Machine Vision and Applications manuscript No.
(will be inserted by the editor)

Hyper-parameter optimization tools comparison for Multiple Object Tracking applications

Francisco Madrigal · Camille Maurice · Frederic Lerasle

Received: date / Accepted: date

Abstract Commonly, an algorithm needs a certain number of variables that control its behavior. The optimal values result in a better performance that could generate profits for companies, make the algorithm stands out from similar applications or improve its ranking in algorithm competitions. However, finding this values is not straightforward because manual tuning could be a stressful and difficult task even for expert users. This paper presents, evaluates and compares 4 tools in the literature for hyper-parameter optimization, selected due to their number of citations, code availability and impact on literature: MCMC, SMAC, TPE and Spearmint. We evaluate these tools using the publicly available source code provided by the authors in a computer vision application: Multiple Object Tracking (MOT), which is a challenging topic and the strategies rely on a set of tunable parameters. This analysis considers the impact of hyper-parameter optimization tools in terms of stability, performance, usability, among others. The evaluations are carried out using public benchmarks such as PETS09 and ETH and the results show how these tools change the performance of a MOT framework and how this would affect the results of real ranked competitions. Our goal is (1) to encourage the reader to use these tools and (2) to provide a some guidelines that helps anyone when tuning his/her methods.

Keywords Stochastic optimization · Parameter tuning · Visual tracking

This work was carried at LAAS-CNRS and supported by the DGA-RAPID project SERVAT: Suivi Et Reconnaissance Visuels, Adaptatif, Temps-réel.

F. Madrigal, C. Maurice, F. Lerasle
LAAS-CNRS
7 av. du Colonel Roche
31400 Toulouse France
E-mail: pacomd@laas.fr

1 Introduction

Multiple Object Tracking (MOT) has several applications such as surveillance, human-machine interaction, among others. It is a hot topic for research because there are still many challenges to be solved, such as occlusion, identity changes or lack of precision [51,33]. In the last decades, MOT community has addressed some of the aforementioned issues with different strategies [11]. Some approaches follow a tracking-by-detection scheme [10,43], others compute the trajectories with the use of graphs [1,42].

Regardless of the strategy used, one thing in common is that most of them require adjusting a set of parameters that control the method behavior. Commonly this tuning is done manually or, sometimes, the parameters are set based on heuristics or assumptions. Hand tuning is feasible when the number of parameters is small. Otherwise, this could be difficult due to time consumption and requirement of expert skills. In either case, it is possible that the configuration has a bias due to the user. Therefore, it is difficult to know if low performance depends on the proposal itself or on the inadequate selection of parameters. This tuning problem is considered in some methods that include techniques that allow to set their parameters on-line [18,57]. In some cases, a furtive search can be made in the (discretized) space of the parameters, but this requires a high computational cost. An efficient way to find optimal values automatically is to use more sophisticated algorithms such as numeric methods [37,2] or machine learning [18], but which technique to use depends on the type of application.

If we classify the proposals in two groups deterministic or stochastic [14], we can observe that the former is easier to estimate than the latter because it provides always the same results with the same parameters. Stochastic based approaches are more difficult to tune because they produce a different result in each execution even with the same parameters [25]. An example is the particle filter-based methods that give a different result in each evaluation. Therefore, they require evaluating several times the same parameters only to decide if those are good enough.

In the literature there are methods that can find the best parameters in an efficient way. Hyper Parameter Tuning (HPT) could be seen as an optimization process, where the goal is to find the set of parameters that maximize (or minimize) a cost function. The optimization schemes [41,7,38] can be roughly classified as local and global optimization [37]. The latter focuses on finding the minimum (or maximum) over the whole parameter space instead of only in the local space. Global optimization [23] can be divided again into three subgroups: Deterministic, Stochastic and Metaheuristic. The first requires the derivate of the cost function [24] meanwhile the third group is built up based on heuristics [50,7]. Although, for stochastic tracking methods, it is impossible to infer any information of the cost function and it then must be treated as a black box. Therefore, the use of deterministic optimization methods is not suitable and the selection of a heuristic-based optimization could not work for all the cases.

In order to use the HPT methods, we need to transform the MOT approach into a cost function, which can be done using an evaluation metric. MOT community has created a set metrics called CLEAR-MOT [5] that allow the evaluation and comparison of the different proposals in a fair and quantitative manner. Furthermore, MOTChallenge [30] community has coded those metrics in a public evaluation toolkit. Also, they provide a framework that includes a collection of datasets, several challenges and a ranking of algorithms, all of those aimed to address MOT challenge. There the state-of-the-art methods are ranked with respect to their performance. In general, the results are so close that even a small improvement of 2 percent can lead to win or lose 10 positions in the ranking. However, the precision of the results may depend on the selection of the meta-parameters in the proposals [34]. Therefore, the ranking could change with a more optimal tuning of the parameters.

From all of these, the problems that make tuning a difficult task, even with optimization, are: (1) performance impact, (2) computational time cost, (3) performance stability, (4) handling of external tools and (5) robustness of those tools. There are multiple studies on the influence of the (online/ offline) free parameters [4,9,47] (HPT tools) that can handle the above problems. However, it is not always clear which one to use.

The objective of this work is to provide the reader with an analysis, as detailed as possible, of several hyper parameter tuning tools. Highlighting their strong features, weak points and other aspects of interest that can guide the reader to select the most pertinent tool for their tracker implementation and beyond. Among all of them, we select the most relevant optimization methods, in terms of number of citations and code availability, and we evaluate them in the computer vision context of multi-person tracking.

Our analysis is aimed at classic tracking using the Particle Filter (PF) that commonly involve several parameters [1]. This study shows the existence of correlation between the parameters and how this affects the search of the optimal parameters. We evaluate the performance and provide a statistical analysis of each tool in terms of performance (CLEAR-MOT), stability, robustness and convergence speed among others. Even if this work is aimed to tracking applications, more precisely in multiple pedestrian tracking, the results can be applied to another vision-based modality. To the best of our knowledge, no similar studies have been proposed in the visual tracking community even though optimization tools are essential for fine-tuning [40].

This paper provides a substantial improvement to a previously published conference paper [35]. Here, we evaluate a monocular MOT framework more generic than the multi-camera scheme of [35]. The analysis is broader, with more tools, considering more aspect than the original such as:

- Tool computational time,
- Results repeatability,
- Approach convergence,
- Size of training set,
- Sensitivity to the initial point,
- Evaluation using a combination of metrics and
- Comparison with state of the art.

Therefore, the analysis focuses on 8 important points that make the tools efficient and practical to use: (1) Convergence speed, (2) Stability, (3) Accuracy, (4) Computational time, (5) Robustness of starting point, (6) training size and (7) number of particles, and (8) Documentation.

The paper is organized as follows: Section 2 gives the details of the selected optimization methods and the description of the corresponding tools. The details of our MOT system and the set of parameters to be optimized are described in section 3. The discussion of the results is presented in section 4. Finally, section 5 describes the conclusion of our analysis.

2 Hyper-optimization methods

At first the use of an optimization algorithm for tuning parameters could be seen as unnecessary, difficult to understand or incorporate to one's application. However, several proposals [53,52] have shown how parameters with different values change the results of the same method. Our work focuses on analyzing various tools and showing how we can adapt them into the MOT context.

There are several strategies for hyper-parameter optimization in the literature [4,9,28]. Classic hyper-parameter optimization methods use brute force methods such as random search or grid search. Some proposals model the problem as a differentiable function that can be optimized using the gradient information [24]. New proposals such as [18] maintain a continuous update of the parameters using reinforcement learning (Q-learning). Methods based on Bayesian optimization have been explored for a long time and have a good adaptability and efficiency.

Bayesian optimization methodology estimates the global maximum (or minimum) of noisy black-box functions and be used in various areas such as machine learning [32,45], robotics [31,12], deep learning [17], combinatorial optimization [48,27], among others. It collects information from the observation and builds a model as close as possible to the unknown function [40]. Most of the methods begin with a general model of the function, given a priori, which is then updated according to the observations obtained from the evaluation of the hyper parameters. This process is iterative and the method carefully chooses the next parameters to evaluate. An intelligent algorithm should have a good trade-off between the exploration of the hyper-parameter space and exploitation of the cost function, i.e. high values when evaluating the cost function. If this is the case, the algorithm will converge faster, requiring fewer function evaluations and therefore less computational time.

In the literature, there exists several proposals with this goal but only a few provide open source code. We privilege those approaches with available code. Nevertheless, in the literature there are methods, easy to implement, that could help to find the optimal parameters. To know if a classical method is enough to find reliable parameters, we start our study with a brute force methodology, *i.e.* grid search, and simple stochastic optimization algorithm as a baseline.

2.1 Exhaustive grid search

Grid search optimization exhaustively evaluate candidates from a grid of parameter values. The grid is built by dividing the n parameters b times and thus the search space has b^n configuration. It is a simple method that does not require additional knowledge and that can be applied immediately to any algorithm. However it suffers from the curse of dimensionality, if either both number is large then the configuration space can explode making difficult to optimize in realistic time, but it is easily parallelizable because the hyper-parameter are independent of each other.

2.2 Bayesian optimization

It may seem that using a sophisticated optimization algorithm is unnecessary, difficult to understand or to use in a particular method. Perhaps a basic method is sufficient to find reliable parameters. To know if this is the case, we begin our study by analyzing a classical stochastic optimization algorithm as a baseline.

2.2.1 Markov Chain Monte Carlo optimization

Generally, classical methods are model-free, with a large theoretical background [36]. Those are relatively simple and easy to implement. Since we consider our function as a black box, we need a strategy that does not rely neither on the gradient nor heuristics. Thus, we employ stochastic optimization based on Markov Chain Monte Carlo as a baseline to compare (in terms of performance and easiness of use) against more robust techniques. The Markov Chain Monte Carlo (MCMC) is a set of stochastic methods based on a sampling procedure. It is widely used in the literature, most commonly to determine numerical approximations, which, in our case, is the optimal value of the loss function. They do not require prior information of the function, *i.e.* gradient. We can find several MCMC proposals in the literature but we focus on the Metropolis-Hastings (MH) algorithm [8]. MH approximates a distribution from which direct sampling is difficult or impossible (*i.e.* the loss function) by drawing samples from a prior known probabilistic distribution. Also, MH does not suffer from the curse of dimensionality that allows estimating high dimensionality distributions efficiently.

To define it mathematically, let $f(\lambda)$ be the function that evaluates the hyper parameters λ. Initially, we choose an arbitrary initial set λ_0 and a probability density $g(\cdot)$ from which we can easily draw new samples given the previous estimation. In our case, we define $g(\lambda_t)$ as a Gaussian distribution centered at the previous λ_t with a fixed variance. In each iteration, we draw a candidate sample $\lambda^* \sim g(\lambda_t)$ which is used to calculate the acceptance ratio $\alpha = f(\lambda^*)/f(\lambda_t)$. We accept the new candidate $\lambda_{t+1} = \lambda^*$ if it improves the function value (i.e. $\alpha > 1$). If not, we accept it with certain probability given by uniform distribution. The details of the algorithm are available in [8].

2.2.2 Spearmint

The previous section describes how MCMC approximates the distribution of the cost functions using sequential samples. Moreover, if we have enough samples we can assume, following the central limit theorem, that those induce a multivariate Gaussian distribution. This assumption about the distribution of the function is known as the Gaussian process [40]. Some optimization proposals are based on this idea and one in particular is the work of Snoek et al. [45] called Spearmint, which analyzes practical considerations to improve this Bayesian optimization algorithm. Spearmint is well known in the community, having over 1000 citations and dedicated websites to explain it[1][2].

