Efficient tuning of kNN hyperparameters for indoor positioning with N-TBEA

Raul Montoliu*, Antoni Pérez-Navarro†, Joaquín Torres-Sospedra‡
* *Institute of New Imaging Technologies*, *Jaume I University*, Castellón, Spain
† *Faculty of Computer Sciences, Multimedia and Telecommunication*, *Universitat Oberta de Catalunya*, Barcelona, Spain
‡ *Algoritmi Research Centre*, *University of Minho*, Guimarães, Portugal

***Abstract*—Machine Learning is a very popular approach for indoor positioning. However, most of models rely on a set of hyperparameters, which need to be properly set. When the number of hyperparameters is large, exploring all the combinations of values (what is known as *brute force*) can be computationally prohibitive, especially in those cases where the training or operational time is high, such as in the kNN algorithm in fingerprint-based indoor positioning. This paper introduces *N-Tuple Bandit Evolutionary Algorithm* (N-TBEA) to find the hyperparameters in this last case. N-TBEA is an efficient exploration technique which evaluates the feasibility of similar combinations of parameters. The results show that N-TBEA can provide a solution with an accuracy similar to the best combination of parameters retrieved using *brute force*, which shows the potential of N-TBEA to be used in other advanced machine learning models.**

***Index Terms*—Indoor positioning; Machine learning; Optimum parameter selection.**

I. Introduction

Localization is enhancing the capabilities of wearables by helping users navigate or providing clues about users' interaction [1]. Many of these applications make sense in indoor spaces since humans tend to be 90% of their time indoors. However, we are still far from having an indoor positioning similar to what Global Navigation Satellite Systems (GNSS) are performing outdoors. Nevertheless, there is an indoor positioning solution that has become very popular: *fingerprinting*.

A fingerprint is a set of signal intensities received at a given position. The fundamental hypothesis is that two similar *fingerprints* in the feature space are assumed to be close in physical space, enabling indoor position estimation. Fingerprinting consists of two steps: an offline (or training) phase where the radio map is created with samples collected at well-known points over the operational area, and an online (or operational) phase where a matching algorithm is used to compare the fingerprint with the radio map and estimate the final position. Among many possibilities, the k nearest neighbors (kNN) algorithm is possibly the most used algorithm in the operational phase [2, 3, 4, 5, 6, 7].

Usually, the performance of the kNN algorithm is assessed by calculating the averaged positioning error over a test set, a set of fingerprints that simulates the *online* phase [2, 8]. Some recent works go a step further and explore the evaluation over multiple scenarios [9, 10, 11].

This work has been partially funded by the following projects: RTI2018-095168-B-C53; H2020-MSCA-IF 101023072; CYTED Network GeoLibero

The kNN has three major drawbacks: first, its computational cost is very high in terms of vector distance calculations. It is $O(n_{tr} \times n_{ts})$, where n_{tr} is the number of training samples (radio map), and n_{ts} the number of samples in the test set (operational phase); second, its accuracy strongly depends on the hyperparameters chosen [12], which in the case of kNN include the value of k, the distance function used to compare two fingerprints and the RSSI representation; and third, the best hyperparameters combination depends on every single scenario and environment [13].

To get the best combination of hyperparameters, one possibility is to use the inefficient algorithm known as *Brute Force* (BF) to evaluate all the possible combinations of hyperparameters. However, the computational load required to extract the optimal hyperparameter values might be excessive, especially in big datasets. An alternative is to fix some hyperparameters and set the best values in two or more stages [12]. Although this strategy provides reasonable results, several combinations of hyperparameters are not explored.

In this paper, we propose to use the *N-Tuple Bandit Evolutionary Algorithm* (N-TBEA) [14] to efficiently obtain the best combination of hyperparameters in fingerprint-based positioning. N-TBEA combines evolutionary search techniques with the use of a data structure, widely used in the field of game theory, composed of multiple multi-arm bandits. Its main advantage is that it is able to efficiently explore the search space, automatically evaluating the most promising hyperparameter configuration instead of using a predefined strategy to manually decide which are the ones to be explored. N-TBEA is able to reach solutions close to the optimum with a small number of iterations and, therefore, in considerably less time than the BF method.

The main contribution of this article is the use of the N-TBEA algorithm to efficiently obtain a good combination of hyperparameters of the kNN algorithm for fingerprinting. In particular, 4 hyperparameters have been tested: 1) 10 different values for k ($[1, 3, \dots, 19]$); 2) 3 ways of calculating the position when $k > 1$; 3) 4 functions to estimate the distance between two fingerprints (Euclidean, City block, Sorensen and Neyman); and 4) 4 different data representation from raw RSSI values (Positive, Normalized, Exponential and Powed). Thus, we have $10 \times 3 \times 4 \times 4 = 480$ possible combinations of the hyperparameters. To the best of our knowledge, this is the first work using N-TBEA for indoor positioning models.

