
Article:
Baruah, Sanjoy, Burns, Alan orcid.org/0000-0001-5621-8816, Davis, Robert Ian orcid.org/0000-0002-5772-0928 et al. (1 more author) (2023) Optimally ordering IDK classifiers subject to deadlines. Real-Time Systems. ISSN 1573-1383

https://doi.org/10.1007/s11241-022-09383-w

Real-Time Systems
https://doi.org/10.1007/s11241-022-09383-w

Optimally ordering IDK classifiers subject to deadlines

Sanjoy Baruah[1] · **Alan Burns**[2] · **Robert I. Davis**[2] · **Yue Wu**[3]

Accepted: 10 April 2022

Abstract
A classifier is a software component, often based on Deep Learning, that categorizes each input provided to it into one of a fixed set of classes. An IDK classifier may additionally output "I Don't Know" (IDK) for certain inputs. Multiple distinct IDK classifiers may be available for the same classification problem, offering different trade-offs between effectiveness, i.e. the probability of successful classification, and efficiency, i.e. execution time. Optimal offline algorithms are proposed for sequentially ordering IDK classifiers such that the expected duration to successfully classify an input is minimized, optionally subject to a hard deadline on the maximum time permitted for classification. Solutions are provided considering independent and dependent relationships between pairs of classifiers, as well as a mix of the two.

Keywords Deep Learning · Hard deadlines · Classifiers · IDK cascades · Optimal synthesis

1 Extended version

The paper *"Optimal Synthesis of IDK cascades"* by Baruah et al. (2021), published in RTNS 2021, presented analysis and algorithms for determining the optimal sequentially ordering of probabilistically *independent* IDK classifiers, achieving the

✉ Robert I. Davis
rob.davis@york.ac.uk

Sanjoy Baruah
baruah@wustl.edu

Alan Burns
alan.burns@york.ac.uk

Yue Wu
yuew29@uw.edu

[1] Washington University in Saint Louis, Saint Louis, MO, USA

[2] University of York, York, UK

[3] University of Washington, Seattle, WA, USA

Published online: 14 May 2022

minimum expected duration for classification, with and without deadline constraints. This paper substantially extends that work by considering *independent* IDK classifiers (Sect. 3), *fully dependent* IDK classifiers (Sect. 4), and a mix of both *independent* and *dependent* relationships between classifiers (Sect. 5). In each case, analysis and optimal algorithms are derived that achieve the minimum expected duration for classification, including under deadline constraints (Sect. 6).

2 Introduction

Software components that are based on Deep Learning and related AI techniques are increasingly being deployed for classification problems in complex resource-constrained Cyber-Physical Systems. Such systems often require accurate predictions to be delivered in real time using limited computational resources.

Much of the recent research into Deep Neural Networks (DNNs) has however focused on improving the accuracy of classification. From a real-time perspective this ongoing quest for improved accuracy has arguably gone too far, resulting in DNNs that take substantial time to process even simple inputs that should in fact be relatively straightforward to classify. For example, Wang et al. (2018) showed that an order-of-magnitude increase in the execution time of DNNs has resulted in a negligible improvement in the accuracy of predictions for a considerable fraction of the ImageNet 2012 benchmark of validation images (Russakovsky et al 2015).

Balancing the trade-off between accuracy and latency becomes important if such DNNs are to be adopted for use in Cyber Physical Systems (CPS) that are expected to respond in a timely manner; for example, a DNN used for image processing within an autonomous driving CPS (Fujiyoshi et al. 2019). With this goal in mind, Wang et al. (2018) observed that if the advanced but slower DNNs were only used in the more challenging cases, then the time taken to achieve successful classification could be reduced. In effect, combining fast DNNs with accurate ones to reduce mean latency without any trade-off in accuracy.

This observation motivated Trappenberg and Back (2000) and Khani et al. (2016) to explore the use of ***IDK classifiers***, which may be viewed as bringing some degree of self awareness to classifiers. An IDK classifier is obtained from an existing base classifier by attaching a computationally light-weight augmenting classifier that enables the base classifier to additionally predict an auxiliary "**I** **D**on't **K**now" (IDK) class depending on the degree of uncertainty in the predictions of the base classifier.[1]

Specifically, an IDK classifier classifies an input as being in the IDK class if the base classifier is not able to predict some actual class for that input with a level of confidence that exceeds a predefined threshold value (Baruah et al. 2021).

The use of multiple classifiers to enhance the overall accuracy and precision of classification has been studied since as least the work of Nilsson (1965). An *ensemble* of classifiers is used to introduce diversity, either in the type of input,

[1] A similar notion is used by Madras et al. (2018), they allow the base classifier to *defer* by outputting the class **PASS**.

the classification model, or in the features of the data sets used during training. Oza and Tumer (2008) provide a review of the use of ensembles of classifiers. Examples of ensembles can be found in: autonomous vehicles, where cameras, LiDAR, radar, microphones and even satellite feeds can be combined to accomplish safe vehicle control; and in person recognition systems where iris images, fingerprints, face recognition and voice recognition are synthesised. The former clearly has strict timing constraints applied to the output of the ensemble's classification; a restricted form of the latter may also need to operate under timing constraints if it is being applied to the real-time monitoring of a crowd scene.

Ensembles of IDK classifiers, with different execution times and probabilities of success (i.e. of not outputting IDK), may be devised for the same classification problem. Wang et al. (2018) proposed arranging such IDK classifiers into *IDK cascades*, which are linear sequences of IDK classifiers designed to work as follows:

1. The first classifier in the IDK cascade is invoked first, for any input that needs to be classified.
2. If the classifier outputs a real class, rather than IDK, then the IDK cascade terminates and characterizes the input as being of the identified class.
3. Otherwise, if the classifier outputs IDK, then the subsequent classifier in the IDK cascade is invoked and the process continues from step 2.

Since it is a requirement that all inputs are successfully classified by an IDK cascade, it is assumed that the last classifier in the cascade always outputs a real class. We refer to a classifier that always outputs a real class as a *deterministic classifier*. There are various forms that the deterministic classifier can take. Wang et al. (2018) proposed that a human expert could be considered to be the deterministic classifier. For example, the driver of a semi-autonomous vehicle could be called upon to decide, in conditions that caused a camera-based classifier to fail (i.e. output IDK), if a partially obscured road sign ahead signifies a lower speed limit, and hence the vehicle should reduce speed, an action that has a safety-related timing constraint. In another application a fully developed DNN could be sufficiently accurate that it can take on the role of deterministic classifier; however, its computation requirements are such that it should only be executed when absolutely necessary, other more efficient classifiers should be used if they can cater for typical inputs. To deal with applications that exhibit high levels of uncertainty, it may be necessary to introduce the class *unclassifiable* that the final arbiter, the deterministic classifier, can output if a real class cannot be identified.

Given a collection of several different IDK classifiers for a particular classification problem, this paper considers how they should be sequentially ordered for execution so as to minimize the expected (i.e. average) duration taken to successfully classify an input, and if a deadline is specified, to additionally guarantee to always meet that deadline. The analysis and the algorithms required to solve these problems are impacted by the relationships between the IDK classifiers concerned.

Two IDK classifiers may behave in a way that is *independent* of one another. By independent, we mean that the probability that the second classifier will

output a real class is independent of whether it is run on all inputs or on only those inputs where the first classifier outputs IDK. For example, the ensemble examples given earlier for autonomous vehicle control and person recognition employ diverse inputs (radar, LiDAR and camera; iris images, face recognition and voice recognition) are likely to exhibit behavior that is independent. Indeed, Madani et al. (2012, 2013) show that very different sources of evidence such as text, audio, and video features are effectively independent. At the other extreme, two image-based classifiers that use the same input image but scale it to different resolutions (e.g. 64×64 or 256×256 pixels) (Hu et al. 2021) may be very similar and exhibit behavior that is *fully dependent*. By fully dependent, we mean that the first less powerful classifier is only able to successfully classify a strict subset of the inputs that the second more powerful classifier can recognize. This informal notion of similarity among classifiers is formalized in Sect. 2 via the concept of *conditional probability*.

The remainder of the paper is organized as follows. Section 2 describes the system model, terminology and notation used, along with the definitions of key concepts. Sections 3 and 4, present solutions for collections of IDK classifiers that are respectively, (i) independent and (ii) fully dependent with respect to one another. Section 5 considers collections of IDK classifiers that have (iii) a mix of dependent and independent relationships. Section 6 extends the analysis and algorithms for all three cases to problems where there is also a deadline constraint. Finally, Sect. 7 concludes with a summary and directions for future research.

3 System model, terminology, and notation

We consider a collection of n classifiers $K_1, K_2, \ldots, K_n$ that may be used for a given classification problem. Each classifier K_i is characterized by parameters (C_i, P_i), specifying its *execution time* C_i and its *success probability* P_i. These parameters denote that the classifier takes at most a time C_i to complete execution when invoked on an input, and returns a real class, rather than IDK, with probability P_i, where $0 < P_i \leq 1$. (For a discussion about how these parameters can be obtained, see (Baruah et al. 2021)). We refer to an ordered linear sequence of such classifiers as an *IDK cascade*.

The actual value of n is application dependent and many different values are to be found in the literature on classifier ensembles. As noted earlier, diversity comes from having independent classifiers with different types of input, different internal models and different training data. Even a single classifier, such as the image-based example described previously, can have a number of different pixel resolutions defined and hence give rise to two, three or more distinct fully dependent classifiers. Combining such fully dependent and independent classifiers can easily lead to IDK cascades with eight or more components. In practice, however, n is unlikely to be greater than ten.

Problem statement: Given a collection of n classifiers $K_1, K_2, \ldots, K_n$ suitable for use on a given classification problem, the objective is to determine which of these classifiers should be executed, and in what order, such that the expected duration to

successfully classify the input is minimized. Stated otherwise, the aim is to obtain the *optimal IDK cascade*. Further, the problem may additionally be subject to the constraint that the maximum time taken to successfully classify the input must be no more than a specified deadline D.

For the expected duration of an IDK cascade to be finite, it is necessary that some classifier K_n with $P_n = 1$ is executed. We therefore assume that such a classifier exists, and refer to it as a *deterministic classifier*; by convention always denoted by K_n. Further, we assume that there is only one such deterministic classifier, since if there were more than one, the one with the shortest execution time should always be preferred and the others discarded. Similarly, we assume that the deterministic classifier has a longer execution time (C_n) than any of the IDK classifiers, since the deterministic classifier should always be preferred over any IDK classifier with the same or longer execution time; such IDK classifiers can therefore be discarded.