The objective of Spearmint [45] is to capture the dependence between the hyper parameters λ and the cost function f. This is achieved with the use of a probability distribution $p(f|\lambda)$, modeled by a Gaussian process. However, instead of finding the maximum likelihood, the proposal marginalizes the model using slice sampling. Each slide is independent from the others, therefore they can be evaluated in parallel, each one with a different marginal. This capability of parallelization makes possible to reduce the computational time of some expensive methods such as Deep Learning [46,54]. Several approaches based on a Gaussian process exploit this mechanism of parallelization [26]. Parallelization is a very important point, multiple applications, that although a bit out of our tuning context, have shown the benefits of parallelism in contexts such as video coding [55,56], allowing to process more information in a fraction of time.

Snoek proposes others practical considerations such as the use of a Matérn $5/2$ kernel instead of squared exponential kernel. The latter is typically used on Gaussian process regressions but this is unrealistically smooth for several optimization problems. Also, the approach introduces the Expected Improvement (EI) per cost acquisition function that samples not only good parameters but also parameters fast to evaluate.

2.3 Structure-based Bayesian optimization methods

Other methods try to describe the manifold of the hyperparameter space through the use of models, generally following an iterative optimization strategy. In each iteration the methods evaluate a hypothesis of the parameters and update the model. After a given number of iterations, they propose a final candidate according to their own methodology, which provides the best performance. This group of methods is called Sequential Model-Based Optimization (SMBO) [27] and is recurrently used when the black box cost function is computationally expensive. We evaluate two state-of-the-art methods, selected based on their popularity (high number of citations), availability of source code and their good performance.

2.3.1 Sequential Model-based Algorithm Configuration

One instantiation of SMBO is the Sequential Model-based Algorithm Configuration [27] (SMAC), which is an optimization method based on machine learning and has over 600 citations. This tool is under support including the new version 3, which is still in development. This iterative approach considers all the collected information when proposing a new candidate, leading to better parameter estimations. Here, SMAC models the posteriori $p(f \mid \lambda)$ as a Gaussian distribution using random forests (RF) and it estimates an empirical mean and variance for each tree. This RF-based model improves the performance in discrete optimization, adapts easily to the data and can handle noisy functions with parameters of high dimensionality. SMAC works in the following way: First, the set of proposal parameters are sampled from a uniform distribution in a given finite range. Second, the samples are divided according to one randomly selected parameter of λ, which later are used to build the regression trees. This division process is repeated until it reaches a minimum number of samples per branch. The configurations are evaluated using an "expected improvement (EI)" criterion and the most promising are selected. Finally, the best configuration, the one with the highest EI score, is compared against the previous one. The new proposal is accepted if it improves the cost function value. If that is the case, the corresponding tree is used in the next iteration.

2.3.2 Tree-structured Parzen Estimator

The Tree-structured Parzen Estimator [3] (TPE) is a well-known model-based method and has over 500 citations. While SMAC and Spearmint model $p(f \mid \lambda)$ explicitly, *i.e.* the estimation of the cost function value given the parameters, TPE factors it. Thus, it models the probability of the parameters given that they improve ($p(\lambda \mid f \geq f^*)$) or worsen ($p(\lambda \mid f < f^*)$) the evaluation of the loss function according to a fixed quantile of the losses observed so far. These two probabilities are modeled with tree-structured Parzen estimators.

In practice, TPE draws, in each iteration, new samples of the parameters and selects the set to test during the next iteration. TPE draws samples of λ uniformly in the configuration space, therefore it does not require initial guesses or training sets. Then, the samples are evaluated according to cost function f. The approach splits the samples in two groups based on their score. The first group has all samples that improve the current score estimation f^* meanwhile the second contains the remaining. Then, both groups are used to model the likelihood probability: a model $g(\lambda) = p(\lambda \mid f \geq f^*)$ for the first group with greater values and $l(\lambda) = p(\lambda \mid f < f^*)$ for the second group with lower values. The models use a 1-D Parzen estimator to measure the density of the groups through a hierarchical structure. The goal is to create new candidates that are most likely to be in the first group. Thus, new samples of $g(\lambda)$ are drawn at each iteration and the one with the highest improvement is

[1] http://fastml.com/tuning-hyperparams-automatically-with-spearmint/

[2] http://www.nersc.gov/users/data-analytics/data-analytics-2/deep-learning/hyperparameter-o/spearmint-bayesian/

then used in the next iteration. TPE defines the Expected Improvement as the ratio between models, $EI(x) = \frac{l(x)}{g(x)}$.

2.4 Associated tools

MCMC is a robust method to estimate the parameter distribution and its implementation is simple and does not require a strong background. Therefore, we create an implementation in C++ using the classic Metropolis-Hastings algorithm [8]. It receives the same input as the tools such as the limits of the configuration space, number of iterations, initial point and the black box cost function. The configuration space is fixed in the same way as the other tools, where new samples follow a Markov chain and are drawn from a Gaussian distribution with mean in previous state and fixed variance. Here, the number of iterations plays an important role in the final results. A small number could lead to an underestimation of the parameter distribution but a large number could generate an over-fitting problem.

Spearmint tool [45] was created by members of the Harvard Intelligent Probabilistic Systems Group. The open source code [3] requires installing both Python and MongoDB [13], being the latter is a database software that keeps the collected information of Spearmint. MongoDB allows different applications (or threads of the same application) to access the same set of data simultaneously. This allows several executions of the tool to communicate, making parallel optimization possible. Also, it can be used to track the process from the last evaluated iteration, allowing the user to continue the optimization or to analyze the results in case there is a problem with the evaluation of the function. Spearmint documentation is very limited, with only a short explanation of how to set up a simple experiment. The included examples give a better idea of the different tool parameters available. It requires defining two files: a parameter file, which set the configuration space limits, and a Python file, which launch the black box function. Additional parameters such as number of iterations are passed as terminal arguments.

SMAC [27] is a publicly available tool for parameter optimization that has been developed for several years. The stable release SMAC v2 has been published in 2015 and v3 is in development. It is based on Java [4], making it easy to install and to run on Windows and Unix environment. The authors provide documentation such as a quick installation guide, an environment setup and a detailed manual describing the tool, which includes a good description of the outputs. The framework is easy to use, requiring the definition of a scenario file (in Python) and a parameter file. The first has the information to launch the black box function, the path to the parameter file and optimization options, *e.g.* the number of iterations. The parameter file describes the configuration space limits, initial point and data type.

TPE [3] is an optimization tool and it forms part of the Hyperopt library[5] and Optunity library [6]. The latter is supported for different languages such as Python, MATLAB, Octave, R, Julia, and Java. It is based on Python, which should have been installed beforehand. Hyperopt implements TPE algorithm for Python and can be used either serially or in parallel via MongoDB. Both libraries provide a simple documentation and small examples of their use. The configuration is simpler than SMAC, limiting it to the call of a minimization function that receives as parameters the objective function, number of iterations and the limits of the configuration space.

3 Study case

In this paper, we evaluate the performance of different hyper-parameter optimization tools. This analysis can be done on any methodology that can be formulated as a cost function. Nevertheless, the relevance of optimization is more evident with complex methods with several parameters.

Some Multiple Object Tracking (MOT) methods fit well in this description [29]. They are widely studied in the literature because there are several challenges in order to detect and track pedestrians/people correctly. Those methods are usually grouped as deterministic or stochastic [49,44]. With the later, the optimal parameters are more challenging to find because the same set of parameters provides different results in each evaluation.

MOTChallenge provides the results of several tracking methods where hundreds of those are based on tracking-by-detection. Thus, we follow a decentralized particle filter strategy in the vein of [10]. Therefore, we implement a tracking-by-detection framework where each target state is estimated by a Particle Filter (PF) in image plane. We define the target state as $\mathbf{X} = \{x, y, u, v\}$ where (x, y) are the 2D image position and (u, v) are the corresponding velocity components. We use the pre-computed detections named ACFINRIA, which are provided by MOTChallenge [30]. Thus, we always have the same detections for all the experiment, limiting the variability of the algorithm to only the tracker. The detector is not a vital part of this work because it is not considered in the tuning process.

Each detection is associated to one tracker only if the distance between them is smaller than a threshold dH. We perform the association with the Hungarian algorithm [10, 39,6] and we create a new tracker for any detection that has been not associated to an existing tracker. Fixing a value to dH, in the image plane, is not that easy because its value

[3] https://github.com/HIPS/Spearmint
[4] http://www.cs.ubc.ca/labs/beta/Projects/SMAC/v2.10.03/quickstart.html
[5] http://hyperopt.github.io/hyperopt/
[6] http://optunity.readthedocs.io/

depends on the image size, frame-rate and how far camera is from the targets. Moreover, small values can omit correct associations meanwhile high values could lead to a wrong association. So, we have to tune dH in such a way that it generalizes the distance for all cases.

We use color information to model the pedestrian appearance with a histogram per target. Then, each tracker contains a reference histogram H_r, which defines individual appearance, and we build a histogram for any incoming detection H_d. The observation model of the particle filter measures both histograms using the Bhattacharyya distance. The reference histogram is updated as the weighted sum of both: $H_r = hur * H_r + (1 - H_d)$, where $0 < hur < 1$ is the histogram updating rate. If the target appearance does not change much over the scene, *i.e.* constant lighting, we can put hur close to 1 or reducing it otherwise. If the value is too small, we risk adding spurious information in the model, which reduce tracking performance. Finding the optimized value is essential to obtain good results.

The motion model of our PF consists of a random walk model, here the state is updated as follows: $X_t = X_{(t-1)} + \eta_t$ where $\eta_t = \{\eta_t^1, \eta_t^2, \eta_t^3, \eta_t^4\}$ is an independent Gaussian noise for each parameter ($\eta_t^* \sim N(0, rwn_*)$). The random walk noise rwn_* allows the tracker to move at different directions with different speed. However, the correct setting depends on the behavior of pedestrians. Small values are good to track slow pedestrians while higher values allow fast targets to be followed. Normally, the sequences have both cases (slow and fast pedestrians) and finding the optimal noise parameters is a difficult task.

Our implementation of the particle filter follows the Sequential Importance Resampling scheme, applying a resampling step in each iteration. The number of particles np plays an important role. In many cases, a large number of particles could improve the state estimation but this also increment the computational cost. Sometimes a small number is adequate for simple scenarios. Therefore, we need to find the number of particles with the best accuracy/computational cost ratio. Finally, this implementation of a basic tracking-by-detection framework has 7 parameters to tune: association distance dH, histogram updating ratio hur, four random walk noise variables rwn_* and the number of particles np. Those components are put together in a vector:

$$\lambda = \{dH, hur, rwn_1, rwn_2, rwn_3, rwn_4, np\}, \tag{1}$$

defining our parameter set used in the optimization process. These are the parameters that a MOT system commonly uses but more parameters can be included (*i.e.* detector threshold) or a different tracking methods with other parameters.

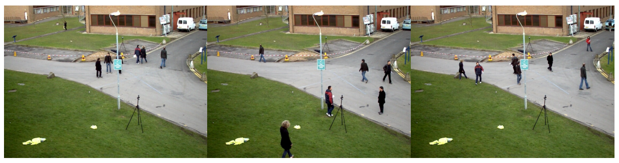


Fig. 1: Examples of sequence S2L1 from PETS 2009 dataset [22]. View from camera 1.

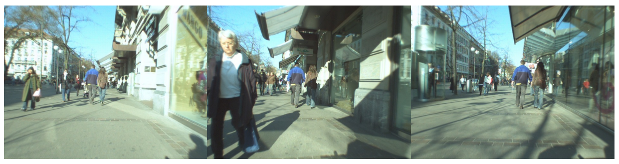


Fig. 2: Examples of sequence Sunny Day from ETH [20].

4 Evaluation of the tools

Initially, we describe the data and metrics used to perform the evaluations as well as the methodology. Finally, we present the results and we discuss them in both quantitative and qualitative aspects.