II. Materials and Methods

A. Fingerprinting-based indoor positioning

As previously said, fingerprinting has two phases: *offline* and *online*. We explain the fundamentals of both phases here.

1) Offline (or training) phase: Let's suppose there are n_q different access points (APs) in the scenario where the indoor positioning system is going to be deployed. The set of all APs is $\Omega = \{\omega_1, \ldots, \omega_{n_q}\}$.

The training database (or *radio map*) $\mathcal{D}_{tr}$ is made up of a set of *fingerprint* and their positions:

$$\mathcal{D}_{tr} = \{\mathcal{F}_{tr}, \mathcal{L}_{tr}\} \tag{1}$$

where tr indicates training, $\mathcal{F}_{tr}$ is the set of *fingerprints* obtained and $\mathcal{L}_{tr}$ their positions. The set $\mathcal{F}_{tr}$ is made up of n_{tr} *fingerprints*, saved as n_{tr} vectors of RSSI measures:

$$\mathcal{F}_{tr} = \left\{\lambda_1^{tr}, ..., \lambda_{n_{tr}}^{tr}\right\} \tag{2}$$

and each *fingerprint* has n_q RSSI values:

$$\lambda_i^{tr} = \left\{\rho_{1,i}^{tr}, ..., \rho_{n_q,i}^{tr}\right\}, \ i \in [1, ..., n_{tr}] \tag{3}$$

The set $\mathcal{F}_{tr}$ can be viewed, from the point of view of machine learning algorithms, as a training database where the rows are the samples and the columns are the features.

On the other hand, $\mathcal{L}_{tr}$ is made up of n_{tr} positions, stored as vectors, representing the position associated with each sample:

$$\mathcal{L}_{tr} = \left\{\tau_1^{tr}, ..., \tau_{n_{tr}}^{tr}\right\} \tag{4}$$

where each position is often provided with just a pair of coordinates (often longitude and latitude). In complex large environments the position may also include the building, floor, or room labels. The coordinates and the other labels, ν, give the number of dimensions, n_l, of the position vector, τ, then:

$$\tau_i^{tr} = \left\{\nu_{1,i}^{tr}, ..., \nu_{n_l,i}^{tr}\right\}, \ i \in [1, ..., n_{tr}] \tag{5}$$

Note that, $\rho_{r,i}^{tr}$ ($r \in [1, \ldots, n_q]$) is the RSSI value obtained for AP ω_r at position τ_i^{tr}, i.e. $\rho_{r,i}^{tr} = RSSI(\omega_r, \tau_i^{tr})$.

2) Online (operational/test) phase: In this phase, the device captures a *fingerprint* λ_j^{ts} (in this case ts indicates test) that will be compared with the *radio map* $\mathcal{D}_{tr}$ to generate an estimate of its position τ_j^{ts}. If we assume that n_{ts} *fingerprints* are obtained, the test database can be defined as:

$$\mathcal{D}_{ts} = \{\mathcal{F}_{ts}, \mathcal{L}_{ts}\} \tag{6}$$

where

$$\mathcal{F}_{ts} = \left\{\lambda_1^{ts}, ..., \lambda_{n_{ts}}^{ts}\right\} \tag{7}$$

$$\mathcal{L}_{ts} = \left\{\tau_1^{ts}, ..., \tau_{n_{ts}}^{ts}\right\} \tag{8}$$

In order to be able to compare a test sample with the training ones, the number of RSSI values of the respective *fingerprints* must be the same and the position in the RSSI vector must represent the same AP in both databases. Therefore, each of the n_{ts} test samples must be composed of a *fingerprint* with n_q RSSI values:

$$\lambda_j^{ts} = \left\{\rho_{1,j}^{ts}, ..., \rho_{n_q,j}^{ts}\right\}, \ j \in [1, ..., n_{ts}] \tag{9}$$

and the position vector must have the same dimensions as its training counterparts:

$$\tau_j^{ts} = \left\{\nu_{1,j}^{ts}, ..., \nu_{n_l,j}^{ts}\right\}, \ j \in [1, ..., n_{ts}] \tag{10}$$

In this case, $\rho_{r,j}^{ts}$ is the RSSI value obtained for AP w_r at position τ_j^{ts}, i.e. $\rho_{r,j}^{ts} = RSSI(\omega_r, \tau_j^{ts})$.

B. The kNN algorithm for indoor positioning

The kNN algorithm is a popular machine learning classifier based on the concept of distance that compares the current sample with all the samples in the training database. In the simplest case ($k = 1$), the label of the current test sample will be the label of the closest training sample according to a distance function.