Combining probabilities: When computing the expected duration for an IDK cascade, it is essential to understand how the prior execution of one classifier K_A impacts the probability that a subsequent classifier K_B will be able to make a successful classification rather than return IDK. Consider two classifiers $K_A = (C_A, P_A)$ and $K_B = (C_B, P_B)$, neither of which is deterministic (i.e. $P_A < 1.0$ and $P_B < 1.0$). Suppose that classifier K_A is called on some input and returns IDK, and in that case classifier K_B is called next. Let $p(K_B|\overline{K_A})$ denote the *conditional probability* that classifier K_B returns an actual class rather than IDK given that classifier K_A failed to do so and therefore returned IDK.

Definition 1 (Independent classifiers) Classifiers K_A and K_B are said to be independent if $p(K_B|\overline{K_A}) = P_B$ and $p(K_A|\overline{K_B}) = P_A$.

That is, the conditional probability that K_B will make a successful classification given that K_A did not, is exactly the same as the probability that K_B will make a successful classification when K_A is not called beforehand, and vice versa. Informally speaking, this happens when K_A and K_B are making their classification decisions in very different ways, for example by using completely different (and uncorrelated) attributes of their inputs, and so the fact that one of the classifiers was unable to classify an input has no bearing on the ability of the other classifier to do so.

At the other extreme are *fully dependent* classifiers. Two classifiers are fully dependent if one successfully classifies only a strict subset of the set of inputs that are successfully classified by the other.

Definition 2 (Dependent classifiers) Classifiers K_A and K_B, satisfying ($P_A < P_B$), are said to be fully dependent if

$$p(K_A|\overline{K_B}) = 0$$

and

$$p(K_B|\overline{K_A}) = \frac{P_B - P_A}{1 - P_A}.$$

So if K_B cannot deliver a successful classification then neither can K_A; and the conditional probability that K_B will make a successful classification given that K_A did not, is given by the proportion of inputs, $P_B - P_A$, that K_B is able to classify that K_A cannot, divided by the proportion of inputs, $1 - P_A$, that K_A fails to classify. For example, if $P_A = 0.6$ and $P_B = 0.8$ then if K_A fails to make a classification then the probability of success for K_B falls from 0.8 to 0.5.

Observe that if the second classifier K_B is deterministic, and hence $P_B = 1$ then we have $p(K_B|\overline{K_A}) = \frac{P_B - P_A}{(1-P_A)} = 1 = P_B$. In other words, deterministic classifiers can be regarded as being either independent (Definition 1) or fully dependent (Definition 2).

Intuitively, IDK cascades can be constructed by placing less effective but faster classifiers earlier in the order, in the hope that they will successfully classify the input most of the time, with more effective but slower classifiers invoked only on those rare occasions that the earlier classifiers fail. This is illustrated by the following example.

Example 1 Suppose, for solving some classification problem, we have a deterministic classifier K_3 (we will add to this example later), with parameters $(C_3 = 10, P_3 = 1)$. Since the classifier is deterministic, it will be the last classifier to run in any IDK cascade in which it is included. Further, assume we also have an IDK classifier K_1 with parameters $(C_1 = 5, P_1 = 0.6)$.

Consider the IDK cascade $\langle K_1;K_3\rangle$, which executes K_1 first and subsequently executes K_3 only if K_1 fails to make a successful classification and returns IDK. Since K_1 is always executed on all inputs, but K_3 only executes when K_1 outputs IDK, which happens with probability $(1 - P_1)$, the expected duration of this IDK cascade is given by:

$$C_1 + (1 - P_1) \times C_3 = \mathbf{9}$$

which is smaller than $C_3 = \mathbf{1}0$, the duration of the cascade $\langle K_3\rangle$ containing only the deterministic classifier. □

The downside of using an IDK cascade rather than only executing the deterministic classifier is that the *worst-case* duration increases. In Example 1, while the IDK cascade $\langle K_1;K_3\rangle$ completes in 5 time units in 60% of cases, in the remaining 40% of cases it takes 15 time units, whereas executing only the deterministic classifier always takes 10 time units. Whether this matters or not depends on whether classification is required by a specified *deadline*. Such problems are considered in Sect. 6.

Example 1 considered the combination of one IDK classifier with a deterministic classifier. The problem of finding an optimal IDK cascade becomes much more interesting and challenging when multiple IDK classifiers are available. In that case, the analysis and algorithms required depend on the relationships between the IDK classifiers, i.e. whether they are independent (Sect. 3), fully dependent (Sect. 4), or consist of groups of dependent classifiers where classifiers from different groups are nevertheless independent of one another (Sect. 5).

Table 1 Parameters of independent IDK classifiers for Example 2

K_i	C_i	P_i
K_1	5	0.60
K_2	3	0.20
K_3	10	1.00

4 Independent IDK classifiers

In this section we consider the problem of determining the optimal IDK cascade, that minimizes the expected duration for successful classification, given a set of n *independent* IDK classifiers $\left\{K_i = (C_i, P_i)\right\}_{i=1}^{n}$. We begin with an illustrative example.

Example 2 Consider again the problem instance from Example 1. Suppose that there is an additional IDK classifier K_2 with parameters $(C_2 = 3, P_2 = 0.2)$ as set out in Table 1. Further, assume that K_2 is known to be independent of K_1. It can be verified that IDK cascade $\langle K_1;K_2;K_3\rangle$ has an expected duration of

$$5 + (1 - 0.6) \times 3 + (1 - 0.6)(1 - 0.2) \times 10 = 5 + 1.2 + 3.2 = \mathbf{9.4}$$

IDK cascade $\langle K_2;K_1;K_3\rangle$ has an expected duration of

$$3 + (1 - 0.2) \times 5 + (1 - 0.2)(1 - 0.6) \times 10 = 3 + 4 + 3.2 = \mathbf{10.2}$$

and IDK cascade $\langle K_2;K_3\rangle$ has an expected duration of

$$3 + (1 - 0.2) \times 10 = 3 + 8 = \mathbf{11}$$

Observe that each of these IDK cascades has an expected duration larger than that of the IDK cascade $\langle K_1;K_3\rangle$ that, as shown in Example 1, has an expected duration of **9**. Hence if minimizing expected duration is the objective, then the IDK classifier K_2 should not be used at all. □

Example 2 illustrates one way of obtaining optimal IDK cascades: simply enumerate all possibilities, compute the expected duration for each, and choose the IDK cascade with the minimum value. However, such an approach is highly inefficient: given n classifiers the number of possible IDK cascades is a very rapidly-growing combinatorial function[2] $\Theta(\sum_{k=0}^{n-1} \frac{(n-1)!}{(n-1-k)!})$. Below, we derive a far more efficient algorithm for synthesizing optimal IDK cascades from a given collection of independent IDK classifiers.

Lemma 1 can be used to compute the expected duration of any linear sequence of independent IDK classifiers.

[2] The deterministic classifier must always be included as the final classifier, and may be preceded by $k = 0 \ldots (n - 1)$ other IDK classifiers in any order, hence the formula follows from that for the arrangement (i.e. permutation without repetition) of k elements from a set of cardinality $n - 1$.

Lemma 1 *Consider an IDK cascade that orders n classifiers. Let* $(\hat{C}_k, \hat{P}_k)$ *denote the execution time and success probability of the k'th classifier in this cascade. The expected duration of the IDK cascade is given by*:

$$\sum_{k=1}^{n} \left(\prod_{j=1}^{k-1} (1 - \hat{P}_j) \right) \hat{C}_k \tag{1}$$

Proof The first classifier will always execute; the second classifier will execute if and only if the first one fails, which happens with probability $(1 - \hat{P}_1)$; the third classifier will execute if and only if the first two both fail, which happens with probability $(1 - \hat{P}_1)(1 - \hat{P}_2)$; and so on. Hence the expected duration is

$$\hat{C}_1 + (1 - \hat{P}_1)\,\hat{C}_2 + (1 - \hat{P}_1)(1 - \hat{P}_2)\,\hat{C}_3 + \cdots + \prod_{j=1}^{n-1} (1 - \hat{P}_j)\,\hat{C}_n$$

which is represented compactly by the expression in the Lemma. □

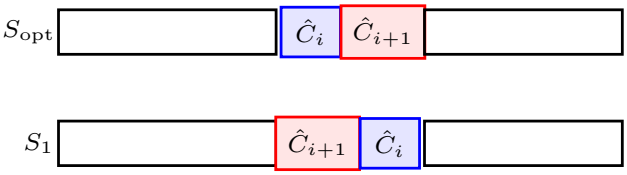
Lemma 2 below identifies, for independent IDK classifiers, an important characteristic of any optimal IDK cascade:

Lemma 2 *Let classifier* K_j *be scheduled for execution after classifier* K_i *in an optimal IDK cascade* (*i.e. of minimum expected duration*). *It must be the case that*:

$$\frac{C_i}{P_i} \leq \frac{C_j}{P_j} \tag{2}$$

Proof We will establish that any two adjacently scheduled classifiers K_i and K_j satisfy (2). The lemma then follows from the transitivity of the $\leq$ relationship on $\mathbb{R}$ (the set of real numbers).

Let S_{opt} denote an optimal IDK cascade, and let $\hat{K}_i$ denote the classifier in the i'th position in this cascade for any $i, 1 \leq i \leq n$. Further, let $(\hat{C}_i, \hat{P}_i)$ denote the execution time and success probability of classifier $\hat{K}_i$. Finally, let S_1 denote an IDK cascade obtained from S_{opt} by swapping the classifiers in the i'th and $(i + 1)$'th positions in S_{opt}:



Using Lemma 1, the expected duration of S_{opt} can be written as the sum of three terms representing respectively the outer summation of (1) for (i) $k \in \{1, \ldots, i-1\}$, (ii) $k \in \{i, i+1\}$, and (iii) $k \in \{i+2, \ldots, n\}$:

$$\begin{aligned}
&\sum_{k=1}^{i-1}\left(\prod_{j=1}^{k-1}(1-\hat{P}_j)\right)\hat{C}_k && \text{//The first } (i-1) \text{ classifiers}\\
&+\prod_{j=1}^{i-1}(1-\hat{P}_j)\left(\hat{C}_i+(1-\hat{P}_i)\hat{C}_{i+1}\right) && \text{//The next two classifiers}\\
&+\sum_{k=i+2}^{n}\left(\prod_{j=1}^{k-1}(1-\hat{P}_j)\right)\hat{C}_k && \text{//The remaining classifiers}
\end{aligned} \tag{3}$$

Next, consider the expected duration of S_1. Again using Lemma 1, the expected duration of S_1 can also be written as the sum of three terms. Since S_1 only differs from S_{opt} in that the i'th and $(i+1)$'th classifiers are swapped, the first and third terms are the same as the first and third terms of (3). However, the middle term is as follows, since the order of the i'th and $(i+1)$'th classifiers has been swapped.[3]

$$\prod_{j=1}^{i-1}(1-\hat{P}_j)\left(\hat{C}_{i+1}+(1-\hat{P}_{i+1})\hat{C}_i\right) \tag{4}$$

S_{opt} is by definition an optimal IDK cascade. Its expected duration is therefore no larger than the expected duration of S_1, and so the middle term of (3) must be no larger than (4):

$$\prod_{j=1}^{i-1}(1-\hat{P}_j)\left(\hat{C}_i+(1-\hat{P}_i)\hat{C}_{i+1}\right)\leq\prod_{j=1}^{i-1}(1-\hat{P}_j)\left(\hat{C}_{i+1}+(1-\hat{P}_{i+1})\hat{C}_i\right) \tag{5}$$

Observing that the term $\prod_{j=1}^{i-1}(1-\hat{P}_j)$ appears on both sides of (5), we have:

$$\begin{aligned}
\left(\hat{C}_i+(1-\hat{P}_i)\hat{C}_{i+1}\right)&\leq\left(\hat{C}_{i+1}+(1-\hat{P}_{i+1})\hat{C}_i\right)\\
&\Leftrightarrow\left(\hat{C}_i+\hat{C}_{i+1}-\hat{P}_i\hat{C}_{i+1}\right)\leq\left(\hat{C}_{i+1}+\hat{C}_i-\hat{P}_{i+1}\hat{C}_i\right)\\
&\Leftrightarrow\hat{P}_{i+1}\hat{C}_i\leq\hat{P}_i\hat{C}_{i+1}\\
&\Leftrightarrow\frac{\hat{C}_i}{\hat{P}_i}\leq\frac{\hat{C}_{i+1}}{\hat{P}_{i+1}}
\end{aligned}$$

□

[3] The difference between (4) and the middle term of (3) is that the roles of i and $(i+1)$ are swapped in the expression $\left(\hat{C}_x+(1-\hat{P}_x)\hat{C}_y\right)$.