4.1 Dataset description

For all the evaluations of the presented tools, we use two sequences extracted from well-known public datasets: PETS 2009 [22] and ETH [20]. Both sets are challenging benchmarks to evaluate the performance of any tracking framework. The PETS2009 dataset has several sequences designed for a specific goal that goes from single pedestrian tracking to flow estimation. The density of crowdedness varies from sparse to highly dense. We use the set S2L1 (see Fig. 1) that consists of 8 synchronized sequences observing a common outdoor scene. Each video has 795 frames recorded at 7 fps with a resolution of 640x480 pixels. It is a structured scene with three portions of road surrounded by grass. The sequence has a medium density level, with 19 pedestrians, and is oriented to single target tracking with challenging situations, such as clutter occlusions. Among all the sequences, we work with the view 2. We focus on this sequence because it has been overexploited in the literature [19,21]. The results have almost reached the ceiling showing small improvement between proposals. This work shows how the ranking of the same tracking framework can change with the use of optimally tuned parameters.

The ETH dataset consists of 8 sequences captured by a stereo pair of cameras mounted to a stroller. The camera has a resolution of 640x480 pixels with a frame-rate of 13-14 fps. Each video shows a different place of a busy street. These are challenging sequences because camera has a low point of view increasing the number of occlusions. We evaluate the tools using the *Sunny Day* sequence, see Fig. 2.

Both sequences represent classic scenarios in pedestrian tracking, the first being a static camera located at the distance with the purpose of surveillance, and the other a mobile camera with the purpose of interacting with the environment.

For our framework, we use detections provided by MOTChallenge, which are generated by using the approach of Dollar [16] named Aggregate Channel Features. This pedestrian detector is trained using the INRIA pedestrian training dataset introduced in [15]. We perform the evaluation using the kit and ground-truth both provided by MOTChallenge [30].

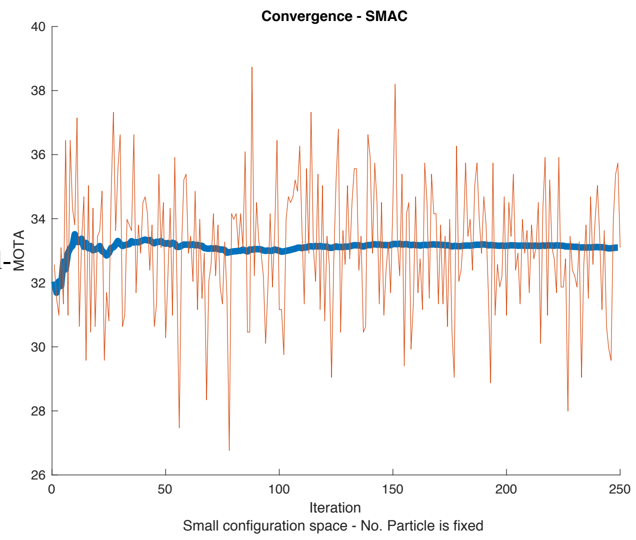


Fig. 3: Evaluation of SMAC using the configuration SF. The function evaluations, using the metric MOTA, are shown in orange. Cumulative distribution is shown in blue.

4.2 Experimental setup

We analyze the performance of several parameters optimization tools, where the loss function is a black box, as mentioned in section 3, that takes as input the set of parameters λ and returns a measure of the tracking performance. It consists of two parts: (1) a tracking-by-detection system in the same vein as Breitenstein approach [10] and (2) a performance evaluation system, which is carried out using the Multiple Object Tracking Challenge Development Kit provided by MOTChallenge [5]. This section describes the configurations used for each tool and the evaluation metrics used.

4.2.1 Tool configurations

In general, tools require defining three aspects: searching space, initial value and type of variable. This information is stored in separate files, with a specific format and name. Some tools as TPE and Spearmint does not require setting an initial guess of the parameters, for SMAC and MCMC we choose the middle point in the search space.

The searching space must be delimited according to the parameters to evaluate. As mentioned in Eq. 1, we have λ with seven variables and from those, only the histogram-update-ratio variable hur is well delimited because it is normalized. For the rest, we can only use our experience to set the limits. Therefore, we define a small configuration space S fixed around from where we expect to find the optimal configuration. In our case, S is set as follows:

$$S = \{dH \in [0.1, 10], hur \in [0.1, 1], \\ rwn_* \in [0.1, 3], np \in [10, 100]\}. \tag{2}$$

However, we do not know if those parameters are the best. Then, we create a larger configuration set L as follows:

$$L = \{dH \in [0.1, 20], hur \in [0.1, 1], \\ rwn_* \in [0.1, 20], np \in [10, 100]\}. \tag{3}$$

Number of parameters could influence the optimization process. Also, it could be a correlation between the number of particles np and the rest of parameters which influences the final results, *i.e.* few particles make the system faster. A higher number could overcome the limits of the other parameters but this represents an additional computational cost. We analyze these aspects by creating two experiment sets labeled as F and NF that fix a priori the number of particle or not, respectively. In this manner, we have a set with 6 parameters and other with 7. The fixed number of particles is not given randomly but is selected with a prior knowledge that this value is close to the optimum. Otherwise, the results will have a lower performance.

4.2.2 Evaluations and comparison protocol

In the literature there exists several metrics to measure the performance of tracking approaches. The best-known are the CLEAR-MOT and those are implemented in the MOTChallenge development kit. From all the metrics, we use the Multiple Object Tracking Accuracy (MOTA) metric as the loss function f because it is, commonly, the key metric when comparing several tracking methods. It compares the algorithm results with respect to the ground-truth. MOTA combines the information of missed detection, mismatches between detected objects, and false positives. The MOTChallenge [30] ranked by default the results using this metric and when comparing the best positions of the 2D MOT 2015

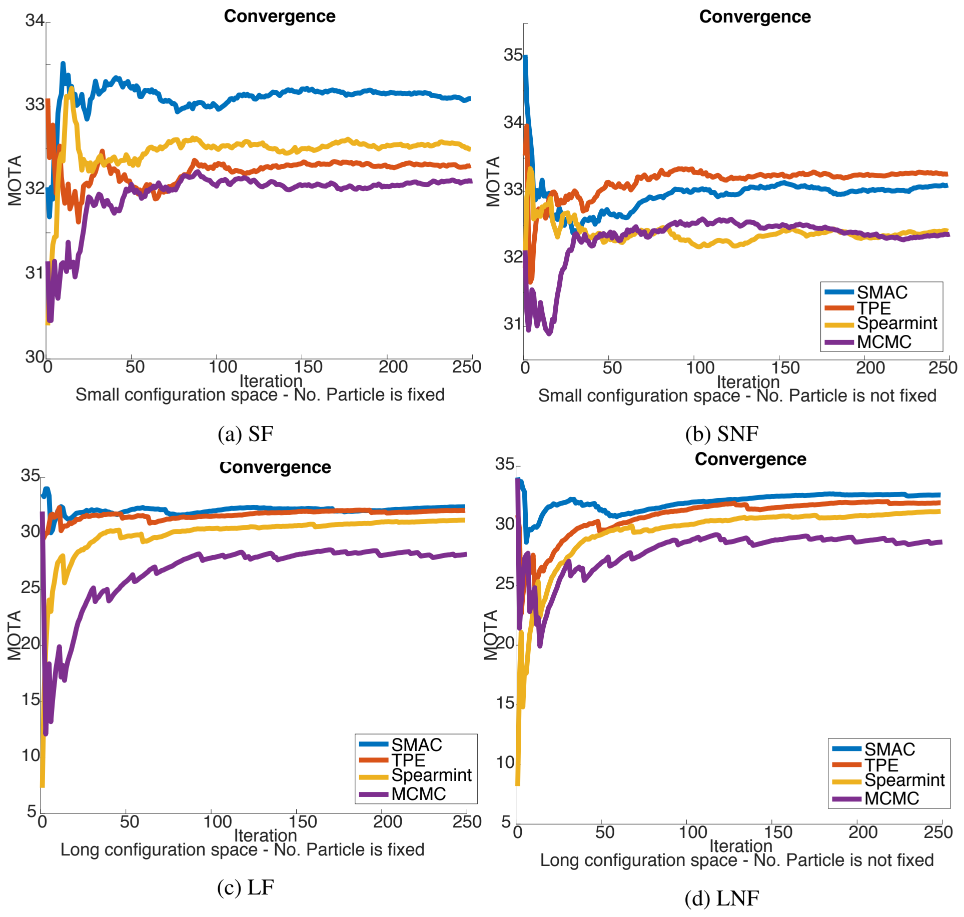


Fig. 4: **Convergence speed evaluation.** Cumulative MOTA of SMAC [27], TPE [3], MCMC and Spearmint [45] using PETS09-S2L1 sequence.

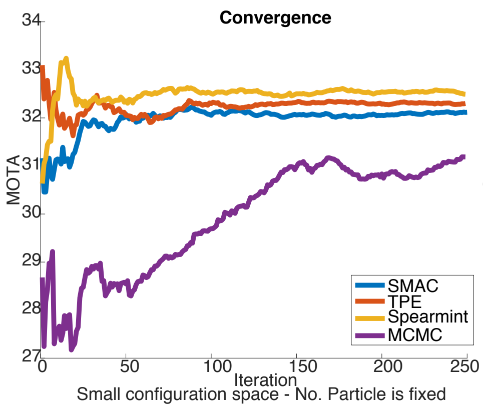
challenge[7], we observe that the approaches achieve a similar tracking precision (MOTP) with a standard deviation of 0.9 over 100. Nevertheless, the difference lies in the accuracy that shows a standard deviation of 5.3. By optimizing this metric, the ranking of the methods could change.

We compare the results of the tools against a baseline based on grid search. Here the idea is to discretize the space of the parameters and analyze which is the configuration that provides the best result. For this evaluation, we use the small configuration without fixing the number of particles (SNF). We consider two versions: (1) a grid search by brute force, called simply grid search, and (2) an Iterative optimization, where the same discretized space is used but one variable is optimized at a time. Due to the stochastic nature of the particle filter, each configuration is evaluated 30 times and the mean is taken as the estimated value for that configuration. The space is divided into b parts for each variable λ, therefore we have b^{λ_n} configurations to evaluate, where λ_n is the number of variables. We divide our space with $b = 3$ and $\lambda_n = 7$, thus we have 2187 configurations to be evaluated. It should be mentioned that this value is higher than the number of evaluations/iterations used by the tools, *e.g.* 2000.

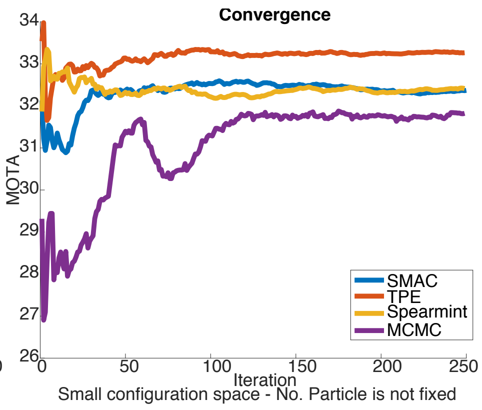
[7] https://motchallenge.net/results/2D_MOT_2015/

The evaluations are carried out using public datasets following a Non-exhaustive cross-validation validation technique. Here, videos are divided into two sets, one is a training set, which is used with the optimization tools to estimate the optimal hyper-parameters, and the other one for test, using the optimized parameters. This division allows avoiding over-fitting of the results and boosting the configurations that generalize better.