In the case of indoor positioning [2], the samples are the *fingerprints* and the labels are the positions. In the simplest case ($k = 1$), the position of the current test *fingerprint* will be the position of the closest training *fingerprint* in feature space. When $k > 1$, the final position corresponds to the centroid (weighted or not) of the positions associated with the closest k *fingerprints* in the feature space. In this case, the centroid position of the k nearest neighbors is calculated as follows:

$$\tau_j^{ts} = \frac{\sum_{h=1}^{k} \tau_{nn(h)}^{tr} \cdot w_h}{\sum_{h=1}^{k} w_h} \tag{11}$$

with

$$w_h = \frac{1}{distance(\lambda_j^{ts}, \lambda_{nn(h)}^{tr})^{\delta}} \tag{12}$$

where τ_j^{ts} is the localization of the $j-th$ test sample to be estimated and $nn(h)$ is a function that, given the $h-th$ neighbor, returns the index of the training sample that corresponds to the h-th nearest neighbor. Finally, δ is a parameter to control the importance in the calculation of the centroid of the nearest training samples. With $\delta = 0$, all the samples have the same importance (i.e. $w_h = 1$ for all samples). When increasing δ, the nearest samples will have higher weights and, therefore, more relevance in the calculation of the centroid.

C. Distance functions

We have tested four different distance functions: Euclidean (Eq. 13), City block (Eq. 14), Sørensen (Eq. 15) and Neyman (Eq. 16). They can be used to estimate the distance between two fingerprints λ_i^{tr} and λ_j^{ts} as follows:

$$d_{Euclidean}(\lambda_i^{tr}, \lambda_j^{ts}) = \sqrt{\sum_{q=1}^{n_q} (\lambda_i^{tr}(q) - \lambda_j^{ts}(q))^2} \tag{13}$$

$$d_{Cityblock}(\lambda_i^{tr}, \lambda_j^{ts}) = \sum_{i=q}^{n_q} |\lambda_i^{tr}(q) - \lambda_j^{ts}(q)| \tag{14}$$

$$d_{Sørensen}(\lambda_i^{tr}, \lambda_j^{ts}) = \frac{\sum_{q=1}^{n_q} (\lambda_i^{tr}(q) - \lambda_j^{ts}(q))^2}{\sum_{q=1}^{n_q} (\lambda_i^{tr}(q) + \lambda_j^{ts}(q))} \tag{15}$$

$$d_{Neyman}(\lambda_i^{tr}, \lambda_j^{ts}) = \sum_{q=1}^{n_q} \frac{(\lambda_i^{tr}(q) - \lambda_j^{ts}(q))^2}{\lambda_i^{tr}(q)} \tag{16}$$

D. Data representations

Other important hyperparameter that can affect to the accuracy of the kNN algorithm is how RSSI data is represented when estimating the distance between two fingerprints. Four alternative data representations have been tested: Positive (Eq. 17), Normalized (Eq. 18), Exponential (Eq. 19) and Powed (Eq. 20). Given a RSS value $\rho = \rho^{tr}_{r,i}$ corresponding to the AP ω_r of $i-th$ fingerprint of the training set (this can also be applied to the test set, i.e. ρ can be also $\rho^{ts}_{r,j}$), and with min and max being the minimum and maximum RSS values in the training dataset, the four data representations can be calculated as follows:

$$Positive(\rho) = \begin{cases} \rho - min & \text{if } \rho \text{ is valid} \\ 0 & \text{otherwise} \end{cases} \tag{17}$$

$$Normalized(\rho) = \frac{Positive(\rho)}{max} \tag{18}$$

$$Exponential(\rho) = \frac{\exp(\frac{Positive(\rho)}{\alpha})}{\exp(\frac{-min}{\alpha})} \tag{19}$$

$$Powed(\rho) = \frac{(Positive(\rho))^{\xi}}{(-min)^{\xi}} \tag{20}$$

In this paper, we have used $\alpha = 24$ (eq.19) and $\xi = e$ (eq.20) as in [13].

E. N-TBEA algorithm

N-TBEA is an optimization algorithm that combines searches optimized by an evolutionary algorithm, with the use of game theory data structures such as *Multi-Armed bandits*.

In game theory, the problem of the multi-armed bandit is when a player, faced with a set of slot machines (*bandit* refers to the first slot machines in Las Vegas) has to decide the machine to play next. When he/she plays, the chosen machine returns a reward. The player's objective is to maximize the sum of the rewards obtained by playing multiple times.