Algorithm 1 Synthesizing an optimal IDK cascade from independent IDK classifiers

OptOrderIndependent($\{(C_1, P_1), (C_2, P_2), \ldots, (C_n, P_n)\}$)
1 Compute C_i/P_i for all $i, 1 \leq i \leq n$
2 Sort in non-decreasing order of C_i/P_i
3 Output the classifiers according to their position in the sorted list, stopping at the
4 deterministic classifier

Lemma 2 implies that independent IDK classifiers adjacent to each other in any optimal IDK cascade must have the ratio of their execution time to their success probability in non-decreasing order. An algorithm for synthesizing optimal IDK cascades immediately presents itself: simply determine these ratios for all of the classifiers, and sort the list in non-decreasing order—see Algorithm 1. As depicted, the algorithm need not enumerate any classifiers beyond the deterministic one, since such a classifier is guaranteed to complete successfully. Observe that Algorithm 1 is highly efficient, its run-time complexity is dominated by the sorting step, and is therefore $\Theta(n \log n)$.

5 Fully dependent IDK classifiers

In this section we consider the problem of determining an optimal IDK cascade, that minimizes the expected duration for successful classification, given a set of *n fully dependent* IDK classifiers $\left\{K_i = (C_i, P_i)\right\}_{i=1}^{n}$.

Since the classifiers are fully dependent, there is no benefit in executing a classifier with smaller probability P_j after one with a larger probability[4] $P_i > P_j$, since from Definition 2, we have $p(K_j|\overline{K_i}) = 0$. In order to minimize the expected duration we must therefore order whichever classifiers are used by their probability of successful classification, smallest first. Further, the deterministic classifier K_n must appear last, since otherwise there would be a non-zero probability of failing to classify some input, and hence the expected duration would no longer be finite.

Without loss of generality, we now assume that the dependent IDK classifiers are indexed according to strictly increasing values of their success probability, i.e. $P_{i+1} > P_i$ for all $i, 1 \leq i < n$, and $P_n = 1$, where K_n is the deterministic classifier.

We begin the analysis by building an illustrative example.

Consider two *fully dependent* IDK classifiers as as set out in Table 2: K_1 with parameters $(C_1 = 5, P_1 = 0.5)$ and K_2 with parameters $(C_2 = 9, P_2 = 0.8)$. Comparing classifiers K_1 and K_2, we say that K_2 is a more *powerful* classifier than K_1, since $P_2 > P_1$. If we were to schedule classifier K_1 to execute only after executing K_2 and

[4] We can assume that the P_i values are distinct, since if two are equal, we may discard the classifier with the larger value of C_i or make a arbitrary choice if both P_i and C_i are equal.

Table 2 Parameters of fully dependent IDK classifiers for Example 3

K_i	C_i	P_i
K_1	5	0.50
K_2	9	0.80
K_3	15	1.00

having K_2 output IDK, then the probability that K_1 would make a successful classification would be **0**, since all of the cases that it can correctly classify would have already been identified by K_1. Alternatively, if we execute K_1 first and only execute K_2 if K_1 fails (i.e. returns IDK), then the probability that K_2 will make a successful classification needs to account for the fact that we now know its input is not one that K_1 is able to classify. In the notation of conditional probability (see Definition 2 in Sect. 2), we can represent this as:

$$p(K_2|\overline{K_1}) = \frac{P_2 - P_1}{1 - P_1} = \frac{0.3}{0.5} = \mathbf{0.6}$$

where the numerator is the probability that an input to the IDK cascade $\langle K_1;K_2\rangle$ will be classified by K_2, and the denominator is the probability that an input to the IDK cascade $\langle K_1;K_2\rangle$ will not be classified by K_1. Next consider a further classifier K_3 added to the IDK cascade, i.e. $\langle K_1;K_2;K_3\rangle$. The probability that K_3 will be executed is given by the probability that both K_1 and then K_2 output IDK. That is:

$$(1 - P_1)(1 - p(K_2|\overline{K_1})) = (1 - P_1)\left(1 - \frac{P_2 - P_1}{1 - P_1}\right) = (1 - P_2)$$

As expected, this simply equates to the probability that K_2 alone is unable to make a classification. This is the case since K_1 effectively adds nothing to the set of inputs that can be classified by K_2.

Example 3 Consider the problem instance described above, where K_1, K_2, and K_3 are fully dependent classifiers, as set out in Table 2. Considering all of the possible IDK cascades where the classifiers run in index order, it can be verified that IDK cascade $\langle K_1;K_2;K_3\rangle$ has an expected duration of

$$5 + (1 - 0.5) \times 9 + (1 - 0.8) \times 15 = 5 + 4.5 + 3 = \mathbf{12.5}$$

IDK cascade $\langle K_1;K_3\rangle$ has an expected duration of

$$5 + (1 - 0.5) \times 15 = 5 + 7.5 = \mathbf{12.5}$$

and IDK cascade $\langle K_2;K_3\rangle$ has an expected duration of

$$9 + (1 - 0.8) \times 15 = 9 + 3 = \mathbf{12}$$

Finally, running only the deterministic classifier K_3, has an expected duration of **15**. Observe that the IDK cascade $\langle K_2;K_3\rangle$ is optimal, and has an expected duration

lower than that of any of the possible IDK cascades that include K_1. Hence if minimizing expected duration is the objective then K_1 should not be used. □

Example 3 again illustrates a simple but inefficient way of synthesizing optimal IDK cascades: enumerate all possible IDK cascades that could potentially be optimal, compute the expected duration for each, and choose the one with the minimum value. However, even though we know that dependent classifiers can only appear in an optimal IDK cascade ordered by increasing probability, the fact that we do not know which classifiers to include means that there are still an exponential number[5] of different IDK cascades to consider: $\Theta(2^{n-1})$, for n classifiers. Below, we derive a more efficient algorithm for synthesizing optimal IDK cascades from a given collection of fully dependent IDK classifiers. We begin with a definition.

Definition 3 (optimal sub-sequence $S(i)$) The optimal sub-sequence $S(i)$ is the sub-sequence of $\langle K_1, K_2, \ldots, K_i \rangle$ of minimum expected duration, with classifier K_i last. The sub-sequence may omit zero or more of the classifiers, with the exception of K_i, but otherwise retains the same (index) ordering. Let $f(i)$ denote the expected duration of this optimal sub-sequence $S(i)$.

Using the terminology introduced in Definition 3, the aim of determining an optimal IDK cascade equates to determining the optimal sub-sequence $S(n)$. We now describe how this may be achieved by inductively determining the optimal sub-sequences $S(1), S(2), \ldots, S(n)$ in order.

In order to determine $S(i)$ for $i \geq 1$, we observe that optimal sub-sequences satisfy the *optimal sub-structure property* (Cormen et al. 2009, p. 379): optimal solutions to any problem instance incorporate optimal solutions to sub-instances. Initially, we have:

$$S(0) = \langle\ \rangle, f(0) = 0, \text{ and } P_0 = 0.0.$$

Further, let K_h denote the classifier immediately preceding K_i in the optimal sub-sequence $S(i)$. It must be the case that $S(i)$ equates to the concatenation of K_i to the end of the optimal sub-sequence $S(h)$. Recalling that $f(h)$ denotes the expected duration of $S(h)$, we therefore have:

$$f(i) = \min_{0 \leq h < i} \left\{ f(h) + (1 - P_h) C_i \right\} \tag{6}$$

[5] As the order of the classifiers is known, the complexity derives from whether or not to select them. Since the deterministic classifier must be included, there are $n - 1$ classifiers each of which may or may not be included, leading to 2^{n-1} possibilities.

where $S(i)$ equates to the concatenation of K_i to the end of $S(\ell)$ where $\ell = \arg\min_{0 \le h < i} \left\{ f(h) + (1 - P_h) C_i \right\}$. (Note, if ℓ is not unique, then this implies that there are multiple optimal sub-sequences with the same minimum duration, and hence any such ℓ may be chosen). Example 4 illustrates how this approach works.

Example 4 Consider again the problem instance from Example 3 where K_1 and K_2 are two fully dependent IDK classifiers, with parameters $(C_1 = 5, P_1 = 0.5)$ and $(C_2 = 9, P_2 = 0.8)$, and K_3 is a deterministic classifier with parameters $(C_3 = 15, P_3 = 1.0)$, as set out in Table 2 .

Let us consider the minimum expected duration $f(i)$ of a sub-sequence ending with each of the three classifiers, i.e. $f(i)$ for $i = 1 \ldots 3$.

- Starting with classifier K_1, trivially we have $f(1) = C_1 = 5$ since no other classifiers can precede K_1, and therefore $S(1) = \langle K_1 \rangle$ is an optimal sub-sequence ending in K_1.
- Considering K_2, there are two possibilities: (i) K_2 is the only classifier in the sub-sequence, in which case $f(2) = C_2 = 9$; (ii) K_2 is preceded by K_1, in which case $f(2) = f(1) + (1 - P_1) \times C_2 = 5 + (1 - 0.5) \times 9 = 9.5$. This is the case because K_2 runs with a probability of $1 - P_1$, the probability that K_1 returned IDK. Since we seek the minimum value, we have $f(2) = 9$. As this minimum value is obtained by executing only K_2, $S(2) = \langle K_2 \rangle$ is an optimal sub-sequence ending in K_2.
- Finally considering K_3, there are three possibilities: (i) K_3 is the only classifier in the sub-sequence, in which case $f(3) = C_3 = 15$; (ii) K_3 is preceded by K_1, in which case $f(3) = f(1) + (1 - P_1) \times C_3 = 5 + (1 - 0.5) \times 15 = 12.5$; and (iii) K_3 is preceded by K_2, in which case $f(3) = f(2) + (1 - P_2) \times C_3 = 9 + (1 - 0.8) \times 15 = 12$. Since we seek the minimum value, $f(3) = 12$, and $S(3) = \langle K_2, K_3 \rangle$ is an optimal sub-sequence ending in K_3.