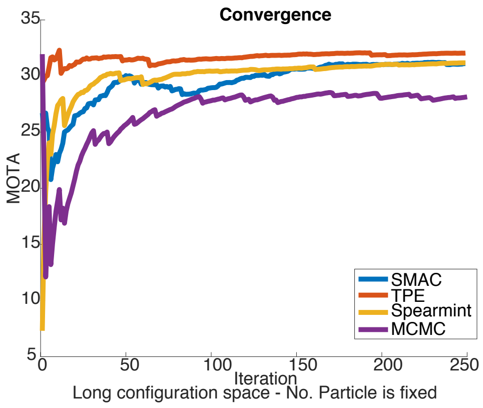
Due to the stochastic nature of the method, i.e. tracking framework, we evaluate several times (10) each possible configuration of all the tools. Thus, the results shown are the mean of all the evaluations. This allows to evaluate the tools in terms of stability and convergence speed. The latter is an important factor that can favor the use of one tool or another. We launch all the evaluations using a Dell Precision Tower 3620, with an Intel Xeon CPU v5 of 3.60GHz and 8 cores, 16 GB of RAM over a Linux system (Ubuntu 14.04).



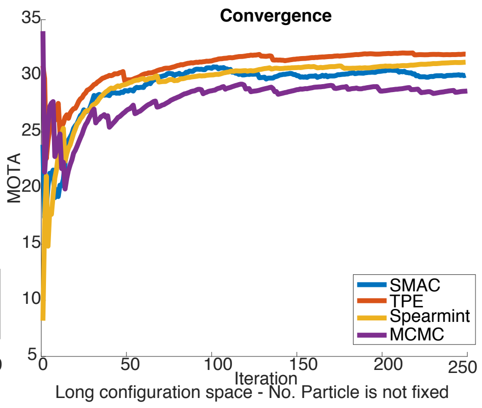
(a) SF



(b) SNF



(c) LF



(d) LNF

Fig. 5: **Convergence speed evaluation.** Cumulative MOTA of SMAC [27], TPE [3], MCMC and Spearmint [45] using ETH Sunny day sequence.

In traditional optimization, the algorithm converges once it reaches a stationary value of the cost function, but this is not possible when optimizing stochastic methods. In the literature there are plenty of termination criteria and selecting the most appropriate depends on the evaluated functions. In theory, the methods will converge to the optimal value when the number of iterations tends to infinity, following the central limit theorem. Therefore, instead of providing an automatic termination criterion, the tools end after a certain number of iterations, set by the user. SMAC, TPE and Spearmint benefit from additional iterations because that allows them to further explore the configuration space. In order to analyze the convergence of the tools, we use the weak convergence criterion that estimates the cumulative distribution of the measured parameters. We can see an example in Fig. 3 using the results of SMAC. The orange line shows the evaluations of the function (tracking framework), in terms of MOTA, and blue line is the cumulative distribution. We observe that after 50 iterations SMAC starts to stabilize. This means that most of the time SMAC is exploring zones with high MOTA values (33).

4.3 Results

We present the results of each hyper-parameter optimization tool using two challenging sequences, each one under two different contexts: static and dynamic cameras.

We analyze the tools in terms of: (1) convergence speed, (2)stability of the optimum value, (3) performance accuracy according to MOTA metric, (4) computational time and influence of (5) starting point, (6) training size and (7) number of particles. Finally, from these evaluations, we propose a ranking of the tools in Tab. 10.

Convergence speed. We analyze the weak convergence criterion for all the tools under different configurations S and L with the number of particle fixed F and non-fixed NF. Fig. 4 and 5 show the results. We observe that SMAC and TPE stabilize faster and to the highest values most of the time. Meanwhile, MCMC stabilizes slower than the others and Spearmint shows an average performance in most of the cases.

Stability. In Figs. 8 and 9 we analyze the performance stability of the tools for the four configurations SF, SNF, LF,

Table 1: **Accuracy.** Evaluation over the PETS09 S2L1 sequence.

Conf.	It	Training				Test			
		MCMC	SMAC	TPE	Spearmint	MCMC	SMAC	TPE	Spearmint
SF	250	32.75	33.45	38.20	33.47	21.65	**20.70**	**21.12**	21.94
SF	500	30.28	**33.55**	39.44	33.53	21.39	20.58	20.58	20.30
SF	1000	33.10	33.46	39.61	33.46	20.64	20.55	20.88	**21.96**
SF	2000	**33.63**	33.43	**39.79**	**34.01**	**21.97**	20.51	20.78	21.62
SNF	250	**35.04**	**33.97**	39.44	33.66	20.95	21.31	**21.67**	18.78
SNF	500	31.69	33.87	39.79	**33.99**	19.89	**22.56**	20.32	21.86
SNF	1000	30.99	33.80	37.68	33.96	18.72	21.76	21.45	20.53
SNF	2000	34.33	33.70	**40.14**	33.94	**22.59**	21.74	20.21	**22.21**
LF	250	34.51	33.61	36.97	33.87	21.05	21.99	**22.55**	22.12
LF	500	33.98	33.28	38.73	34.03	19.02	22.96	21.77	**22.56**
LF	1000	**35.56**	33.56	39.09	33.91	18.82	21.75	21.62	22.28
LF	2000	33.45	**33.84**	**39.09**	**34.23**	**22.34**	**22.16**	21.51	19.22
LNF	250	32.75	33.95	36.97	34.26	21.18	20.85	21.52	5.43
LNF	500	30.46	33.92	37.32	34.27	20.79	20.55	21.40	20.76
LNF	1000	**33.45**	33.99	38.56	34.25	20.75	**21.85**	21.41	**22.00**
LNF	2000	31.87	**34.01**	**40.14**	**34.51**	**22.06**	21.79	**21.97**	21.11
St.D		1.53	0.23	1.05	0.30	1.17	0.75	0.60	3.98

Table 2: **Accuracy.** Evaluation over ETH Sunny Day sequence.

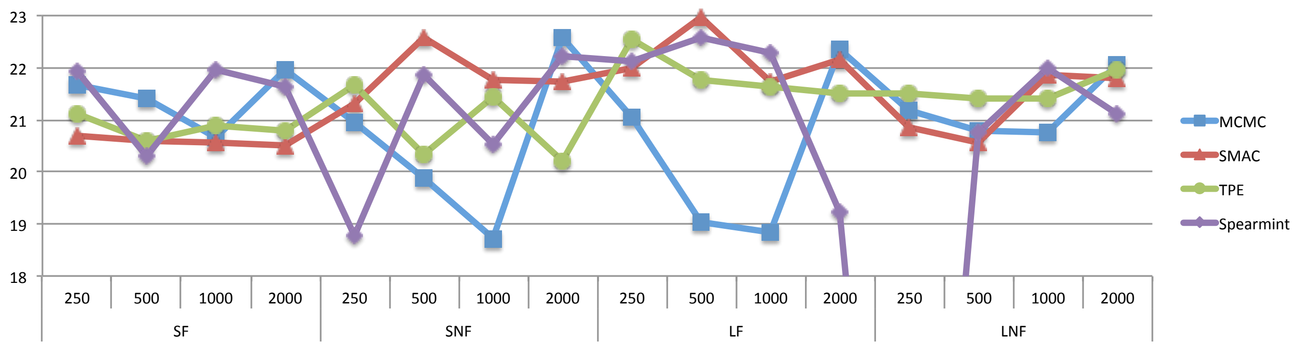
Conf.	It	Training				Test			
		MCMC	SMAC	TPE	Spearmint	MCMC	SMAC	TPE	Spearmint
SF	250	**42.39**	34.39	46.95	35.00	40.75	40.70	40.71	40.62
SF	500	11.68	35.16	**48.48**	**36.35**	29.91	40.84	40.62	40.65
SF	1000	41.12	35.25	46.70	35.98	**40.79**	40.88	**40.95**	40.64
SF	2000	25.13	**35.71**	48.22	36.23	35.09	**40.97**	40.69	**40.65**
SNF	250	**43.91**	34.96	45.43	36.56	40.72	40.62	**41.03**	40.52
SNF	500	12.69	33.73	**48.22**	36.34	39.83	40.77	40.66	**40.81**
SNF	1000	29.44	34.47	47.21	**37.03**	40.74	40.80	40.75	40.44
SNF	2000	20.05	**35.17**	47.21	35.86	**40.75**	**40.85**	40.59	40.60
LF	250	**43.91**	34.41	47.21	35.04	40.48	40.66	40.77	40.78
LF	500	23.86	**35.58**	47.21	35.04	13.17	40.66	**41.01**	**41.21**
LF	1000	27.16	35.30	47.46	37.08	**40.68**	40.73	40.50	40.84
LF	2000	14.98	34.66	**47.62**	**37.97**	34.04	**40.81**	40.67	40.58
LNF	250	**31.22**	35.33	46.45	35.92	**40.88**	40.89	40.79	**41.02**
LNF	500	14.72	33.75	**49.49**	36.05	36.34	40.72	**40.82**	40.23
LNF	1000	31.73	35.22	47.46	36.38	40.66	40.75	40.80	40.57
LNF	2000	22.59	**35.90**	48.73	**37.73**	40.22	**40.89**	40.75	36.30
St. D.		10.81	0.62	0.94	0.84	6.95	0.10	0.14	1.08

Table 3: Comparison between best and worst estimations computed by tools, Tabs. 1 and 2, and base-line grid search methods.

Tool	Result	PETS		ETH Sunny	
		Training	Test	Training	Test
MCMC	Best	35.56	33.98	43.91	59.51
SMAC	Best	34.01	35.13	35.90	59.64
TPE	Best	40.14	34.70	49.49	59.72
Spearmint	Best	34.51	34.33	37.97	59.98
MCMC	Worst	30.28	33.39	11.68	59.21
SMAC	Worst	33.28	34.01	33.73	59.40
TPE	Worst	36.97	33.44	45.43	59.42
Spearmint	Worst	33.46	32.73	35.00	59.17
Grid search	mean	28.64	30.90	32.25	56.6
It. Op.	mean	28.51	30.18	32.31	57.6

LNF. The evaluations are done with 1000 iterations, meaning that each tool evaluate 1000 times the cost function. The MOTA output is used to build up a box plot that represents MOTA distribution. Both Figs. show the median (red line), the first and third quantiles (blue box) and the outliers (red crosses). We can observe that all the boxes of MCMC are the bigger than the others.

Accuracy. The previous results indicate that TPE and SMAC have a better convergence rate with an efficient use of each iteration. We can observe in Tables 1 and 2 the results in terms of precision using the MOTA metric. We evaluate the tools for all the configurations with a fixed number of iterations using the training set (*i.e.* first part of the sequence). At the end, the tool gives a configuration and its associated



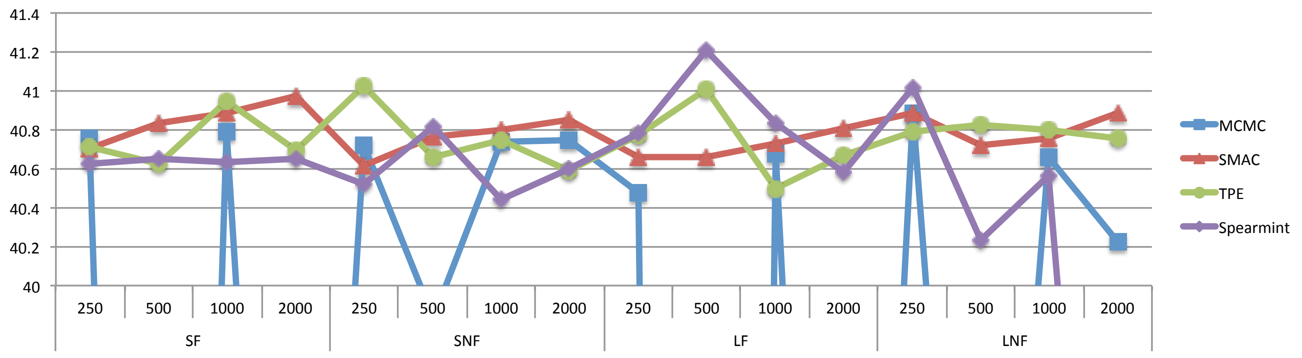


Fig. 6: **Accuracy evaluation with MOTA.** Each point on the line represents the MOTA value obtained for each configuration. Top: Evaluation using the test part of PETS2009 S2L1 sequence, see Tab. 1. Bottom: Evaluation using the test part of ETH Sunny Day sequence, see Tab. 2.