In the field of artificial intelligence, bandits are used to decide which is the best action for an agent to play (e.g., in the Monte Carlo Tree Search algorithm [15]). In this case, the actions correspond to each of the machines that make up the multi-armed bandit. For each possible action a, the sum of the rewards obtained (s_a) and the number of times that action has been selected (n_a) will be saved. This choice is commonly made using the UCB formula, which balances exploitation (prioritizing action that has accumulated better rewards in the past) with exploration (prioritizing less explored actions). The formula UCB_β for a given multi arm bandit β is expressed as follows:

$$UCB_\beta(a) = \overline{s_a} + c\sqrt{\frac{\ln N}{n_a + \epsilon}} \tag{21}$$

where a is one of the possible actions that can be chosen in the multi-armed bandit, $\overline{s_a}$ is the average of the rewards that have been obtained in the past by choosing the action a, N is the total number of times an action has had to be chosen, n_a is the number of times the action a has been taken, and c is a constant that balances both concepts of the equation (usually $c = \sqrt{2}$). The ϵ value is a small number (e.g. $\epsilon = 0.5$) that is used to avoid having to divide by zero in the equation. In those cases, i.e. when an action has never been chosen before, the value obtained will be very high to give the opportunity to explore, at least once, all possible actions.

In the case of the N-TBEA algorithm, multiarm bandits will be used to select the best combination of hyperparameters from a set of combinations. To do this, a bandit will be created for each existing hyperparameter in the problem and will have as many actions as possible values of the hyperparameter. Assuming that the problem has m_p hyperparameters, there will be m_p one-dimensional bandits. Furthermore, and this is the key of the method, bandits will be created for all possible combinations of two parameters, i.e. $\frac{(m_p \times (m_p - 1))}{2}$ two-dimensional bandits will be created. In this case, the actions of a two-dimensional bandit will be all possible combinations of the values of those two parameters. Depending on the problem, it is possible to continue creating bandits with more dimensions. We call *model* Π the set of all bandits used.

Algorithm 1 N-TBEA

```
Require: $f$: function to be evaluated
Require: $m_p \in N^+$: Number of hyperparameters
Require: $m_v \in N^+$: Number of neighbors
Require: $\phi \in (0,1)$: Mutation probability
Require: $m_t \in N^+$: Number of initial evaluations
Require: $m_i$: Number of iterations
Ensure: $solution$: best hyperparameters configuration found
 1: $\Pi \leftarrow CreateModel(m_p)$
 2: $current \leftarrow Initialization(f, m_t)$
 3: $\Upsilon \leftarrow [current]$
 4: $iter \leftarrow 0$
 5: while $iter < m_i$ do
 6:     $Population \leftarrow GetNeighbors(current, m_v, \phi)$
 7:     $current \leftarrow argmax_{x \in Population} T_{UCB}(x)$
 8:     $\Upsilon \leftarrow \Upsilon \bigcup current$
 9:     $reward \leftarrow f(current)$
10:     $UpdateModel(\Pi, current, reward)$
11:     $iter \leftarrow iter + 1$
12: end while
13: $solution = GetBestSolution(\Upsilon, \Pi, m_v)$
```

The N-TBEA procedure is described in Algorithm 1. To make it easier to understand, we assume that the number of dimensions of the problem is $m_p = 3$ with 4 possible actions ($[0, 1, 2, 3]$) in each dimension. If we use one-dimensional and two-dimensional bandits, the model will be composed of 3 one-dimensional bandits (β_1, β_2 and β_3) and $\frac{(3\times 2)}{2} = 3$ two-dimensional ones ($\beta_{1,2}$, $\beta_{1,3}$ and $\beta_{2,3}$). Therefore, the model $\Pi = [\beta_1, \beta_2, \beta_3, \beta_{1,2}, \beta_{1,3}, \beta_{2,3}]$.

The process starts creating the model Π (line 1) and with an initialization step (line 2), where m_t individuals are randomly created. An individual, using the terminology of evolutionary algorithms, is a possible combination of hyperparameters.

For example, a possible individual could be the combination $[1, 3, 0]$. That is, for the first hyperparameter the value 1 is chosen, for the second the value 3 is chosen and for the third the value 0. Each individual is evaluated using the function f, obtaining a value that indicates how good or bad that combination is. The use of f is expensive in processing time since, in the case of fingerprinting, it requires executing the full kNN algorithm.

The next step is updating each bandit belonging to the model Π. The update consists of adding the reward obtained by evaluating the individual to the value s_a corresponding to the action to be evaluated. It will also be incremented by one n_a, to increment the number of times that the action a has been chosen. For example, if the individual is $[1, 3, 0]$, the bandit β_1 will update action 1 with the reward obtained by evaluating that individual with the function f, β_2 the action 3 and β_3 the action 0. In addition, the bandit $\beta_{1,2}$ will update the action $[1, 3]$, $\beta_{1,3}$ the action $[1, 0]$ and $\beta_{2,3}$ the action $[3, 0]$. The objective of this initialization is to fill the model with relevant information. At the end of initialization, the individual with the best reward is defined as $current$ individual.