As any valid IDK cascade must end with the deterministic classifier, K_3 in this case, we observe that the optimal IDK cascade for this example is $S(3) = \langle K_2, K_3 \rangle$, with the minimum expected duration of $f(3) = 12$. □

Algorithm 2 Synthesizing an IDK cascade with minimum expected duration, from fully dependent classifiers

```
OptOrderDependent({(C_1, P_1), (C_2, P_2), ..., (C_n, P_n)})
 1  // Assume sorted in order of increasing P_i
 2  P_0 = 0 // A notational convenience
 3  f(0) = 0
 4  for i = 1 to n
 5        q(i) = arg min_{0≤h<i} {f(h) + (1 − P_h)C_i}
 6        f(i) = f(q(i)) + (1 − P_{q(i)})C_i
 7
 8  // Queue Q, initialized to contain the deterministic classifier
 9  Q = [K_n]
10  r = q(n)
11  do
12        Q.prepend(K_r)
13        r = q(r)
14  while (r ≠ 0)
15  // Q now comprises the optimal IDK cascade
16  Output Q
```

As shown in Algorithm 2, the recurrence in (6) can be evaluated via a loop iterating over the values of k from 1 to n, with the $\min_{0\leq h<i}\{\}$ term leading to an overall run-time complexity that is quadratic $\Theta(n^2)$, rather than exponential, in the number of classifiers.

6 Independent groups of fully dependent IDK classifiers

In this section, we broaden the scope to include both independent and fully dependent IDK classifiers. We consider the problem of determining an optimal IDK cascade, given multiple independent groups of fully dependent IDK classifiers,[6] and a single deterministic classifier, i.e. we have a collection:

$$\Gamma = \left\{K_i = (C_i, P_i)\right\}_{i=1}^{n}$$

of n IDK classifiers, such that K_n is a deterministic classifier, with $P_n = 1$, and the remaining classifiers are partitioned into $m < n$ groups $\Gamma_1, \Gamma_2, \ldots, \Gamma_m$ such that for any two distinct classifiers K_A and K_B:

– If they are in the same group then they are fully dependent with respect to one another as per Definition 2. Without loss of generality, we assume that such fully dependent classifiers are indexed in increasing order of their probabilities, P_i; and

[6] Groups of size one represent IDK classifiers that are independent of all other IDK classifiers.

– If they are in different groups then they are independent with respect to one another as per Definition 1.

From Definition 2, considering two fully dependent IDK classifiers K_A and K_B from the same group, with $P_B > P_A$, any sub-sequence of IDK classifiers where K_A occurs after K_B cannot be part of an optimal IDK cascade, since the probability of K_A making a classification when K_B did not is zero ($p(K_A|\overline{K_B}) = 0$). For the opposite ordering, with K_A first, the *conditional probability* that K_B will make a classification given that K_A did not is given by:

$$p(K_B|\overline{K_A}) = \frac{P_B - P_A}{1 - P_A} \tag{7}$$

Further, the probability $p(\overline{S})$ that a sub-sequence $S = \langle K_A, K_B \rangle$ of fully dependent IDK classifiers will fail to make a classification can be expressed in terms of conditional probabilities:

$$p(\overline{S}) = p(\overline{K_A})p(\overline{K_B}|\overline{K_A}) = (1 - P_A)\left(1 - \frac{P_B - P_A}{1 - P_A}\right) = (1 - P_B) \tag{8}$$

which equates to the probability that the most powerful of the dependent IDK classifiers will fail to make a classification.

Considering a sub-sequence $S = \langle K_A, K_I, K_B \rangle$, where K_A and K_B are fully dependent IDK classifiers in the same group, with K_B the more powerful classifier (i.e. $P_B > P_A$), and K_I is an independent IDK classifier (i.e. from a different group), then the probability that the sub-sequence will fail to make a classification is given by:

$$\begin{aligned} p(\overline{S}) &= p(\overline{K_A})p(\overline{K_I})p(\overline{K_B}|\overline{K_A}) \\ &= (1 - P_A)(1 - P_I)\left(1 - \frac{P_B - P_A}{1 - P_A}\right) \\ &= (1 - P_I)(1 - P_B) \end{aligned} \tag{9}$$

In general, considering an IDK cascade S composed of classifiers from independent groups of fully dependent IDK classifiers, the probability that the IDK cascade will fail to make a classification is given by:

$$p(\overline{S}) = \prod_{j \in Z} \left(1 - P_j\right) \tag{10}$$

where Z is a set containing for each group **only** the single most powerful IDK classifier from the group that is present in S.

We illustrate the analysis for a mix of IDK classifiers via Example 5, obtained from Example 4 by renaming the deterministic classifier to K_4, and adding an independent classifier K_3.

Table 3 Parameters of independent groups of dependent IDK classifiers for Example 5

K_i	C_i	P_i	Γ_i
K_1	5	0.50	Γ_1
K_2	9	0.80	Γ_1
K_3	8	0.75	Γ_2
K_4	15	1.00	

Example 5 Suppose that we have four IDK classifiers with parameters as set out in Table 3, where K_4 is the deterministic classifier and the other three IDK classifiers are partitioned into two groups as follows:

$$\Gamma_1 = \{K_1, K_2\}, \Gamma_2 = \{K_3\}$$

Since sub-sequences where K_2 appears before K_1 cannot be part of an optimal IDK cascade, we ignore such cases and enumerate all other IDK cascades. Similar to Example 4, it can be verified that the IDK cascade $\langle K_1;K_2;K_4\rangle$ has an expected duration of

$$5 + (1 - 0.5) \times 9 + (1 - 0.8) \times 15 = \mathbf{12.5}$$

IDK cascade $\langle K_1;K_4\rangle$ has an expected duration of

$$5 + (1 - 0.5) \times 15 = \mathbf{12.5}$$

and IDK cascade $\langle K_2;K_4\rangle$ has an expected duration of

$$9 + (1 - 0.8) \times 15 = \mathbf{12}$$

finally $\langle K_4\rangle$ has an expected duration of **15**.

Building upon the IDK cascades above and inserting K_3, it can be verified that IDK cascade $\langle K_3;K_1;K_2;K_4\rangle$ has an expected duration of

$$8 + (1 - 0.75) \times 5 + (1 - 0.75)(1 - 0.5) \times 9 + (1 - 0.75)(1 - 0.8) \times 15 = \mathbf{11.125}$$

IDK cascade $\langle K_1;K_3;K_2;K_4\rangle$ has an expected duration of

$$5 + (1 - 0.5) \times 8 + (1 - 0.5)(1 - 0.75) \times 9 + (1 - 0.75)(1 - 0.8) \times 15 = \mathbf{10.875}$$

IDK cascade $\langle K_1;K_2;K_3;K_4\rangle$ has an expected duration of

$$5 + (1 - 0.5) \times 9 + (1 - 0.8) \times 8 + (1 - 0.75)(1 - 0.8) \times 15 = \mathbf{11.85}$$

IDK cascade $\langle K_3;K_1;K_4\rangle$ has an expected duration of

$$8 + (1 - 0.75) \times 5 + (1 - 0.75)(1 - 0.5) \times 15 = \mathbf{11.125}$$

IDK cascade $\langle K_1;K_3;K_4\rangle$ has an expected duration of

$$5 + (1 - 0.5) \times 8 + (1 - 0.75)(1 - 0.5) \times 15 = \mathbf{10.875}$$

IDK cascade $\langle K3;K_2;K_4\rangle$ has an expected duration of

$$8 + (1 - 0.75) \times 9 + (1 - 0.75)(1 - 0.8) \times 15 = \mathbf{11}$$

IDK cascade $\langle K_2;K3;K_4\rangle$ has an expected duration of

$$9 + (1 - 0.8) \times 8 + (1 - 0.75)(1 - 0.8) \times 15 = \mathbf{11.35}$$

and finally, IDK cascade $\langle K3;K_4\rangle$ has an expected duration of

$$8 + (1 - 0.75) \times 15 = \mathbf{11.75}$$

Hence, there are two optimal IDK cascades, $\langle K_1;K_3;K_4\rangle$ and $\langle K_1;K_3;K_2;K_4\rangle$, both of which have an expected duration of **10.875**. □

Example 5 illustrates a crucial point regarding the difficulty of determining an optimal IDK cascade for independent groups of dependent IDK classifiers. For the sub-problem with only one fully dependent group and the deterministic classifier (i.e. K_1, K_2, and K_4) then the optimal IDK cascade is $\langle K_2;K_4\rangle$, yet once an extra group is added (effectively just the independent IDK classifier K_3) then the use of K_1 becomes essential to obtain an optimal IDK cascade, i.e. either $\langle K_1;K_3;K_4\rangle$ or $\langle K_1;K_3;K_2;K_4\rangle$. This shows that a divide-and-conquer approach, first determining the local solution for each group of dependent IDK classifiers and then using only those classifiers in the global solution, is not optimal.

Lemma 2 shows that when we consider only independent IDK classifiers, then an optimal IDK cascade can be obtained by considering the classifiers in order of the increasing ratio of their execution time to their success probability, which is simply C_i/P_i in that case. Further, when we consider only fully dependent IDK classifiers, then an optimal IDK cascade can be obtained by considering the classifiers in order of their increasing probability P_i, with the problem effectively reduced to determining which classifiers, if any, to omit.

The problem is however subtly different when we attempt to build an optimal IDK cascade from multiple independent groups of fully dependent classifiers. In this case, there is no single complete ordering of classifiers from which we can make a step-by-step selection of the ones to use. The reason for this is as follows. When selecting between two mutually independent classifiers K_2 and K_3 from two different groups ($\Gamma_1 = \{K_1, K_2\}, \Gamma_2 = \{K_3\}$) then the two classifiers should be ordered according to increasing values of the ratio of their execution time to their probability

of making a successful classification *of any as yet unclassified input*. (This ordering follows directly from the proof of Lemma 2 for independent classifiers). However, for an IDK classifier such as K_2 that is a member of a group of fully dependent IDK classifiers, this probability is a conditional one that depends on the previous classifier (e.g. K_1), if any, from the same group that appears in the IDK cascade. The probability of K_2 making a successful classification of any as yet unclassified input and hence its relative ordering with respect to K_3 can therefore depend on whether or not K_1 is present in the IDK cascade. Specifically, when $C_2/P_2 < C_3/P_3 < C_2/\left(\frac{P_2-P_1}{1-P_1}\right)$ then the ordering of K_2 and K_3 switches depending on whether or not K_1 is present in the IDK cascade. This issue is illustrated in Example 6.