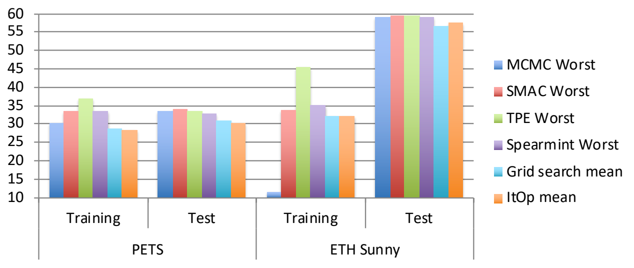


Fig. 7: Comparison between worst MOTA results of the tools and Grid search (last two columns of each group), see Tab. 3

MOTA value. We evaluate 10 times each tool and we show the mean value in the training column. This number offers a good compromise between the computation time and the number of evaluations needed to estimate the behavior of these stochastic tools. In all the cases, the variance is less than 0.2. Then, we evaluate 30 times the parameters computed by the tools using the test set, results are shown in the test column. Last row shows the mean standard deviation of each column. For a better visualization, Fig. 6 shows the MOTA values obtained in the test part for both tables.

The Tab. 3 summarizes the previous results, leaving only the best and the worst ones. At the same time, it includes a comparison with the proposed baseline. We analyze the whole sequences using the best parameters of each tools. The results are shown in Tables 4 and 5. In both cases, SMAC and TPE show the best results. The results displayed at the webpage of MOTChallenge are the average values across all sequences. Nevertheless, they provide the individual results of each single video. We calculate the mean and standard deviation of all provided results for the ETH and PETS sequences. The results are shown in the last two lines of the tables.

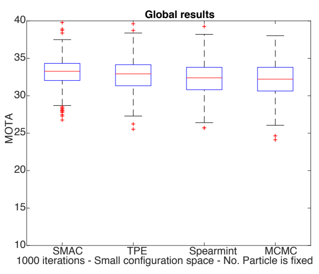
Computational time. Tab. 6 and Fig. 11 show the mean computational time used for each configuration (S, L) using the PETS sequence. The time of each iteration of MCMC is practically the same, having a linear increment with respect to the number of iterations. Meanwhile SMAC is a bit slower than MCMC, but it is faster than the rest of the tools. TPE has a good performance in comparison, but Spearmint computational time increases at each iteration, which could be a limitation. Although in our experiment we observe that Spearmint performs, in general, better with more iterations. The results with ETH sequence are similar and the only difference is, therefore, a different number of frames to process.

Table 4: **Accuracy.** Evaluation over the whole PETS09-S2L1 sequence with metrics: IDF1 - ID F1 Score, Rcll - Recall, Prcn - Precision, MT - Mostly tracked targets, ML - Mostly lost targets, FP - False positives, FN - False negatives (missed targets), ID Sw, - The total number of identity switches, MOTA - Multiple Object Tracking Accuracy, MOTP - Multiple Object Tracking Precision, MOTAL - Multi-object tracking accuracy in [0,100] with log10 (idswitches).

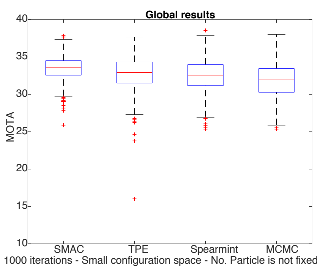
Method	Conf.	IDF1 (↑)	Rcll (↑)	Prcn (↑)	MT (↓)	ML (↓)	FP (↓)	FN (↓)	IDs (↓)	MOTA (↑)	MOTP (↑)	MOTAL (↑)
Grid search		39.8	72.4	64.2	6	2	1876	1283	47	30.9	68.5	32
It. Op.		39.38	72.01	65.13	7.4	1.5	1875.9	1261.5	46.3	30.18	68.36	32.81
MCMC	SF	43.22	75.07	65.12	7.27	0.43	1869.40	1158.60	39.97	**33.98**	69.37	**34.80**
MCMC	SNF	42.77	74.49	65.18	7.53	1.57	1849.30	1185.03	40.53	33.83	69.55	34.67
MCMC	LN	42.33	74.15	65.03	7.17	1.57	1853.27	1201.23	40.73	33.39	69.37	34.24
MCMC	LNF	42.38	74.29	65.24	7.07	1.83	1838.97	1194.60	40.57	33.85	69.47	34.68
SMAC	SF	42.41	74.88	64.78	7.57	0.67	1892.13	1167.50	44.13	33.21	69.39	34.12
SMAC	SNF	43.15	75.33	65.24	7.43	0.60	1865.17	1146.03	43.53	34.26	69.56	35.17
SMAC	LN	43.82	75.11	65.12	7.57	0.53	1868.90	1157.03	40.90	34.01	69.39	34.84
SMAC	LNF	43.97	75.80	65.60	7.60	0.20	1847.27	1125.07	41.93	**35.13**	69.61	**36.00**
TPE	SF	43.06	75.01	65.01	7.53	0.53	1876.47	1160.63	44.67	33.68	69.35	34.61
TPE	SNF	43.14	75.10	64.88	7.70	1.37	1888.47	1157.40	46.83	33.44	69.52	34.42
TPE	LN	44.05	75.09	65.15	7.57	0.53	1866.33	1157.53	41.30	34.05	69.33	34.89
TPE	LNF	43.69	75.49	65.44	7.33	0.43	1852.50	1139.30	42.27	**34.70**	69.57	**35.58**
Spearmint	SF	41.27	74.76	64.62	7.50	1.43	1901.47	1173.47	50.93	32.73	69.37	33.79
Spearmint	SNF	42.45	75.33	65.21	7.53	0.50	1867.40	1146.23	43.83	34.21	69.61	35.12
Spearmint	LN	43.36	74.90	64.91	7.53	0.83	1881.83	1165.90	42.40	33.50	69.41	34.37
Spearmint	LNF	42.75	75.73	65.33	7.54	0.53	1857.40	1141.13	43.13	**34.33**	69.61	**35.17**
MOTChallenge	Mean	-	83.29	81.24	14	-	1029.17	775	167	57.13	71.17	60.701
MOTChallenge	St. D.	-	5.541	10.77	2.08	-	894.224	261.054	108.5	16.23	0.442	15.376

Table 5: **Accuracy.** Evaluation over the whole ETH-SUNNY sequence with metrics: IDF1 - ID F1 Score, Rcll - Recall, Prcn - Precision, MT - Mostly tracked targets, ML - Mostly lost targets, FP - False positives, FN - False negatives (missed targets), ID Sw, - The total number of identity switches, MOTA - Multiple Object Tracking Accuracy, MOTP - Multiple Object Tracking Precision, MOTAL - Multi-object tracking accuracy in [0,100] with log10 (idswitches).

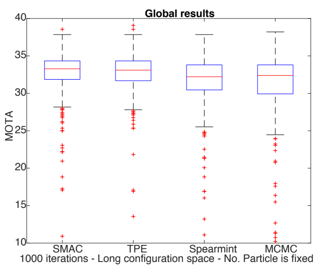
Method	Conf.	IDF1 (↑)	Rcll (↑)	Prcn (↑)	MT (↓)	ML (↓)	FP (↓)	FN (↓)	IDs (↓)	MOTA (↑)	MOTP (↑)	MOTAL (↑)
Grid search		50.95	69.91	84.38	8.78	9.1	242.17	560.67	11.35	56.6	77.21	56.81
It. Op.		51.34	70.34	85.29	8.93	9.07	230.83	551.04	11.02	57.6	77.96	57.93
MCMC	SF	53.02	71.67	85.95	9.10	9	217.40	526.50	11.10	**59.37**	78.19	59.90
MCMC	SNF	54	71.16	86.3	10.5	8.8	209.9	535.9	10.2	59.31	78.19	59.8
MCMC	LN	52.74	71.82	86.03	9.2	8.9	217.2	523.7	11.3	59.51	78.06	**60.07**
MCMC	LNF	52.15	71.28	86.1	9	9	213.8	533.5	10.6	59.21	77.93	59.71
SMAC	SF	52.78	71.76	86.14	9	9	214.5	524.7	10.4	**59.64**	77.97	**60.16**
SMAC	SNF	52.47	71.72	86.03	9.4	9	216.4	525.4	11.5	59.46	78.2	60.02
SMAC	LN	53.2	71.46	86.27	8.9	9	211.2	530.4	10.5	59.52	77.9	60.01
SMAC	LNF	52.89	71.72	86.01	9.10	8.9	216.90	525.80	11.60	59.40	78.06	59.96
TPE	SF	52.73	71.74	86.16	9.1	9	214.5	524.9	10.9	59.6	78.07	60.16
TPE	SNF	52.92	71.77	86.25	9.2	8.9	212.4	524.6	11.5	**59.72**	78.09	**60.27**
TPE	LN	52.89	71.6	86.11	9	9	214.7	527.9	11.4	59.42	78.07	59.97
TPE	LNF	53.02	71.78	86.2	8.8	9	213.6	524.2	11.2	59.69	77.92	60.24
Spearmint	SF	52.69	71.62	85.79	9	9	220.6	527.2	10.8	59.17	78.13	59.7
Spearmint	SNF	52.87	71.65	86.04	9	9	216.2	526.9	11	**59.41**	78.07	**59.95**
Spearmint	LN	52.3	71.77	86.25	9.1	9	212.8	524.5	11.2	59.31	78.25	59.26
Spearmint	LNF	52.65	71.85	86.48	9.2	9	218.7	523.1	11.2	58.98	78.41	59.55
MOTChallenge	Mean	-	61.13	86.26	5.33	8.17	211.67	736.33	41.50	47.66	77.77	49.78
MOTChallenge	St. D.	-	3.22	10.40	2.21	1.34	211.58	63.17	40.96	10.87	1.88	10.64



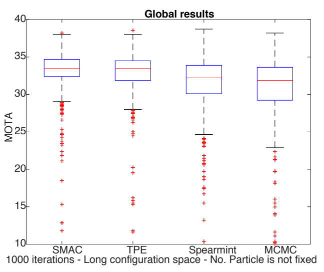
(a) SF



(b) SNF

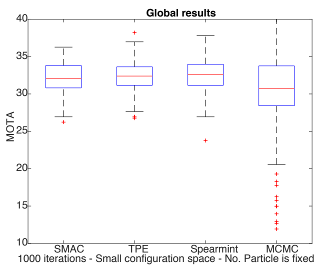


(c) LF

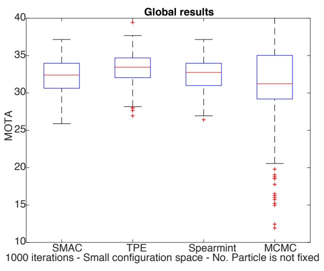


(d) LNF

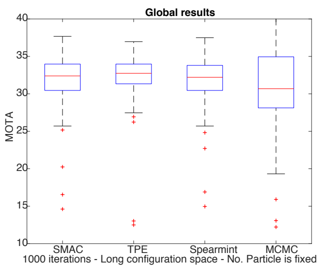
Fig. 8: **Performance stability.** Analysis using PETS sequence. We show the distribution of the MOTA values obtained with 1000 iterations of each tool: SMAC [27], TPE [3], MCMC and Spearmint [45].