After initialization, the iterative process begins (lines 5 to 12). In each iteration, a population of m_v similar individuals (or neighbors) is generated (line 6) and by means of Eq. 22 a value is calculated (line 7) that indicates the preference of being the next hyperparameter configuration to be evaluated using f. Note that instead of using the slow f to evaluate the m_v neighbors, the fast Multi arm bandit model Π is used to obtain the more promising parameter configuration. The best individual becomes the new current. It is stored in the set of relevant combinations Υ, is evaluated using f (line 8) and all the bandits that make up the N-TBEA model Π are updated using the obtained reward (line 9).

$$T_{UCB}(x) = \sum_{\beta \in \Pi} UCB_{\beta}(x) \tag{22}$$

Neighbors are calculated by modifying one of the hyperparameters by substituting the existing value for another of the possible values of that hyperparameter. Furthermore, for the rest of the hyperparameters there is a probability ϕ of also being modified by mutation. For example, three possible neighbors of the individual $[1, 3, 0]$ could be: $[1, 4, 0]$ (by changing the second), $[1, 3, 2]$ (by changing the third) or $[3, 3, 1]$ (changing the first and also the third by mutation).

The neighbor with the highest T_{UCB} becomes the new individual $current$ and will be the one evaluated by f in the next iteration. As can be seen, in each iteration, a single evaluation of f (kNN in this paper) is performed, and m_v evaluations are performed using T_{UCB}.

The use of this algorithm will make sense whenever the evaluation using f is much more expensive, in processing time, than its equivalent using T_{UCB}. In the case of kNN, $O(n_{tr} \times n_{ts})$ vector distance calculations are required, but the evaluation of T_{UCB} is much faster since it just requires a few one-dimensional algebraic operations.

After the iterative process, a population with m_v neighbors is generated for the individuals that have been *current* one in each iteration (i.e. the ones in Υ) and they are fast evaluated using the N-TBEA model (line 13). The one with the best T_{UCB} will be the proposed solution. This step is necessary because the algorithm cannot ensure that any of the evaluated individuals (those in Υ) is optimal, but it can ensure that one of them or its neighbors will be very close to the optimal.

F. Proposed method

In this work, the N-TBEA algorithm is proposed to find the best combination of the kNN algorithm hyperparameters, where *best* means the one that provides the lowest mean positioning error for the test samples included in $\mathcal{D}_{ts}$, using $\mathcal{D}_{tr}$ as radio map. Therefore, the evaluation function f will be the execution of the algorithm kNN to obtain the position of the test samples and the subsequent calculation of the error of the estimated position versus the real one.

Here, an *individual* is a combination of the kNN hyperparameters and an *action* for a specific bandit, will be the choice of a certain value for a hyperparameter.

III. Experiments and results

A. Data

The data under study consists of a series of databases that reflect the signal strength of the several Wi-Fi or BLE signals detected in a set of points located in indoor spaces. In particular, we have used DSI1–DSI2 [16], MAN1–MAN2 [17, 18], TUT1–TUT2 [19] and UEXB1–UEXB3 [20].

We adopt the experimental setup from [13], which includes nine datasets with diversity in the location, technology, radio map size, density of evaluation and number of detected APs . For instance, MAN1 WiFi dataset has 14300 training samples, 460 test samples and 28 APs, whereas UEXB3 BLE dataset has 240 test samples, 60 evaluation samples and 30 APs.

B. Experimental set up

The following $m_p = 4$ parameters γ_i (with $i \in [1, \ldots, 4]$) of the kNN algorithm have been selected:

- γ_1: Number of neighbors k. 10 possible values, from 0 to 9, are taken which correspond to $k = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]$ respectively;
- γ_2: This hyperparameter corresponds with δ (see equation 12). 3 possible values, from 0 to 2, are taken which correspond to $\delta = [0, 1, 2]$ respectively;
- γ_3: Distance functions (see Section II-C). 4 possible values, from 0 to 3, are taken which correspond to Euclidean [0], City Block [1], Sørensen [2] and Neyman [3];
- γ_4: Data representation (see Section II-D). 4 possible values, from 0 to 3, are taken which correspond to Positive [0], Normalized [1], Exponential [2] and Powed [3].

As an illustrative example, the *individual* $x = [3, 2, 1, 0]$ identifies the use of a k-NN algorithm with $k = 7$, and $\delta = 2$, City Block distance and Positive data representation.

