algorithm 3 Recursive function to compute all the joint probabilities from the vector of probabilities ($\boldsymbol{v}$) of a given node

function *JointProbs*

Input: $\boldsymbol{P}$, $\boldsymbol{v}$, d // P is the vector of the joint probabilities

Output: Each element of P multiplied by the corresponding probabilities of $\boldsymbol{v}$

if $d = 1$ **then**

return $[\boldsymbol{P}[1] * (1 - \boldsymbol{v}[1]), \boldsymbol{P}[2] * \boldsymbol{v}[1]]$

else

$\boldsymbol{v}' \leftarrow \boldsymbol{v}[2 : end]$

return $[(1-\boldsymbol{v}[1]) * JointProbs(\boldsymbol{P}[1 : len(\boldsymbol{P})/2], \boldsymbol{v}'[1 : len(\boldsymbol{v}')/2], d-1),$

$\boldsymbol{v}[1] * JointProbs(\boldsymbol{P}[len(\boldsymbol{P})/2+1 : end], \boldsymbol{v}'[len(\boldsymbol{v}')/2+1 : end], d-1)]$

Algorithm 4 Iterative function to compute the missing probabilities of the vector $\boldsymbol{v}$ for a new node

function *CompNewProbs*

Input: $\boldsymbol{v}$, d, *Labels*, *Level*, $\boldsymbol{x}$, $\{[\mathbf{W} = \{\boldsymbol{w}^i\}, \boldsymbol{\beta} = \{\beta^i\}], i = 1, \ldots, m\}$ a CC Linear Model

Output: $\boldsymbol{v}$

$LabelComb \leftarrow BinaryPerm(d-1)$ // All bit patterns of length $d-1$

$j \leftarrow 1$

for $i = [1 + 2^{d-1} - 2^{d-2} : 2^{d-1}]$ **do** // Left probabilities

$\boldsymbol{v}[i] \leftarrow Prob(\boldsymbol{x}, [\mathbf{W}, \boldsymbol{\beta}], [Labels\ LabelComb(j)], Level)$

$j \leftarrow j + 1$

$j \leftarrow 1$

for $i = [1 + len(\boldsymbol{v}) - 2^{d-2} : len(\boldsymbol{v})]$ **do** // Right probabilities

$\boldsymbol{v}[i] \leftarrow Prob(\boldsymbol{x}, [\mathbf{W}, \boldsymbol{\beta}], [Labels\ LabelComb(j)], Level)$

$j \leftarrow j + 1$