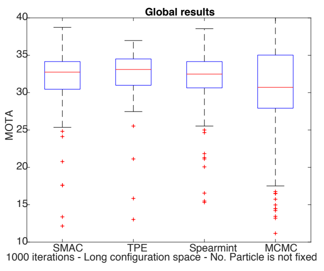
(a) SF



(b) SNF

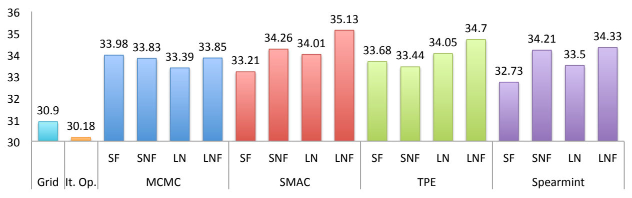


(c) LF

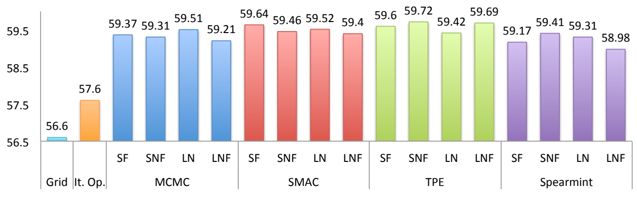


(d) LNF

Fig. 9: **Performance stability.** Analysis using ETH Sunny day sequence. We show the distribution of the MOTA values obtained with 1000 iterations of each tool: SMAC [27], TPE [3], MCMC and Spearmint [45].



(a) PETS09



(b) ETH-SUNNY

Fig. 10: **Accuracy.** MOTA metric evaluation of SMAC [27], TPE [3], MCMC and Spearmint [45], see Tabs. 4 and 5.

Influence of starting point. In optimization, the initial parameters could influence the final results. TPE and Spearmint use the lower limits of S and L as starting point by default and this cannot be changed. Nevertheless, SMAC and MCMC allow setting the initial point. We evaluate 10 parameters randomly generated 5 times each using the training set with 500 iterations. We use the large configuration without fixing the number of particles because this setting has shown the best performance. The tool results are evaluated 30 times using the test set. We calculate the mean and standard deviation of the MOTA metric of the training and test set. The results are shown in Table 7 for PETS and ETH sequences respectively.

Influence of the training size. The tool performance increases when more data is available. However, this is not possible in all cases. We analyze how training size impacts the results. We evaluate 4 sizes corresponding to the initial 5, 10, 15 and 20 percent of the video. Table 8 shows the results for both sequences, we observe that all the tools work well after 10 percent. MCMC, SMAC and TPE increase the precision when more frames are used.

Influence of the number of particles. In our evaluations, the number of particles is fixed to 30. This is motivated because this number gives a good trade-off between computational time and precision. However, a variation of the value changes the result performance. We analyze this behavior by testing different amounts of particles under the large configuration. We focus on the analysis of SMAC and TPE because these have shown the best performance so far. The optimized hyper-parameters are then evaluated over the whole sequence of PETS and the mean of 30 evaluations is shown in the Table 9.

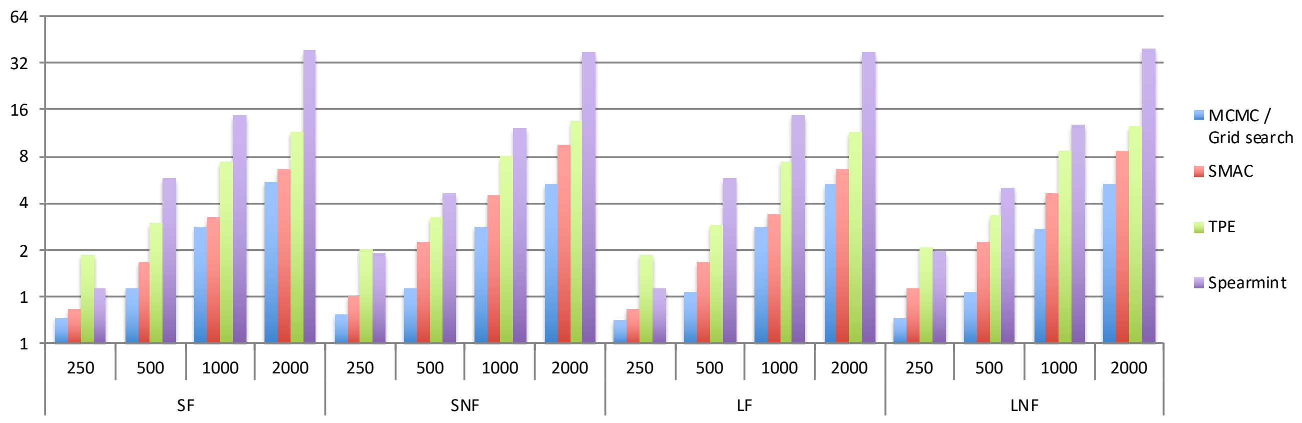


Fig. 11: **Computational time**, in hours, used by the tools with the PETS09 S2-L1 sequence.

Table 6: **Computational time**, in hours, used by the tools with the PETS09 S2-L1 sequence.

		Time (hours)			
Conf.	No. Iter.	MCMC	SMAC	TPE	Spearmint
SF	250	0.73	0.83	1.86	1.14
	500	1.12	1.69	2.96	5.83
	1000	2.79	3.27	7.34	14.78
	2000	5.41	**6.72**	**11.60**	38.36
SNF	250	0.77	1.02	2.02	1.92
	500	1.14	2.26	3.21	4.64
	1000	2.86	4.46	8.00	12.24
	2000	5.37	**9.42**	**13.38**	37.17
LF	250	0.70	0.83	1.85	1.15
	500	1.09	1.69	2.92	5.80
	1000	2.83	3.40	7.31	14.71
	2000	5.33	**6.63**	**11.52**	37.17
LNF	250	0.72	1.15	2.08	1.98
	500	1.08	2.26	3.31	4.99
	1000	2.77	4.60	8.66	12.78
	2000	5.24	**8.73**	**12.57**	39.95

Table 7: **Influence of starting point.** Mean MOTA results with 10 parameter sets randomly generated. We test them with the large configuration with not fixed particles and 500 iterations.

		Training		Test	
Sequence	Method	MOTA	St. D.	MOTA	St. D.
PETS	MCMC	33.95	0.72	19.13	1.56
PETS	SMAC	34.05	0.08	22.39	0.66
ETH	MCMC	19.77	6.37	27.92	14.71
ETH	SMAC	34.81	0.76	40.23	0.30

4.4 Discussion

Hyper-parameter optimization tools are designed to support algorithms that require tuning multiple variables. Therefore, they must be intuitive to use and adapt well to different scenarios. This paper gives an original insight of relevant tools for MOT frameworks and this section highlights some interesting aspect of the tools based on the results previously shown.

Convergence speed. The analysis of weak convergence criteria (Figs. 4 and 5) shows interesting results. In this case, both SMAC and TPE find the best values of MOTA most of the time and converge faster than the rest on any configuration. MCMC baseline takes longer to stabilize and the cumulative distribution reaches lower MOTA values compared to the others. In the short configuration, Spearmint can find high MOTA values in a few iterations but it takes more time to process the large configuration.

We test the tools with two configurations (see section 4.2.1): short and large. The small configuration is selected around where it is expected to have the optimal parameter configuration. It is set based on expert knowledge about the parameters of the MOT framework. This prior information helps optimization tools to converge faster with more stable MOTA values compared to the larger configuration. However, if these limits cannot be clearly delineated or if we know little about the effect of the parameters on the system, we can use a larger configuration and get the same results with just more iterations.

We conclude that both TPE and SMAC generally converge to the optimal MOTA value in a faster way than the other approaches. Therefore, these tools are recommended when a result with few function evaluations is required.

One way to reduce computing time is to optimize fewer variables. For this case we analyze two configurations, one labeled as F (Fixed) with 7 parameters and the other as NF (Non-Fixed) with 6, which keeps the number of particles fixed. The variables are set at values close to the optimum, which requires expert knowledge about the tracking method. Otherwise the results do not reach an optimal performance. Although this reduces the complexity of the optimization, we observe that the results labeled as NF converge more

Table 8: **Influence of the training size.** Performance evaluation changing the size of training set.

Seq.	%	Training				Test			
		MCMC	SMAC	TPE	Spearmint	MCMC	SMAC	TPE	Spearmint
PETS	5	30.973	30.69	37.85	7.085	17.5833	21.35	20.1	18.58
PETS	10	31.135	31.18	**39.09**	34.3	17.6033	21.88	19.97	21.21
PETS	15	31.527	32.46	37.15	**34.85**	18.9267	22.09	22.6	**22.56**
PETS	20	**31.89**	**33.8**	38.2	32.11	**19.4733**	**22.19**	**22.76**	20.87
ETH	5	6.7961	17.88	27.42	**33.13**	18.2	28.09	18.2	38.45
ETH	10	21.224	34.87	48.58	33.11	19.2	38.26	19.2	**40.08**
ETH	15	23.529	35.42	**46.36**	33.07	38.95	40.17	38.95	39.32
ETH	20	**37.223**	**39.28**	40.17	32.82	**39.71**	**40.53**	**39.19**	39.94

quickly than the others. We observe the same phenomenon with other variables, if we fix rwn_* then dH and np tend to have a higher value. This is because the tools use these variables to compensate the rest. More clearly, a high number of particles are considered when the other parameters do not perform well in the tracking system.

Stability. Ideally, tools should explore the configuration space efficiently, evaluating promising parameters. If this is the case, we expect that the evaluations follow a Gaussian distribution with a small variance. Otherwise, the variance increases when evaluating suboptimal parameters, which translates in a waste of iterations. The box plot quantifies this analysis in the Figs. 8-9. This shows how MCMC is, overall, unstable to achieve a proper solution and constantly evaluating, and accepting, hyper-parameters with low MOTA values. On the other hand, TPE is more robust, using wisely each iteration to explore promising parameters.

SMAC outperforms the rest with respect to PETS dataset, showing a smaller variance than the others. It finds stable parameters that generalize well in this scenario with static camera. However, SMAC has problems to handle the moving camera of ETH sequence. This is because it tries to optimize as quickly as possible the variables related to distance. This makes it oscillate between configurations adapted for moments where the targets are far from the camera and configurations for targets close to the camera. Thus, it is advisable to use SMAC or TPE for methods whose cost function contains noise, i.e. stochastic methods and TPE is more reliable with non-static sequences.

Accuracy. From Tabs. 1-2, we observe that the results of MCMC are unstable, reaching different values in each iteration. This is because their implementation does not consider that the evaluation of cost function has noise, *i.e.* two evaluations of the same parameters result in two different MOTA values. This is most clearly seen in Fig. 6 where the MCMC line (blue) changes drastically in each configuration. Then, its performance is affected by the stochastic output of the particle filter-based tracker. One way to overcome this issue is to approximate the distribution of the parameter set using, for example, MCMC. This will require a larger number of evaluations, which makes it computationally expensive. In addition, training results show that MCMC needs few iterations to converge in any configuration (small or large and fixed or non-fixed) but, on the contrary, a greater number of iterations generates over-fitting and reduces its performance.

Spearmint provides more stable results, with a standard deviation of 0.3, see Fig. 6, but the given parameters do not generalize well the rest of the sequence. Moreover, the number of iterations needed to obtain good results is proportional to the complexity of the configuration, *i.e.* the large non-fixed configuration needs more iterations to reach the optimal value. Unlike, TPE converges slowly but it provides robust parameters, presenting the highest values on the training set. However, this is due to TPE reports the highest value found meanwhile SMAC and Spearmint report the expected mean value. SMAC appears to be more stable than the others in both training and testing sets, red line on Fig. 6. The results from the training set are generally consistent with those from the testing set.