Example 6 The parameters of the classifiers used in this example are as set out in Table 4. IDK classifiers K_1 and K_2, with parameters $(C_1 = 5, P_1 = 0.5)$ and $(C_2 = 9, P_2 = 0.8)$ respectively, are members of the same group $\Gamma_1 = \{K_1, K_2\}$. IDK classifier K_3, with parameters $(C_3 = 10, P_3 = 0.5)$, is a member of a separate group $\Gamma_2 = \{K_3\}$ and hence is independent of K_1 and K_2. Finally, K_4, with parameters $(C_4 = 20, P_4 = 1.0)$ is the deterministic classifier.

In attempting to build an optimal IDK cascade, we can apply the previously derived rules when selecting from classifiers that are dependent. Hence, if more than one dependent IDK classifier from the same group is present in the IDK cascade, then those classifiers must appear in order of their probability i.e. K_1 first then K_2, since $P_2 > P_1$.

Similarly, when selecting between classifiers that are independent of one another, those classifiers must appear in order of the ratio of their execution time to their probability of successfully classifying any as yet unclassified input, noting that this probability is conditional on the previous classifier, if any, from the same dependent group that already appears in the IDK cascade.

Table 4 Parameters of independent groups of dependent IDK classifiers for Example 6

K_i	C_i	P_i	Γ_i
K_1	5	0.50	Γ_1
K_2	9	0.80	Γ_1
K_3	10	0.75	Γ_2
K_4	20	1.00	

The probability of K_2 correctly classifying any as yet unclassified input is given by $P_2 = 0.8$ if K_1 does not appear in the IDK cascade, and by $p(K_2|\overline{K_1}) = \frac{0.8-0.5}{1-0.5} = 0.6$ if K_1 appears before K_2. Further, since $C_2/P_2 = 11.25$, $C_3/P_3 = 13.333$, and $C_2/\left(\frac{P_2-P_1}{1-P_1}\right) = 15$ then the relative order of K_2 and K_3 depends on whether or not K_1 is used. If K_1 is present, then including K_3 before K_2 will result in a lower overall expected duration, otherwise it is more effective to include K_2 before K_3. We enumerate only the IDK cascades relevant to this point. It can be verified that IDK cascade $\langle K_1;K_2;K_3;K_4\rangle$ has an expected duration of

$$5 + (1 - 0.5) \times 9 + (1 - 0.8) \times 10 + (1 - 0.8)(1 - 0.75) \times 20 = \mathbf{12.5}$$

Switching the order of K_2 and K_3, IDK cascade $\langle K_1;K_3;K_2;K_4\rangle$ has an expected duration of

$$5 + (1 - 0.5) \times 10 + (1 - 0.5)(1 - 0.75) \times 9 + (1 - 0.75)(1 - 0.8) \times 20 = \mathbf{12.125}$$

Omitting K_1, it can be verified that IDK cascade $\langle K_2;K_3;K_4\rangle$ has an expected duration of

$$9 + (1 - 0.8) \times 10 + (1 - 0.8)(1 - 0.75) \times 20 = \mathbf{12}$$

Switching the order of K_2 and K_3, IDK cascade $\langle K_3;K_2;K_4\rangle$ has an expected duration of

$$10 + (1 - 0.75) \times 9 + (1 - 0.75)(1 - 0.8) \times 20 = \mathbf{13.25}$$

Observe that when K_1 is included, it is more effective for K_2 to appear after K_3, whereas if K_1 is omitted, then it is more effective for K_2 to precede K_3, as in the optimal IDK cascade $\langle K_2;K_3;K_4\rangle$. □

The issues of inclusion (illustrated by Example 5) and ordering (illustrated by Example 6) make the problem of determining an optimal IDK cascade for multiple independent groups of fully dependent IDK classifiers much more difficult than the individual cases of all independent or all fully dependent classifiers addressed in Sects. 3 and 4.

Algorithm 3 Synthesizing an IDK cascade with minimum expected duration, from multiple independent groups of dependent classifiers

OPTORDERMIXED($\Gamma = \{(C_1, P_1), (C_2, P_2), \ldots, (C_n, P_n)\}$)

// Γ is partitioned into dependent groups of classifiers
Initialize B to an array of n bits
$minLength = \infty$
$minVec = [0, 0, \ldots, 0]$
// Cycle through all the possible values of $B[\ldots]$ that have ($B[n] == 1$) in any order
for $B = [0, 0, \ldots, 1]$ **to** $[1, 1, \ldots, 1]$
    tmp = COMPUTELENGTH(Γ, B)
    if ($tmp < minLength$)
        $minLength = tmp$
        $minVec = B$
return IDK cascade comprising the classifiers indicated by $minVec$, in the order
    they are marked as "used" by COMPUTELENGTH($\Gamma, minVec$)

COMPUTELENGTH(Γ, B)

// Determine the expected duration of the IDK cascade that includes only the
// classifiers K_i for which $B[i] == 1$
Initialize the IDK cascade to be empty, and mark all included classifiers as "unused"
$length = 0$
$prob = 1.0$
repeat
    for each dependent group with unused classifiers
        –Identify as a *candidate classifier* the least-powerful (i.e., smallest P_i) as yet unused classifier K_i from the group
        –Let P_j be the probability for the previous classifier K_j, if any, from the same group, that is also included according to $B[j]$, with $P_j = 0$ if there is no such classifier
        –Compute the conditional probability $\widehat{P}_i = \frac{P_i - P_j}{1 - P_j}$ for K_i using Definition 2
    Select the candidate classifier that has the minimum ratio of execution time to conditional probability. Let K_k denote this selected classifier, $\widehat{P}_k$ its conditional probability, and C_k its execution time
    Mark K_k as "used".
    $length = length + (C_k \times prob)$ // since K_k is executed with probability $prob$
    $prob = (prob \times (1 - \widehat{P}_k))$ // Since K_k fails with probability $(1 - \widehat{P}_k)$
until (deterministic classifier K_n has been selected)
return $length$

Algorithm 3 is a straightforward exponential time algorithm[7] that finds the optimal solution by explicitly considering all possibilities, in the following manner. Letting the n-bit array B denote a particular subset of the classifiers (with $B[i]$==1 indicating that the i'th classifier K_i is included in this subset and $B[i]$==0 indicating that it is not), the procedure OPTORDERMIXED(Γ) iterates through all 2^{n-1} subsets

[7] Although such an algorithm cannot be considered efficient, it is unlikely that $n > 20$ classifiers would be available for any given problem, and hence such an optimal algorithm may nevertheless prove useful in practice where the number of classifiers that could be used is limited.

of the classifiers in Γ that include the deterministic classifier K_n, in order to determine which has the smallest expected duration. The expected duration of the subset denoted by B is computed by procedure COMPUTELENGTH(Γ, B), by repeatedly:

- Determining the set of candidate classifiers that could be used next. This candidate set comprises the first (i.e. lowest power) as yet unused classifier from each group that is present in the trial set according to B. (Note, the deterministic classifier is considered to occupy a group on its own and is therefore always included as a candidate).
- Selecting from the candidate classifiers the one with the minimum ratio of its execution time to its conditional probability, and marking it as used.

It is evident that the running time of Algorithm 3 is $\Theta(2^{n-1}n^2)$, and hence exponential in the number of classifiers n.

Note that while Algorithm 3 provides an optimal solution to the simpler problems involving (i) only independent IDK classifiers (Sect. 3), and (ii) only fully dependent IDK classifiers (Sect. 4), it has higher complexity and hence is less efficient than Algorithms 1 and 2 that were specifically designed for those cases.

7 Timing constraints on classification

Previous sections considered the problem of determining the optimal IDK cascade, that minimizes the expected duration for successful classification, given a set of independent IDK classifiers (Sect. 3), a set of fully dependent IDK classifiers (Sect. 4), or independent groups of fully dependent IDK classifiers (Sect. 5). In each case there was no constraint placed on the maximum duration of the IDK cascade. In this section, we consider a variant of each of these problems in which a timing constraint is specified, thus the objective is to determine the optimal IDK cascade, that minimizes the expected duration for successful classification, subject to the constraint that the maximum duration of the IDK cascade does not exceed a specified hard deadline D. We suspect, but have not yet been able to prove, that the problem of determining an optimal IDK cascade, subject to a deadline constraint, is NP-hard in each of these three cases.

A problem instance is now specified as:

$$\left\langle \Gamma = \left\{K_i = (C_i, P_i)\right\}_{i=1}^{n}, D \right\rangle \tag{11}$$

where $K_1, K_2, \ldots, K_n$ are n IDK classifiers with K_n the deterministic classifier (i.e., $P_n = 1$), and $D \in \mathbb{N}$ is the specified deadline.

The *maximum duration* of an IDK cascade is simply the sum of the execution times (C_i) of all of the classifiers deployed in that cascade. Since we must ensure that classification is always completed successful within the deadline D, it follows that a problem instance is *feasible* if and only if $C_n \leq D$. In other words, if and only if the deterministic classifier has a duration that does not exceed the deadline. Other problem instances are *infeasible* and are not considered further.

7.1 Independent IDK classifiers with a deadline

In this subsection, we consider the problem of determining an optimal IDK cascade, subject to a deadline constraint, given a set of *n independent* IDK classifiers.

We observe that Lemma 2 (from Sect. 3) continues to hold irrespective of the presence of a deadline. Hence, adjacent classifiers in an optimal IDK cascade must satisfy the property that the ratio of their execution time to their success probability (i.e. C_i/P_i) is non-decreasing. We assume, without loss of generality, that the classifiers are indexed according to non-decreasing C_i/P_i; i.e., for all i we have:

$$\frac{C_i}{P_i} \leq \frac{C_{i+1}}{P_{i+1}}$$

This assumption can be realized for any problem instance by sorting, in $\Theta(n \log n)$ time. Since any optimal IDK cascade must always end with the deterministic classifier, we assume that any IDK classifier K_j that has a ratio of its execution time to its success probability that is greater than that of the deterministic classifier is discarded. (From Lemma 2, we know that any such IDK classifier cannot appear before the deterministic classifier in any optimal IDK cascade, and hence discarding it has no effect on optimality.) Hence we may potentially reduce the value of n, with K_n again representing the deterministic classifier.

Given a modified problem instance as specified above, we apply the technique of dynamic programming (Bellman 1957) to determine an optimal IDK cascade of minimum expected duration, subject to the constraint that the maximum duration does not exceed D. We begin with a definition.