The configuration proposed for this article provides 480 ($10 \times 3 \times 4 \times 4$) possible combinations of the 4 hyperparameters. In this way, we can compare *BF* to *N-TBEA*.

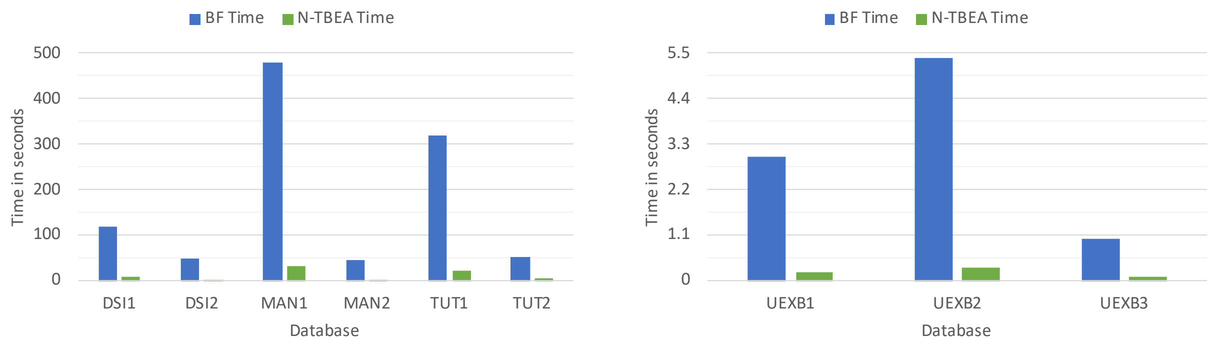

Fig. 1. Processing time in minutes of each algorithm for DSI1–2, MAN1–2, and TUT1–2 databases (left) and UEXB1–3 databases (right)

C. *Experiments*

Two experiments have been carried out. In the first, the kNN algorithm has been tested for all proposed databases using a BF technique, i.e. testing all of the 480 existing combinations for the 4 hyperparameters. The result of each evaluation is the mean positioning error of the samples in the test set. i.e., the Euclidean distance between the position estimated by the kNN algorithm and the actual position. In the second experiment, the N-TBEA algorithm has been used where the evaluation function f, for a hyperparameter configuration, is the execution of the kNN algorithm using that hyperparameter configuration. The value obtained in each evaluation will be the mean positioning error of the samples of the test set, as in Experiment 1.

The N-TBEA algorithm has been configured to initialize the model with $m_t = 20$ random individuals. Subsequently, the algorithm will start the process with the best combination of the previous ones and will perform only $m_I = 10$ iterations. Therefore, the total number of evaluations of a combination using the kNN will be just 30 compared to the 480 execution runs required for BF in Experiment 1. The parameters of the N-TBEA algorithm also include: number of neighbors $m_v = 50$, mutation probability $\phi = 20\%$ and $c = \sqrt{2}$. Since the proposed N-TBEA based algorithm has a random component, the whole process has been repeated 10 times with different random initialization to obtain generalizable results.

All the experiments were run on a computer with Intel Core i7-4790 processor, 16GB of RAM and Ubuntu 20.04.4. Datasets and developed algorithms in Python 3 are available in https://github.com/montoliu/MontoliuPerezTorresICUMT22.

D. *Results*

Table I shows the results of *BF* and *N-TBEA* for all databases (see also Figs. 1–2). The columns labelled with *Best* provides the lowest mean positioning error over all evaluation samples of all combinations in BF and 10 runs of N-TBEA. Similarly, the columns labelled with *Worst* provides the highest mean positioning error. For N-TBEA, the average and standard deviation of the mean positioning error over the 10 runs are also provided. Columns labelled with *Time* report the execution time of BF and N-TBEA.

TABLE I
RESULTS (POSITIONING ERROR AND EXECUTION TIME) OF THE EXPERIMENTS FOR EACH DATABASE.

	BF			N-TBEA			
Dataset	Best [m]	Worst [m]	Time [min]	Mean [m]	Best [m]	Worst [m]	Time [min]
DSI1	3.7	6.4	117.3	3.8 (0.2)	3.7	4.3	7.9
DSI2	3.6	8.1	49.3	3.7 (0.1)	3.6	3.8	3.2
MAN1	2.1	3.1	480.0	2.2 (0.1)	2.1	2.3	31.6
MAN2	1.8	2.6	44.0	1.9 (0.1)	1.8	2.1	2.9
TUT1	4.7	12.9	318.6	4.9 (0.1)	4.7	5.1	20.4
TUT2	7.7	20.0	50.9	8.7 (0.7)	7.8	9.8	3.7
UEXB1	3.0	6.3	3.0	3.1 (0.1)	3.0	3.4	0.2
UEXB2	4.0	5.8	5.4	4.3 (0.2)	4.0	4.6	0.3
UEXB3	6.1	13.0	1.0	6.1 (0.0)	6.1	6.2	0.1

According to the mean positioning errors reported in Table I, the selection of the hyperparameters strongly affect the accuracy of kNN. In TUT2 dataset the difference between the worst combination and the best one in terms of mean positioning erorr is 13.3 m. Although the difference between the worst combination and the best one is just 0.8 m in MAN2, the relative increase is around 50%. The mean of this difference across all databases studied is 4.6 m.