Tab. 3 summarizes the information from the previous tables and leaves only the best and worst results of both the datasets and the evaluations sets. Last two lines show the results of the baseline Grid search and the iterative optimization version. In both cases, test results are the lowest even that MCMC, this is more evident when observing the Fig. 7. This is because the grid is very large and its results are suboptimal, this could be improved by dividing more the grid but this will increase the number of evaluations.

Tables 4 and 5 compare our results with those provided by MOTChallenge. We focus on the MOTA metric because it is the most used to compare algorithms [30,21] so Fig. 10 summarizes this metric for both datasets. If we compare the results of the methods against the ones obtained by MOTChallenge, we observe that in the results of PETS sequence are below the mean, meanwhile, with the ETH sequence, MOTA is above the mean. This is due to the fact that the sequence of PETS has been evaluated by dozens of methods, each one contributing to a slight increase with respect to the previous one. However, ETH sequence has been

Table 9: **Influence of the number of particles.** Evaluation on PETS09 sequence fixing the number of particles.

Method	Conf.	No. Part.	Time (hrs.)	IDF1 (↑)	Rcll (↑)	Prcn (↑)	MT (↓)	ML (↓)	FP (↓)	FN (↓)	IDs (↓)	MOTA (↑)	MOTP (↑)	MOTAL (↑)
TPE	LF	10	1.94	42.68	73.99	64.12	7.20	0.90	1923.9	1209	44	31.64	69.09	32.55
TPE	LF	30	2.17	44.05	75.09	65.15	7.57	0.53	1866.3	1158	41.30	34.05	69.33	34.9
TPE	LF	50	2.38	43.91	75.49	65.51	7.50	0.40	1847.1	1139	40.3	**34.88**	69.40	35.7
TPE	LF	70	2.69	42.09	74.92	65.01	7.60	1.20	1873.5	1165	45	34.63	69.57	35.56
TPE	LF	100	3.17	42.55	75.69	65.39	7.30	0.10	1861	1130	41.6	34.74	69.66	**35.62**
SMAC	LF	10	1.38	44.33	74.38	64.74	6.90	0.80	1883.9	1190	38.7	33.02	69.10	33.82
SMAC	LF	30	1.69	43.82	75.11	65.12	7.57	0.53	1868.9	1157	40.9	34.01	69.39	34.84
SMAC	LF	50	1.82	42.47	75.46	65.36	8.00	1.00	1857.7	1141	44.1	34.54	69.54	35.44
SMAC	LF	70	2.56	45.02	75.62	65.46	7.20	0.40	1854.7	1132	40.5	34.86	69.57	35.71
SMAC	LF	100	2.66	43.52	75.82	65.46	7.80	0.60	1859.4	1124	43.9	**34.87**	69.55	**35.77**

Tool	Convergence	Stability	Accuracy	CPU cost	Init. Cond.	Training size	Accessibility	Rank
MCMC [8]	+	+	+	+++	+	+	+	4
Spearmint [45]	++	++	++	+	-	++	+	3
SMAC [27]	+++	++	+++	+++	+++	+++	+++	1
TPE [3]	+++	+++	++	++	-	+++	++	2

Table 10: Evaluation summary: (+) Good, (++) Better, (+++) Best.

analyzed with only few methods, which are not at the top of the ranking.

Our system does not attempt to compete with the methods in the literature, but to show how the same framework can outperform itself by using better parameters. In Tab. 4, we observe that the results of the methods have a MOTA value of 3 - 4 percent higher than the grid search method. It may not seem to be much, but if we check the MOTChallenge ranking, scores between approaches are only different by a few decimals. Therefore, this slight change in several sequences could end in an increase of 5 positions in the ranking. In general, all methods give a good performance, but among them SMAC gives a more accurate result. Therefore, we recommend its use when the main goal is the accuracy.

We analyze the performance of the tools considering other metrics and making a combination between them. However, the MOTA metric includes many features that range from false positives, false negatives and identity switch. Those are strongly related to other metrics and therefore the results obtained reached the same conclusions as when using MOTA exclusively.

Computational time. The time used for each tool plays an important role when selecting one. When comparing the configurations F and NF of Fig. 11, we observe MCMC and Spearmint have a similar cost. However, SMAC and TPE present a different behavior. The CPU time is larger when the number is not fixed, the Table 6 highlights this phenomenon. This is because both tools need to build a model for 7 parameters instead of 6. Thus, the computational time of both tools will increase according to the number of parameters to be optimized. In this case, SMAC is the one with the largest increment.

Computational time of the grid search-based methods is linear, as same as MCMC, but its accuracy is not good since the size of the grid is small. Increasing the grid size is ideal to improve this quality however this adversely affects the computational time. For example, changing the value of b from 3 to 4 results in more than sixteen thousand configurations to evaluate, and since the function is stochastic each configuration must be evaluated several times before being accepted.

Setup time. Both SMAC and Spearmint adjust several files that control the optimization of the tools, variable search space and storage of the results. This data file processing is called scenario and it creation could take a few minutes to an expert in these tools, but for a new user it can take a couple of hours to understand the required file structures. On the contrary, TPE is more intuitive since it is a function that receives the configurations of the scenario as arguments.

Influence of starting point. The choice of starting point influences the outcome of most optimization systems. A good method should be able to handle this and converge to the global optimum, avoiding getting stuck in the local minimum. We analyze this aspect and comparing Table 7 against Tables 1 and 2, we observe that SMAC and MCMC show a similar behavior regardless the initial parameters. Moreover, the analysis of SMAC in terms of convergence shows that the only difference is that the number of iterations varies with respect to how far it is from the optimal minimum. Although MCMC still has problems to converge, staying on local minimums for several iterations and over-fitting problems.

Influence of training size. It is natural that increasing the size of training data also increases the accuracy. Our results show this same effect. Also, we can see that starting from 10 percent the increase is small. This is because the initial 10 percent of the evaluated sequences describes in a general way the movement of pedestrians in the scene and likewise the optimal parameters that should be used.

Influence of the number of particles. The number of particles plays an important role in this type of trackers. We observe that when this number is fixed, the results improve when more particles are used, but this also increases the computational time. Furthermore, analyzing the parameter sets evaluated by the tools without fixing this number, we noticed that many good candidates maintained a low number of particles, in the range of 10 to 50.

Accessibility. Overall, these tools define the search space but only SMAC and (our implementation of) MCMC can set the initial parameters. In contrast, TPE and Spearmint use the lower limit as initial position making them simpler to configure. SMAC has the best ratio between CPU time and performance, followed by TPE. On the contrary, Spearmint has a constant increase in time in each iteration, which is a problem because the results are better with more iterations. TPE has the advantage that it can be called as a function, making it easier to handle. In comparison, SMAC and Spearmint are more complex and require creating a set of files and folders with specific formats.

The documentation is a vital part when we decide to use a new tool. In this aspect, SMAC surpasses the rest with detailed official documentation, active discussion forums and developer support. MCMC has a strong background in the literature with many examples for different topics. TPE has a small but good documentation of its use. However, Spearmint's official documentation is almost non-existent, mostly limited to the installation and how to launch a simple example. Additionally, SMAC and Spearmint collect information from each iteration, which is available so that the user can observe and analyze it. SMAC is more organized, separating specific data in many easy-to-understand files. Meanwhile, Spearmint provides summary files of each iteration and a database with the global information. TPE and MCMC do not provide information as exhaustive as the other two, but simple results such as the parameters and the value of the cost function in each iteration.

From the above, we created Tab. 10 that summarizes our experience when using these tools. We consider that SMAC gives the best compromise with respect to our criteria mentioned in Tab. 10. It is the easiest to use despite the number of files to configure. The documentation is extensive and detailed and it has an active community supporting it. In addition, SMAC shows to be more stable in both convergence efficiency and exploration, which is important for the repeatability of the results.

We consider TPE as a good option but we rank it in second place. It offers a wide range of possibilities but the limited documentation makes it difficult to use for different scenarios. It also has a good convergence rate, even surpassing SMAC in some cases. Spearmint gives good performance but, according to our results, not as SMAC and TPE. Furthermore, the small documentation and the computational time are characteristics that make it difficult to use. However, MCMC presents the worst performance and therefore we rank it as number four. In this case, MCMC is not available as an optimization library and requires to be implemented. The results are fair and it is the fastest of all the tools, but it is difficult to set the correct number of iterations to avoid over-fitting.

5 Conclusions

In this paper, we have presented a comparative study of four relevant hyper-parameter optimization approaches in the context of MOT system. The tools are reviewed with respect to performance criteria, accessibility, computational cost time, among others. To the best of our knowledge, there are no comparative studies of optimization tools that study the tuning influence over tracking systems. This application is an example but can be extended to others.

We have shown how the same application can provide better results by using a better combination of parameters, and how these can be found using expert tools. Our goal is to introduce the reader these four optimization methods, with their respective tools, and motivate their use with respect to a criterion provided in the Tab. 10. Thus, new methods that use optimized hyper parameters will give results that reflect their maximum potential. We have highlighted the strengths and weaknesses of each as detailed as possible considering many criteria.

References

1. Berclaz, J., Turetken, E., Fleuret, F., Fua, P.: Multiple object tracking using k-shortest paths optimization. IEEE Trans. on Pattern Analysis and Machine Intelligence (2011)
2. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. J. Mach. Learn. Res. **13**, 281–305 (2012). URL http://dl.acm.org/citation.cfm?id=2188385.2188395
3. Bergstra, J.S., , Bardenet, R., , Bengio, Y., Kégl, B.: Algorithms for hyper-parameter optimization. In: J. Shawe-Taylor, R.S. Zemel, P.L. Bartlett, F. Pereira, K.Q. Weinberger (eds.) Advances in Neural Information Processing Systems, vol. 24, pp. 2546–2554 (2011)
4. Bergstra, J.S., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for hyper-parameter optimization. In: J. Shawe-Taylor, R.S. Zemel,