Definition 4 Let $E(k, d)$ denote the minimum expected duration for the following sub-problem of the problem instance specified by (11)

$$\left\langle \left\{K_i = (C_i, P_i)\right\}_{i=k}^{n}, d \right\rangle$$

That is, only the classifiers $K_k, K_{k+1}, \ldots, K_n$ are available, and there is a sub-deadline of d.

Using the notation from Definition 4, the expected duration of the optimal IDK cascade for the complete problem instance is $E(1, D)$, i.e. where the deadline is D, and all n classifiers $K_1, K_2, \ldots, K_n$ are available.

In building a solution we first look at the sub-problem where only the deterministic classifier K_n is available, and compute the values of $E(n, d)$ for all values of d. We have:

$$E(n, d) = \begin{cases} \infty, & \text{if } d < C_n \\ C_n, & \text{otherwise (i.e. if } d \geq C_n) \end{cases} \tag{12}$$

In other words, if the deadline is smaller than the execution time C_n of the deterministic classifier K_n, then the sub-problem instance is infeasible, represented by an

expected duration of infinity. Otherwise, $d \geq C_n$ and so the optimal IDK cascade for the sub-problem comprises the deterministic classifier K_n, and hence has an expected duration of C_n.

Now, assuming that we have already determined the values of $E(k+1, d')$ for all d', we can compute the values of $E(k, d)$ for all d as follows:

$$E(k,d) = \min\Big(E(k+1,d), C_k + (1-P_k)\cdot E(k+1, d-C_k)\Big) \tag{13}$$

where

- The first term within the min() reflects the decision not to use classifier K_k, and hence the expected duration is equal to the minimum expected duration $E(k+1, d)$ using only the classifiers $K_{k+1}, K_{k+2}, \ldots, K_n$.
- The second term within the min() reflects the decision to use classifier K_k. In which case classifier K_k always executes, taking a time C_k, since according to Lemma 2 it precedes the remaining classifiers in an optimal IDK cascade. When classifier K_k executes, it fails to make a successful classification with a probability $(1 - P_k)$, and when this happens, the remainder of the IDK cascade is executed with a minimum expected duration of $E(k+1, d-C_k)$, since the classifiers remaining are $K_{k+1}, K_{k+2}, \ldots, K_n$, and the available time remaining is $d - C_k$.

Equation (12) can be used to determine $E(n, d)$ for all d. Having done so, repeated application of (13) can be used to determine $E(n-1, d)$, $E(n-2, d)$,..., $E(1, d)$, for all d, and hence obtain the value of $E(1, D)$, which as mentioned earlier is the expected duration of an optimal IDK cascade for the complete problem instance. Further, the optimal IDK cascade that has this duration can be deduced by observing which of the first or the second term in the min() is smaller each time that (13) is applied.

Example 7 Suppose that we have three independent IDK classifiers K_1, K_2, and K_3 with parameters $(C_1 = 1, P_1 = 0.4)$, $(C_2 = 3, P_2 = 0.9)$, and $(C_3 = 2, P_3 = 0.5)$ and a deterministic classifier K_4 with parameters $(C_4 = 10, P_4 = 1.0)$ as shown in Table 5, and a deadline $D = 13$. Note the classifiers are again ordered according to the ratio C_i/P_i as they must be in any optimal IDK cascade of independent IDK classifiers, with the deterministic classifier last.

Table 5 Parameters of independent IDK classifiers for Example 7

K_i	C_i	P_i
K_1	1	0.40
K_2	3	0.90
K_3	2	0.50
K_4	10	1.00

Table 6 Values of $E(k, d)$ for Example 7

d	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$E(4, d)$	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	10	10	10	10	10	10	10
$E(3, d)$	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	10	10	7	7	7	7	7
$E(2, d)$	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	10	10	7	4	4	3.7	3.7
$E(1, d)$	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	10	7	7	4	3.4	3.4	3.22

Let us consider the minimum expected duration $E(k, d)$ of an IDK cascade starting with each of the four classifiers, $E(k, d)$ for $k = 4 \ldots 1$, where the IDK cascade has a maximum length not exceeding d for values of $d = 0 \ldots D$. When no such IDK cascade is possible, then $E(k, d) = \infty$. The computed values of $E(k, d)$ are shown in Table 6 and are explained below. Note, the values of $E(k, d)$ are assumed to be ∞ for all negative values of d.

- Starting with $k = 4$, trivially we have $E(4, d) = \infty$ for $d < C_4 = 10$ and $E(4, d) = C_4 = 10$ for $d \geq C_4$ since the only available classifier is K_4.
- Considering $k = 3$: (i) If K_3 is not included in the IDK cascade, then $E(3, d) = E(4, d)$, (ii) otherwise K_3 is included, in which case $E(3, d) = C_3 + (1 - P_3) \times E(4, d - C_3)$. As we are interested in the minimum values, we take the minimum of $E(3, d)$ from the two cases for each value of d, thus $E(3, d) = \infty$ for $d < 10$ and $E(3, d) = 10$ for $10 \leq d < 12$ (from (i)), and $E(3, d) = C_3 + (1 - P_3) \times E(4, d - C_3) = 7$ for $d \geq 12$ (from (ii)).
- Considering $k = 2$: (i) If K_2 is not included in the IDK cascade, then $E(2, d) = E(3, d)$, (ii) otherwise K_2 is included, in which case $E(2, d) = C_2 + (1 - P_2) \times E(3, d - C_2)$. Again, taking the minimum of $E(2, d)$ from the two cases, we have $E(2, d) = \infty$ for $d < 10$, $E(2, d) = 10$ for $10 \leq d < 12$, and $E(2, d) = 7$ for $d = 12$ (from (i)), and $E(2, d) = 4$ for $12 < d \leq 14$ and $E(2, d) = 3.7$ for $d > 14$ (from (ii)).
- Finally, considering $k = 1$: (i) If K_1 is not included in the IDK cascade, then $E(1, d) = E(2, d)$, (ii) otherwise K_2 is included, in which case $E(1, d) = C_1 + (1 - P_1) \times E(2, d - C_1)$. Taking the minimum of $E(1, d)$ from the two cases, we have $E(1, d) = \infty$ for $d < 10$, $E(1, d) = 10$ for $d = 10$, and $E(1, d) = 4$ for $d = 13$ (from (i)), and $E(1, d)$ given by the values shown in Table 6 for the remaining values of d (from (ii)). Note that the value of $E(1, 12) = 7$ is obtained via both (i) and (ii).

As any valid IDK cascade must end with the deterministic classifier, K_4 in this case, the minimum expected duration for the optimal IDK cascade, subject to a deadline of $D = 13$, is given by $E(1, D) = E(1, 13) = 4$ for this example. Tracing back how that result was obtained, we see that it is for the IDK cascade $\langle K_2, K_4 \rangle$, with classifiers K_1 and K_3 unused. We note that the optimal IDK cascade is, as expected, dependent on the length of deadline permitted. If the deadline were longer, $D \geq 16$, then all four classifiers could be used with an expected duration of 3.22, whereas

with a shorter deadline of $D = 14$ or $D = 15$ the optimal IDK cascade would be $\langle K_1, K_2, K_4 \rangle$, with an expected duration of 3.4. For a shorter deadline of $D = 12$ the optimal IDK cascade would be either $\langle K_1, K_4 \rangle$ or $\langle K_3, K_4 \rangle$, with an expected duration of 7. Further, with a deadline of $D = 11$, only $\langle K_1, K_4 \rangle$ would be optimal with the same expected duration of 7. Finally, with a deadline of $D = 10$, it is only possible to use the deterministic classifier K_4 for an expected duration of 10. □

Algorithm 4 provides the pseudo-code implementation of the above method. The overall complexity of the algorithm is dominated by the nested **for** loops. The outer **for** loop executes n times and the inner one D times, hence the overall complexity is pseudo-polynomial $\Theta(nD)$ in the number of classifiers, assuming that they are pre-sorted, which can be done in $\Theta(n \log n)$ time.

Algorithm 4 Determining an optimal IDK cascade of independent IDK classifiers with minimum expected duration and bounded maximum duration

```
OptOrderMixed(⟨{(C_1, P_1), (C_2, P_2), ..., (C_n, P_n)}, D⟩)
// Input assumed be sorted according to C_i/P_i: C_i/P_i ≥ C_{i+1}/P_{i+1} for all i
// Also assumed that (i) P_n = 1.0; and (ii) C_n ≤ D, otherwise infeasible.
// Also assumed that P_i < 1 for all i < n.
E[(1, ..., n) × (0, ..., D)] of real numbers // Filled in using Dynamic Programming
// Initializing E[n, d] for all d (See (12))
for d = 0 to (C_n − 1)
    E[n, d] = ∞
for d = C_n to D
    E[n, d] = C_n
// Implementing the recurrence (See (13))
for k = (n − 1) downto 1
    for d = 1 to D
        // Compute E[k, d] according to (13)
        E[k, d] = E[k + 1, d] // Initialize E[k, d] to first term in the min() of (13)
        if (d ≥ C_k) // It is possible to execute K_k...
            // Expected duration if K_k is executed (second term in min() of (13))
            tmp = C_k + (1 − P_k) · E[k + 1, d − C_k]
            if (tmp < E[k, d]) // It is better to execute K_k...
                E[k, d] = tmp // Update E[k, d] to second term in min() of (13)
// Print the optimal schedule
d = D
for k = 1 to n − 1
    if (E[k, d] ≠ E[k + 1, d])
        // K_k must have been selected...
        print out K_k
        d = d − C_k
print out K_n
```

7.2 Fully dependent IDK classifiers with a deadline

In this subsection, we consider the problem of determining an optimal IDK cascade, subject to a deadline constraint, given a set of *n fully dependent* IDK classifiers.

Since the classifiers are fully dependent, we observe that irrespective of the presence of a deadline, there is no benefit in executing a classifier with smaller probability P_j after one with a larger probability $P_i > P_j$, since from Definition 2 (in Sect. 2), we have $p(K_j|\overline{K_i}) = 0$. We therefore assume, without loss of generality, that the classifiers are ordered according to increasing values of their probability P_i, with the deterministic classifier K_n with $P_n = 1$ last. This assumption can be realized for any problem instance by sorting, in $\Theta(n \log n)$ time.

Similar to the independent case considered in Sect. 6.1, we again apply dynamic programming to determine an optimal IDK cascade of minimum expected duration, subject to the constraint that the maximum duration does not exceed D. The following definition is analogous to Definition 4.

Definition 5 Let $F(k, d)$ denote the minimum expected duration of an IDK cascade ending with classifier K_k (and hence achieving a success probability equal to P_k) while also guaranteeing to meet a deadline d for the following sub-problem of the problem instance specified in (11)

$$\left\langle \left\{ K_i = (C_i, P_i) \right\}_{i=1}^{k}, d \right\rangle$$

That is, only the classifiers $K_1, K_2, \ldots, K_k$ are available, and there is a deadline of d.