In terms of efficiency, the execution time of the kNN models depends on the radio map size, the number of evaluation samples and the number of APs. With a moderate-size dataset such as MAN1, BF requires 480 min (8 h) to evaluate all combinations. In bigger datasets, covering bigger multi-floor areas and/or with hundreds of thousands of samples (see, for instance, [5]), the execution time of evaluating all the combinations with BF might be prohibitive.

Fig. 1 shows the time needed (in minutes) to run BF and N-TBEA in all datasets. N-TBEA is approximately 15 times faster than the BF algorithm. Note that N-TBEA only evaluates 30 times using the full kNN algorithm whereas the BF algorithm performs 480 evaluations. For instance, in TUT1 database, the BF algorithm needed more than 300 min to obtain the best hyperparameters combination. However, the proposed N-TBEA based needed just 20 min to obtain a solution that, on average, is only 0.2 m worse. Even in the worst case of N-TBEA, the value obtained is only 0.4 m worse than the best possible combination. In some repetitions, the proposed N-TBEA provides the same error as BF.

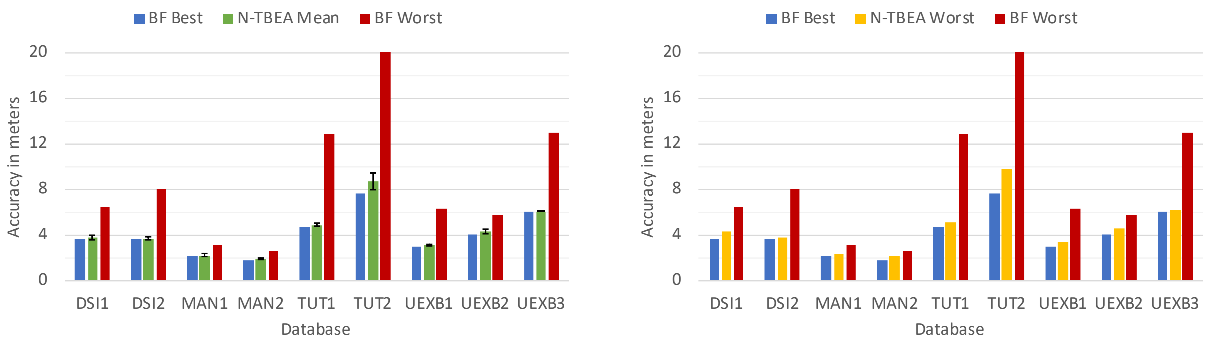

Fig. 2. Results obtained for the combination of parameters tested that give the minimum and the maximum positioning error. Left: comparison between mean N-TBEA results and Best and Worst BF results. Right: comparison between worst N-TBEA results and Best and Worst BF results.

Fig. 2 (left) shows, for each dataset, the best (blue) and worst (red) mean positioning error obtained using the BF algorithm, and the mean value (green) obtained from the 10 repetitions of the proposed N-TBEA based algorithm. Fig. 2 (right) shows, instead of the mean, the worst value obtained using N-TBEA among the 10 repetitions (yellow). The difference between N-TBEA Mean and BF Best is on average 0.2 m, with only one dataset, TUT2, with more than 0.5 m of difference. In the worst-case scenario, the difference between the worst N-TBEA combination and the best combination is on average 0.6 m, being below 0.5 m in 6 datasets. The worst result is obtained for TUT2 where the difference is 2.1 m. Furthermore, the performance of the overall best and worst cases depend on the dataset. Although the overall worst error is significantly high in some datasets, above 12 m in TUT1, TUT2, UEXB3, the worst combination provided by N-TBEA performs similarly to the best overall combination, almost matching the best performance for TUT1 and UEXB3.

The results show the potential of N-TBEA to efficiently retrieve a good combination of parameters, which is of utmost relevance in huge radio maps.

IV. Conclusions

This work has presented a method to efficiently obtain a good configuration of the hyperparameters of the kNN algorithm to solve an indoor positioning problem. The proposed method is based on the use of the N-TBEA technique, which has the main advantage of obtaining solutions similar to the optimal one in a very short processing time.