P.L. Bartlett, F. Pereira, K.Q. Weinberger (eds.) Advances in Neural Information Processing Systems 24, pp. 2546–2554. Curran Associates, Inc. (2011)
5. Bernardin, K., Stiefelhagen, R.: Evaluating multiple object tracking performance: the clear mot metrics. EURASIP Journal on Image and Video Processing **2008**(1), 1–10 (2008)
6. Bewley, A., Ge, Z., Ott, L., Ramos, F., Upcroft, B.: Simple online and realtime tracking. In: 2016 IEEE International Conference on Image Processing (ICIP), pp. 3464–3468 (2016). DOI 10.1109/ICIP.2016.7533003
7. Binitha, S., Sathya, S.: A survey of bio inspired optimization algorithms **2**, 137–151 (2012)
8. Bishop, C.M.: Pattern Recognition and Machine Learning (Information Science and Statistics). Springer-Verlag New York, Inc., Secaucus, NJ, USA (2006)
9. Boussad, I., Lepagnot, J., Siarry, P.: A survey on optimization metaheuristics. Information Sciences **237**, 82 – 117 (2013). Prediction, Control and Diagnosis using Advanced Neural Computations
10. Breitenstein, M.D., Reichlin, F., Leibe, B., Koller-Meier, E., Van Gool, L.: Online multiperson tracking-by-detection from a single, uncalibrated camera. IEEE Trans. on Pattern Analysis and Machine Intelligence **33**(9), 1820–1833 (2011)
11. Brunetti, A., Buongiorno, D., Trotta, G.F., Bevilacqua, V.: Computer vision and deep learning techniques for pedestrian detection and tracking: A survey. Neurocomputing **300**, 17 – 33 (2018). DOI https://doi.org/10.1016/j.neucom.2018.01.092
12. Burgard, W., Brock, O., Stachniss, C.: Active Policy Learning for Robot Planning and Exploration under Uncertainty, pp. 352–. MIT Press (2008). URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6280104
13. Chodorow, K., Dirolf, M.: MongoDB: The Definitive Guide, 1st edn. O'Reilly Media, Inc. (2010)
14. Collins, R.T., Carr, P.: Hybrid stochastic / deterministic optimization for tracking sports players and pedestrians. In: D. Fleet, T. Pajdla, B. Schiele, T. Tuytelaars (eds.) Computer Vision – ECCV 2014, pp. 298–313. Springer International Publishing, Cham (2014)
15. Dalal, N., Triggs, B.: Histograms of oriented gradients for human detection. In: In CVPR, pp. 886–893 (2005)
16. Dollr, P., Appel, R., Belongie, S., Perona, P.: Fast feature pyramids for object detection. IEEE Transactions on Pattern Analysis and Machine Intelligence **36**(8), 1532–1545 (2014). DOI 10.1109/TPAMI.2014.2300479
17. Domhan, T., Springenberg, J.T., Hutter, F.: Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In: Proceedings of the 24th International Conference on Artificial Intelligence, IJCAI'15, pp. 3460–3468. AAAI Press (2015). URL http://dl.acm.org/citation.cfm?id=2832581.2832731
18. Dong, X., Shen, J., Wang, W., Liu, Y., Shao, L., Porikli, F.: Hyperparameter optimization for tracking with continuous deep q-learning. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2018)
19. Dubuisson, S., Gonzales, C.: A survey of datasets for visual tracking. Machine Vision and Applications **27**(1), 23–52 (2016). DOI 10.1007/s00138-015-0713-y. URL https://doi.org/10.1007/s00138-015-0713-y
20. Ess, A., Leibe, B., Gool, L.V.: Depth and appearance for mobile scene analysis. In: 2007 IEEE 11th International Conference on Computer Vision, pp. 1–8 (2007). DOI 10.1109/ICCV.2007.4409092
21. Ferryman, J., Ellis, A.L.: Performance evaluation of crowd image analysis using the pets2009 dataset. Pattern Recognition Letters **44**, 3 – 15 (2014). DOI https://doi.org/10.1016/j.patrec.2014.01.005. Pattern Recognition and Crowd Analysis
22. Ferryman, J., Shahrokni, A.: Pets2009: Dataset and challenge. In: Twelfth IEEE Int. Workshop on Performance Evaluation of Tracking and Surveillance, pp. 1–6 (2009). DOI 10.1109/PETS-WINTER.2009.5399556
23. Floudas, C.A., Gounaris, C.E.: A review of recent advances in global optimization. Journal of Global Optimization **45**(1), 3 (2008). DOI 10.1007/s10898-008-9332-8. URL https://doi.org/10.1007/s10898-008-9332-8
24. Franceschi, L., Donini, M., Frasconi, P., Pontil, M.: Forward and reverse gradient-based hyperparameter optimization. In: D. Precup, Y.W. Teh (eds.) Proceedings of the 34th International Conference on Machine Learning, *Proceedings of Machine Learning Research*, vol. 70, pp. 1165–1173. PMLR, International Convention Centre, Sydney, Australia (2017)
25. Guo, L.: Stability of recursive stochastic tracking algorithms. SIAM Journal on Control and Optimization **32**(5), 1195–1225 (1994). DOI 10.1137/S0363012992225606. URL https://doi.org/10.1137/S0363012992225606
26. Haftka, R.T., Villanueva, D., Chaudhuri, A.: Parallel surrogate-assisted global optimization with expensive functions – a survey. Structural and Multidisciplinary Optimization **54**(1), 3–13 (2016). DOI 10.1007/s00158-016-1432-3. URL https://doi.org/10.1007/s00158-016-1432-3
27. Hutter, F., , Hoos, H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Proc. of LION-5, pp. 507–523 (2011)
28. Hutter, F., , Hoos, H., Leyton-Brown, K.: An efficient approach for assessing hyperparameter importance. In: Proceedings of the 31st International Conference on International Conference on Machine Learning, *ICML'14*, vol. 32, pp. I–754–I–762 (2014)
29. Kim, K., Davis, L.: Multi-camera tracking and segmentation of occluded people on ground plane using search-guided particle filtering. Computer Vision–ECCV 2006 pp. 98–109 (2006)
30. Leal-Taixé, L., Milan, A., Reid, I., Roth, S., Schindler, K.: MOTChallenge 2015: Towards a benchmark for multi-target tracking (2015). URL http://arxiv.org/abs/1504.01942
31. Lizotte, D., Wang, T., Bowling, M., Schuurmans, D.: Automatic gait optimization with gaussian process regression. In: Proceedings of the 20th International Joint Conference on Artifical Intelligence, IJCAI'07, pp. 944–949. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2007). URL http://dl.acm.org/citation.cfm?id=1625275.1625428
32. Luo, G.: A review of automatic selection methods for machine learning algorithms and hyper-parameter values. Network Modeling Analysis in Health Informatics and Bioinformatics **5**(1), 18 (2016). DOI 10.1007/s13721-016-0125-6. URL https://doi.org/10.1007/s13721-016-0125-6
33. Luo, W., Zhao, X., Kim, T.: Multiple object tracking: A review. CoRR **abs/1409.7618** (2014). URL http://arxiv.org/abs/1409.7618
34. Maggio, E., Taj, M., Cavallaro, A.: Efficient multitarget visual tracking using random finite sets. IEEE Transactions on Circuits and Systems for Video Technology **18**(8), 1016–1027 (2008). DOI 10.1109/TCSVT.2008.928221
35. Maurice, C., Madrigal, F., Lerasle, F.: Hyper-optimization tools comparison for parameter tuning applications. In: 2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS), pp. 1–6 (2017). DOI 10.1109/AVSS.2017.8078499
36. Minton, S., Johnston, M.D., Philips, A.B., Laird, P.: Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. Artificial Intelligence **58**(1-3), 161–205 (1992)
37. Nocedal, J., Wright, S.J.: Numerical Optimization, second edition. World Scientific (2006)
38. Pazhaniraja, N., Paul, P.V., Roja, G., Shanmugapriya, K., Sonali, B.: A study on recent bio-inspired optimization algorithms. In: 2017 Fourth International Conference on Signal Processing,

Communication and Networking (ICSCN), pp. 1–6 (2017). DOI 10.1109/ICSCN.2017.8085674
39. Qin, Z., Shelton, C.R.: Improving multi-target tracking via social grouping. In: 2012 IEEE Conference on Computer Vision and Pattern Recognition, pp. 1972–1978 (2012). DOI 10.1109/CVPR.2012.6247899
40. Shahriari, B., Swersky, K., Wang, Z., Adams, R.P., de Freitas, N.: Taking the human out of the loop: A review of bayesian optimization. Proceedings of the IEEE **104**(1), 148–175 (2016). DOI 10.1109/JPROC.2015.2494218
41. Shan, S., Wang, G.G.: Survey of modeling and optimization strategies to solve high-dimensional design problems with computationally-expensive black-box functions. Structural and Multidisciplinary Optimization **41**(2), 219–241 (2010). DOI 10.1007/s00158-009-0420-2
42. Shen, J., Liang, Z., Liu, J., Sun, H., Shao, L., Tao, D.: Multiobject tracking by submodular optimization. IEEE Transactions on Cybernetics pp. 1–12 (2018). DOI 10.1109/TCYB.2018.2803217
43. Shen, J., Yu, D., Deng, L., Dong, X.: Fast online tracking with detection refinement. IEEE Transactions on Intelligent Transportation Systems **19**(1), 162–173 (2018). DOI 10.1109/TITS.2017.2750082
44. Smeulders, A.W.M., Chu, D.M., Cucchiara, R., Calderara, S., Dehghan, A., Shah, M.: Visual tracking: An experimental survey. IEEE Transactions on Pattern Analysis and Machine Intelligence **36**(7), 1442–1468 (2014). DOI 10.1109/TPAMI.2013.230
45. Snoek, J., Larochelle, H., Adams, R.P.: Practical bayesian optimization of machine learning algorithms. In: Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2, NIPS'12, pp. 2951–2959. Curran Associates Inc., USA (2012)
46. Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., Patwary, M., Prabhat, M., Adams, R.: Scalable bayesian optimization using deep neural networks. In: F. Bach, D. Blei (eds.) Proceedings of the 32nd International Conference on Machine Learning, *Proceedings of Machine Learning Research*, vol. 37, pp. 2171–2180. PMLR, Lille, France (2015)
47. Tsamardinos, I., Rakhshani, A., Lagani, V.: Performance-estimation properties of cross-validation-based protocols with simultaneous hyper-parameter optimization. In: A. Likas, K. Blekas, D. Kalles (eds.) Artificial Intelligence: Methods and Applications, pp. 1–14. Springer International Publishing, Cham (2014)
48. Wang, Z., Zoghi, M., Hutter, F., Matheson, D., de Freitas, N.: Bayesian optimization in high dimensions via random embeddings. In: International Joint Conferences on Artificial Intelligence (IJCAI) - Distinguished Paper Award (2013). URL http://www.cs.ubc.ca/ hutter/papers/13-IJCAI-BO-highdim.pdf
49. Watada, J., Musa, Z., Jain, L.C., Fulcher, J.: Human tracking: A state-of-art survey. In: R. Setchi, I. Jordanov, R.J. Howlett, L.C. Jain (eds.) Knowledge-Based and Intelligent Information and Engineering Systems, pp. 454–463. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)
50. Weise, T.: Global Optimization Algorithms - Theory and Application, second edn. Self-Published (2009). URL http://www.it-weise.de/. Online available at http://www.it-weise.de/
51. Wu, Y., Lim, J., Yang, M.H.: Object tracking benchmark. IEEE Transactions on Pattern Analysis and Machine Intelligence **37**(9), 1834–1848 (2015). DOI 10.1109/TPAMI.2014.2388226
52. Yan, C., Xie, H., Chen, J., Zha, Z.J., Hao, X., Zhang, Y., Dai, Q.: An effective uyghur text detector for complex background images. IEEE Transactions on Multimedia pp. 1–1 (2018). DOI 10.1109/TMM.2018.2838320
53. Yan, C., Xie, H., Liu, S., Yin, J., Zhang, Y., Dai, Q.: Effective uyghur language text detection in complex background images for traffic prompt identification. IEEE Transactions on Intelligent Transportation Systems **19**(1), 220–229 (2018). DOI 10.1109/TITS.2017.2749977
54. Yan, C., Xie, H., Yang, D., Yin, J., Zhang, Y., Dai, Q.: Supervised hash coding with deep neural network for environment perception of intelligent vehicles. IEEE Transactions on Intelligent Transportation Systems **19**(1), 284–295 (2018). DOI 10.1109/TITS.2017.2749965
55. Yan, C., Zhang, Y., Xu, J., Dai, F., Li, L., Dai, Q., Wu, F.: A highly parallel framework for hevc coding unit partitioning tree decision on many-core processors. IEEE Signal Processing Letters **21**(5), 573–576 (2014). DOI 10.1109/LSP.2014.2310494
56. Yan, C., Zhang, Y., Xu, J., Dai, F., Zhang, J., Dai, Q., Wu, F.: Efficient parallel framework for hevc motion estimation on many-core processors. IEEE Transactions on Circuits and Systems for Video Technology **24**(12), 2077–2089 (2014). DOI 10.1109/TCSVT.2014.2335852
57. Yaseen, M.U., Anjum, A., Rana, O., Antonopoulos, N.: Deep learning hyper-parameter optimization for video analytics in clouds. IEEE Transactions on Systems, Man, and Cybernetics: Systems pp. 1–12 (2018). DOI 10.1109/TSMC.2018.2840341