Using the notation from Definition 5, the expected duration of the optimal IDK cascade for the complete problem instance specified in (11) is $F(n, D)$. In other words where all n classifiers $K_1, K_2, \ldots, K_n$ are available and the deadline is D.

In building a solution we first look at the sub-problem where only classifier K_1 is available, and compute the values of $F(1, d)$ for all values of d. We have:

$$F(1, d) = \begin{cases} \infty, & \text{if } d < C_1 \\ C_1 & \text{otherwise} \end{cases} \tag{14}$$

In other words, if the deadline d is smaller than the execution time C_1 of the classifier K_1, then the sub-problem instance is infeasible, represented by an expected duration of infinity. Otherwise, $d \geq C_1$ and so the optimal IDK cascade for the sub-problem comprises the classifier K_1, and hence has an expected duration of C_1.

Now, assuming that we have already determined the values of $F(k-1, d')$ for all d', we can compute the values of $F(k, d)$ for all d as follows:

$$F(k, d) = \begin{cases} \infty, & \text{if } d < C_k \\ \min\left(C_k, \min_{1 \leq i < k} \left\{ F(i, d - C_k) + (1 - P_i) \times C_k \right\} \right) & \text{otherwise} \end{cases} \tag{15}$$

where

Table 7 Parameters of fully dependent IDK classifiers for Example 8

K_i	C_i	P_i
K_1	1	0.50
K_2	3	0.80
K_3	6	0.99
K_4	8	1.00

- A deadline d that is smaller than the execution time C_k of the classifier K_k, required by the sub-problem instance to end the IDK cascade, indicates that the sub-problem is infeasible and so $F(k, d) = \infty$.
- The first term within the min() reflects the decision to use only classifier K_k, and hence the expected duration is equal to C_k.
- The second term within the min() reflects the decision to append classifier K_k to some IDK cascade that ends with a classifier K_i, where $i < k$, i.e. K_i appears earlier in the order of the classifiers than K_k, and hence $P_i < P_k$. Since classifier K_k, with execution time C_k, must be appended to this suffix IDK cascade, the deadline available for the suffix IDK cascade is reduced to $d - C_k$. The minimum expected duration of such a suffix IDK cascade is therefore given by $F(i, d - C_k)$. Further, as the classifiers are fully dependent and ordered by their increasing probabilities, the probability that the suffix IDK cascade ending in K_i will fail to make a successful classification and therefore K_k will need to execute is given by $(1 - P_i)$. Hence the overall minimum expected duration when classifier K_k is immediately preceded by K_i is given by $F(i, d - C_k) + (1 - P_i) \times C_k$. Since we are interested in the minimum expected duration of any IDK cascade ending in K_k with duration not exceeding d, we take the minimum over all possible values of i, i.e. $1 \leq i < k$.

We note that the recurrence in (15) can be evaluated within a nested loop, with the outer loop iterating over values of k from 1 to n, and the inner loop iterating over values of d from 0 to D (Table 7).

Example 8 Suppose that we have three fully dependent IDK classifiers K_1, K_2, and K_3, with parameters $(C_1 = 1, P_1 = 0.5)$, $(C_2 = 3, P_2 = 0.8)$, and $(C_3 = 6, P_3 = 0.99)$ respectively, and a deterministic classifier K_4 with parameters $(C_4 = 8, P_4 = 1.0)$, and a deadline $D = 16$. Note the classifiers are again ordered according to their probabilities, as they must be in any optimal IDK cascade of fully dependent classifiers, with the deterministic classifier last.

Let us consider the minimum expected duration $F(i, d)$ of an IDK cascade ending with each of the four classifiers, i.e. $F(i, d)$ for $i = 1 \ldots 4$, where the IDK cascade has a maximum length not exceeding d for values of $d = 0 \ldots D$. When no such IDK cascade is possible, then $F(i, d) = \infty$. The computed values of $F(i, d)$ are shown in

Table 8 Values of $F(i, d)$ for Example 8

d	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$F(1, d)$	∞	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
$F(2, d)$	∞	∞	∞	3	2.5	2.5	2.5	2.5	2.5	2.5	2.5	2.5	2.5	2.5	2.5	2.5	2.5
$F(3, d)$	∞	∞	∞	∞	∞	∞	6	4	4	4	3.7	3.7	3.7	3.7	3.7	3.7	3.7
$F(4, d)$	∞	∞	∞	∞	∞	∞	∞	∞	8	5	5	4.6	4.1	4.1	4.1	4.08	4.08

Table 8 and are explained below. Note, the values of $F(i, d)$ are assumed to be ∞ for all negative values of d.

- Starting with classifier K_1, trivially we have $F(1, d) = \infty$ for $d < C_1 = 1$ and $F(1, d) = C_1 = 1$ for $d \geq C_1$ since no other classifiers can precede K_1 in an optimal IDK cascade.
- Considering K_2, there are two possibilities: (i) K_2 is the only classifier in the cascade, in which case $F(2, d) = C_2 = 3$ for $d \geq C_2$ and $F(2, d) = \infty$ otherwise; (ii) K_2 is preceded by K_1, in which case $F(2, d) = F(1, d - C_2) + (1 - P_1)C_2$, since for K_2 to run after K_1 and finish by d, K_1 must finish by $d - C_2$. As we are interested in the minimum values, we take the minimum of $F(2, d)$ from the two cases for each value of d, thus $F(2, d) = \infty$ for $d < 3$ and $F(2, 3) = C_2 = 3$ (from (i)), and $F(2, d) = F(1, d - C_2) + (1 - P_1)C_2 = 2.5$ for $d \geq 4$ (from (ii)).
- Considering K_3, there are three possibilities: (i) K_3 is the only classifier in the cascade, in which case $F(3, d) = C_3 = 6$ for $d \geq C_3$ and $F(3, d) = \infty$ otherwise; (ii) K_3 is preceded by K_1, in which case $F(3, d) = F(1, d - C_3) + (1 - P_1)C_3$; (iii) K_3 is preceded by K_2, in which case $F(3, d) = F(2, d - C_3) + (1 - P_2)C_3$. As we are interested in the minimum values, we take the minimum of $F(3, d)$ from the three cases for each value of d, thus $F(3, d) = \infty$ for $d < C_3 = 6$ and $F(3, 6) = C_3 = 6$ (from (i)), $F(3, d) = F(1, d - C_3) + (1 - P_1)C_3 = 4$ for $d \in (7, 8, 9)$ (from (ii)), and $F(3, d) = F(2, d - C_3) + (1 - P_2)C_3 = 3.7$ for $d \geq 10$ (from (iii)).
- Finally, considering K_4, there are four possibilities: (i) K_4 is the only classifier in the cascade, in which case $F(4, d) = C_4 = 8$ for $d \geq C_4$ and $F(4, d) = \infty$ otherwise; (ii) K_4 is preceded by K_1, in which case $F(4, d) = F(1, d - C_4) + (1 - P_1)C_4$; (iii) K_4 is preceded by K_2, in which case $F(4, d) = F(2, d - C_4) + (1 - P_2)C_4$; (iv) K_4 is preceded by K_3, in which case $F(4, d) = F(3, d - C_4) + (1 - P_3)C_4$. As we are interested in the minimum values, we take the minimum of $F(4, d)$ from the four cases for each value of d, thus $F(4, d) = \infty$ for $d < C_4 = 8$ and $F(4, 8) = C_4 = 8$ (from (i)), $F(4, d) = F(1, d - C_4) + (1 - P_1)C_4 = 5$ for $d \in (9, 10)$ (from (ii)), $F(4, d) = F(2, d - C_4) + (1 - P_2)C_4$ for $d \in (11, 12, 13, 14)$ (from (iii)), and $F(4, d) = F(3, d - C_4) + (1 - P_3)C_4 = 4.08$ for $d \geq 15$ (from (iv)).

As any valid IDK cascade must end with the deterministic classifier, K_4 in this case, the minimum expected duration for the optimal IDK cascade is given by $F(n, D) = f(4, 16) = 4.08$ for this example. Tracing back how that result was

obtained, we see that it is for the IDK cascade $\langle K_1, K_3, K_4 \rangle$, with K_2 unused. We note that the optimal IDK cascade is, as expected, dependent on the length of deadline permitted. If the deadline were longer, $D \geq 18$, then all four classifiers could be used with an expected duration of 3.78, whereas with a shorter deadline of $D = 12$ the optimal IDK cascade would be $\langle K_1, K_2, K_4 \rangle$, with K_2 unused, and an expected duration of 4.1. Further, with a deadline of $D = 11$, only $\langle K_2, K_4 \rangle$ would be used for an expected duration of 4.6, and with a deadline of 9 or 10, only $\langle K_1, K_4 \rangle$ would be used for an expected duration of 5. □

Algorithm 5 Determining an optimal IDK cascade of fully dependent IDK classifiers with minimum expected duration and bounded maximum duration

```
OptOrderDependentDeadline(⟨{(C_1, P_1), (C_2, P_2), ..., (C_n, P_n)}, D⟩)
// Input assumed to be sorted in order of increasing P_i
// Also assumed that C_n ≤ D, otherwise the problem instance is infeasible.
// F[k, d] is the minimum expected duration of an IDK cascade made from classifiers
// K_1, K_2, ..., K_k, with K_k as the last classifier, that has a worst-case duration ≤ d.
// Classifier K_{G[k,d]} is the next to last classifier in this IDK cascade.
F[(1, ..., n) × (0, ..., D)] of real numbers
G[(1, ..., n) × (0, ..., D)] of integers
// Initialize F[1, d] for all d (See (14))
for d = 0 to (C_1 − 1)
    F[1, d] = ∞
for d = C_1 to D
    F[1, d] = C_1
// Implement the recurrence (See (15))
for k = 2 to n
    for d = 1 to C_k − 1
        F[k, d] = ∞
    for d = C_k to D
        F[k, d] = C_k
        G[k, d] = k
        // Implement the min within the recurrence (See (15))
        for i = 1 to (k − 1)
            if (F[k, d] > F[i, d − C_k] + (1 − P_i) × C_k)
                // Expected duration reduced by using K_i as the next to last
                // classifier
                F[k, d] = F[i, d − C_k] + (1 − P_i) × C_k
                G[k, d] = i
// Print the optimal IDK cascade (in reverse order)
d = D
k = n
repeat
    print out K_k
    d' = d − C_k
    k = G[k, d]
    d = d'
until (k == G[k, d])
```

Algorithm 5 provides the pseudo-code implementation of the above method. The overall complexity of the algorithm is dominated by the three nested **for** loops, hence the overall complexity is pseudo-polynomial $\Theta(n^2 D)$ in the number

of classifiers, assuming that they are pre-sorted, which can be done in $\Theta(n \log n)$ time.

7.3 Independent groups of fully dependent IDK classifiers with a deadline

In this subsection, we consider the problem of determining an optimal IDK cascade, subject to a deadline constraint, given multiple independent groups of fully dependent IDK classifiers,[8] and a single deterministic classifier.