The performance of our proposal has been verified both, in terms of accuracy of the best combination obtained and in processing time, compared to the case of using a Brute Force algorithm that calculates all possible combinations. The proposed hyperparameter search algorithm presents a good trade-off between performance (positioning error) and efficiency (execution time required to retrieve the hyperparameter values), being 15 times faster than Brute Force and really close to the overall best configuration in several datasets.

Future work will focus on testing the proposed method on more complex databases and considering more hyperparameters. We consider N-TBEA has the potential to be used with any Machine Learning model for indoor positioning.

References

[1] A. Ometov *et al.*, "A survey on wearable technology: History, state-of-the-art and current challenges," *Computer Networks*, vol. 193, 2021.

[2] P. Bahl and V. Padmanabhan, "RADAR: An in-building rf-based user location and tracking system," in *Proceedings IEEE INFOCOM*, 2000.

[3] J. Hu *et al.*, "Experimental analysis on weight k-nearest neighbor indoor fingerprint positioning," *IEEE Internet of Things J*, vol. 6, 2019.

[4] M. T. Hoang *et al.*, "A soft range limited k-nearest neighbors algorithm for indoor localization enhancement," *IEEE Sensors Journal*, vol. 18, no. 24, pp. 10 208–10 216, 2018.

[5] M. Aernouts *et al.*, "Sigfox and lorawan datasets for fingerprint localization in large urban and rural areas," *Data*, vol. 3, no. 2, 2018.

[6] Z. Zhao *et al.*, "I-wknn: Fast-speed and high-accuracy wifi positioning for intelligent sports stadiums," *Computers & Electrical Engineering*, vol. 98, 2022.

[7] S. Liu, R. De Lacerda, and J. Fiorina, "Performance analysis of adaptive k for weighted k-nearest neighbor based indoor positioning," in *2022 IEEE 95th Vehicular Technology Conference*, 2022.

[8] "ISO/IEC 18305:2016 Information technology - Real time locating systems - Test and evaluation of localization and tracking systems,"

[9] N. Saccomanno, A. Brunello, and A. Montanari, "What you sense is not where you are: On the relationships between fingerprints and spatial knowledge in indoor positioning," *IEEE Sensors Journal*, vol. 22, no. 6, pp. 4951–4961, 2022.

[10] J. Torres-Sospedra *et al.*, "A comprehensive and reproducible comparison of clustering and optimization rules in wi-fi fingerprinting," English, *IEEE Transactions on Mobile Computing*, vol. 21, no. 3, pp. 769–782, Mar. 2021.

[11] A. Brunello, A. Montanari, and N. Saccomanno, "A genetic programming approach to wifi fingerprint meta-distance learning," *Pervasive and Mobile Computing*, vol. 85, p. 101 681, 2022.

[12] J. Torres-Sospedra *et al.*, "Comprehensive analysis of distance and similarity measures for wi-fi fingerprinting indoor positioning systems," *Expert Systems with Applications*, vol. 42, no. 23, 2015.

[13] C. Rodriguez-Martinez and J. Torres-Sospedra, "Revisiting the analysis of hyperparameters in k-nn for wi-fi and ble fingerprinting: Current status and general results," in *2021 Inter. Conf. on Indoor Positioning and Indoor Navigation (IPIN'21)*, 2021.

[14] S. M. Lucas, J. Liu, and D. Perez-Liebana, "The N-Tuple Bandit Evolutionary Algorithm for Game Agent Optimisation," in *2018 IEEE Congress on Evolutionary Computation (CEC'18)*, IEEE, 2018.

[15] C. B. Browne *et al.*, "A survey of mcts methods," *IEEE Trans. on Comp. Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, 2012.

[16] A. Moreira, I. Silva, and J. Torres-Sospedra, *The dsi dataset for wi-fi fingerprinting using mobile devices*, version 1.0, Zenodo, Apr. 2020.

[17] T. King *et al.*, "Distribution of fingerprints for 802.11-based positioning systems," in *Int. Conf. on Mobile Data Management*, 2007.

[18] T. King *et al.*, *Crawdad dataset mannheim/compass (v. 2008-04-11)*, [available] https://crawdad.org/mannheim/compass/20080411, 2008.

[19] S. Shrestha, J. Talvitie, and E. S. Lohan, "Deconvolution-based indoor localization with wlan signals and unknown access point locations," in *Proc. of 2013 Int. Conf. on Localization and GNSS*, 2013.

[20] F. J. Aranda *et al.*, "Multi-slot ble raw database for accurate positioning in mixed indoor/outdoor environments," *Data*, vol. 5, no. 3, 2020.