In Sects. 6.1 and 6.2, we used dynamic programming techniques to extend the polynomial time methods for obtaining optimal IDK cascades for (i) independent IDK classifiers (Sect. 3) and (ii) fully dependent IDK classifiers (Sect. 4) into pseudo-polynomial methods for solving these problems subject to a hard deadline constraint. However, for the problem considered in this subsection, we take a different approach.

In Sect. 5 we presented the method described by Algorithm 3 for obtaining optimal IDK cascades given multiple independent groups of fully dependent IDK classifiers. Recall that Algorithm 3 examines each possible subset of classifiers that could potentially comprise the optimal IDK cascade. This is controlled via an n-bit array B that is used to indicate which of the n classifiers will be included in a trial solution, i.e. $B[j]==1$ indicates that the j-th classifier is included and $B[j]==0$ indicates that it is not. Note, since the deterministic classifier is always included, $B[n]==1$.

Algorithm 3 is easily adapted to also account for a specified hard deadline D: we can simply discard any trial solution where the sum of the execute times of the classifiers used exceeds the deadline, i.e. $\sum_{\forall k:B[k]=1} C_k > D$. Since the deadline feasibility of each of the 2^{n-1} trial solutions can be evaluated in $\Theta(n)$ time, the overall complexity remains $\Theta(2^{n-1}n^2)$. Below we provide an illustrative example.

Example 9 The parameters of the classifiers used in this example are given in Table 9, and are the same as those used in Example 6 in Sect. 5. IDK classifiers K_1 and K_2, with parameters $(C_1 = 5, P_1 = 0.5)$ and $(C_2 = 9, P_2 = 0.8)$ respectively, are members of the same group $\Gamma_1 = \{K_1, K_2\}$. IDK classifier K_3, with parameters $(C_3 = 10, P_3 = 0.5)$, is a member of a separate group $\Gamma_2 = \{K_3\}$ and hence is independent of K_1 and K_2. Finally, K_4, with parameters $(C_4 = 20, P_4 = 1.0)$ is the deterministic classifier.

Assuming a deadline $D = 36$, then trial solutions $\{K_1, K_2, K_3, K_4\}$ and $\{K_2, K_3, K_4\}$ are infeasible, since they have a maximum duration of 44 and 39 respectively.

Table 9 Parameters of independent groups of dependent IDK classifiers for Example 9

K_i	C_i	P_i	Γ_i
K_1	5	0.50	Γ_1
K_2	9	0.80	Γ_1
K_3	10	0.75	Γ_2
K_4	20	1.00	

[8] Recall that groups of size one represent IDK classifiers that are independent of all other IDK classifiers.

Examining all other trial solutions, it can be verified that the optimal IDK cascade compliant with the deadline is $\langle K_1, K_3, K_4 \rangle$ with an expected duration of

$$5 + (1 - 0.5) \times 10 + (1 - 0.5)(1 - 0.75) \times 20 = \mathbf{12.5}$$

and a maximum duration of 35. □

The Algorithms 3, 4, and 5 that are used to determine optimal IDK cascades with a bounded maximum duration have complexity of $\Theta(2^{n-1}n^2)$, $\Theta(nD)$, and $\Theta(n^2D)$ respectively. In cases where the run time of these algorithms becomes prohibitive, due to large numbers of classifiers or long deadlines, then heuristic methods could potentially be employed, as discussed in Baruah et al. (2021). Such heuristics are not explored further here.

8 Context and conclusions

Learning-enabled components, particularly those based on Deep Neural Networks (DNNs), are increasingly being used in safety-critical real-time systems. It is imperative that the real-time scheduling theory community respond to this development by coming up with appropriate techniques to enable the offline analysis of systems that use such components.

In this work, we have adapted and applied algorithmic techniques from real-time scheduling theory to a proposed DNN use-case (Wang et al. 2018) that seeks to strike a balance between accuracy and timeliness by arranging individual DNN-based classifiers, augmented by the ability to classify inputs as belonging to an additional IDK class, into IDK cascades. The intuition behind the design of such IDK cascades is simple yet elegant: if the earlier classifiers in an IDK cascade can successfully identify simple-to-classify inputs, then later more sophisticated and hence slower classifiers need only be invoked rarely on truly challenging inputs. We were able to formalize this intuition, and from that develop algorithms that synthesize optimal IDK cascades from a given set of classifiers, both when the sole objective is optimizing expected duration and when there is additionally a hard deadline constraint. We were able to provide optimal algorithms for collections of IDK classifiers that are respectively, (i) independent, (ii) fully dependent, and (iii) a mix of the two.

The research presented in this paper can be viewed as a proof of concept of the principle that real-time scheduling can contribute to better design of real-time systems that use learning-enabled components, and it behoves us, as a community, to take a closer look at such systems. Future research into the scheduling of IDK classifiers could investigate the complexity of the problem from a theoretical perspective. For example, is the problem of determining an optimal IDK cascade with a deadline constraint NP-hard? Are there more efficient solutions to the problem of determining an optimal IDK cascade with independent groups of dependent classifiers with no deadline constraint than the one presented in this paper? Finally, this paper considered only the sequential execution of IDK classifiers, effectively the single processor case. There is a potentially rich area of associated research that can

be undertaken focusing on similar problems in the multiprocessor case, where IDK classifiers can execute in parallel.

References

Baruah SK, Burns A, Wu Y (2021) Optimal synthesis of IDK-cascades. In: Queudet A, Bate I, Lipari G (eds) RTNS'2021: 29th International Conference on Real-Time Networks and Systems, Nantes, France, April 7–9, 2021, ACM, pp 184–191, https://doi.org/10.1145/3453417.3453425,

Bellman R (1957) Dynamic programming, 1st edn. Princeton University Press, Princeton, NJ, USA

Cormen TH, Leiserson CE, Rivest RL, Stein C (2009) Introduction to Algorithms, 3rd edn. MIT Press, http://mitpress.mit.edu/books/introduction-algorithms

Fujiyoshi H, Hirakawa T, Yamashita T (2019) Deep learning-based image recognition for autonomous driving. IATSS Research 43(4):244–252 https://doi.org/10.1016/j.iatssr.2019.11.008, https://www.sciencedirect.com/science/article/pii/S0386111219301566

Hu Y, Liu S, Abdelzaher TF, Wigness M, David P (2021) On exploring image resizing for optimizing criticality-based machine perception. In: 27th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2021, Houston, TX, USA, August 18–20, 2021, IEEE, pp 169–178, https://doi.org/10.1109/RTCSA52859.2021.00027,

Khani F, Rinard MC, Liang P (2016) Unanimous prediction for 100% precision with application to learning semantic mappings. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7–12, 2016, Berlin, Germany, Volume 1: Long Papers, The Association for Computer Linguistics, https://doi.org/10.18653/v1/p16-1090,

Madani O, Georg M, Ross DA (2012) On using nearly-independent feature families for high precision and confidence. In: Hoi SCH, Buntine WL (eds) Proceedings of the 4th Asian Conference on Machine Learning, ACML 2012, Singapore, Singapore, November 4–6, 2012, JMLR.org, JMLR Proceedings, vol 25, pp 269–284, http://proceedings.mlr.press/v25/madani12.html

Madani O, Georg M, Ross DA (2013) On using nearly-independent feature families for high precision and confidence. Mach Learn 92(2–3):457–477. https://doi.org/10.1007/s10994-013-5377-0

Madras D, Pitassi T, Zemel RS (2018) Predict responsibly: Improving fairness and accuracy by learning to defer. In: Bengio S, Wallach HM, Larochelle H, Grauman K, Cesa-Bianchi N, Garnett R (eds) Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3–8, 2018, Montréal, Canada, pp 6150–6160, https://proceedings.neurips.cc/paper/2018/hash/09d37c08f7b129e96277388757530c72-Abstract.html

Nilsson N (1965) Learning machines. McGraw-Hill, New York

Oza NC, Tumer K (2008) Classifier ensembles: select real-world applications. Inf Fusion 9(1):4–20. https://doi.org/10.1016/j.inffus.2007.07.002

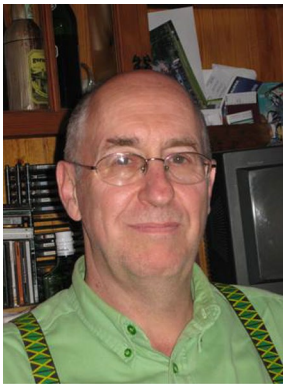
Russakovsky O, Deng J, Su H, Krause J, Satheesh S, Ma S, Huang Z, Karpathy A, Khosla A, Bernstein MS, Berg AC, Fei-Fei L (2015) Imagenet large scale visual recognition challenge. Int J Comput Vis 115(3):211–252. https://doi.org/10.1007/s11263-015-0816-y

Trappenberg TP, Back AD (2000) A classification scheme for applications with ambiguous data. In: Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks, IJCNN 2000, Neural Computing: New Challenges and Perspectives for the New Millennium, Como, Italy, July 24–27, 2000, Volume 6, IEEE Computer Society, pp 296–301, https://doi.org/10.1109/IJCNN.2000.859412,

Wang X, Luo Y, Crankshaw D, Tumanov A, Yu F, Gonzalez JE (2018) IDK cascades: Fast deep learning by learning not to overthink. In: Globerson A, Silva R (eds) Proceedings of the Thirty-Fourth Conference

on Uncertainty in Artificial Intelligence, UAI 2018, Monterey, California, USA, August 6–10, 2018, AUAI Press, pp 580–590, http://auai.org/uai2018/proceedings/papers/212.pdf

Sanjoy Baruah is the Hugo F. & Ina Champ Urbauer Professor of Computer Science and Engineering at Washington University in Saint Louis. His research interests and activities are in real-time and safety-critical system design, scheduling theory, and resource allocation and sharing in distributed computing environments.



Alan Burns holds a Personal Chair in the Department of Computer Science at the University of York in the UK. He is a Fellow of the Royal Academy of Engineering and a Fellow of the IEEE. He has chaired the IEEE Technical Committee on Real-Time Systems and received their "Outstanding Technical Achievement and Leadership Award" in 2006. His research interests include real-time system scheduling, mixed-criticality systems and programming languages.

Robert I. Davis is a Reader in the Real-Time Systems Research Group at the University of York, UK. Robert received his PhD in Computer Science from the University of York in 1995. Since then he has founded three start-up companies, all of which have succeeded in transferring real-time systems research into commercial products. Robert's research interests include real-time scheduling and schedulability analyses for single-core, multi-core, and networked systems.

Yue Wu is a graduate student in the Department of Mathematics at the University of Washington, Seattle, where she is working towards a PhD in Mathematics. She holds a bachelor's degree in computer science from Washington University in St. Louis, with a second major in mathematics. Her current research interests are in optimization and theoretical computer science.