Hyper-optimization tools comparison for parameter tuning applications

Camille Maurice, Jorge Francisco Madrigal Diaz, Frédéric Lerasle

Hyper-optimization tools comparison for parameter tuning applications

Camille Maurice
LAAS-CNRS
Toulouse, France
cmaurice@laas.fr

Francisco Madrigal
LAAS-CNRS
Toulouse, France
jfmadrig@laas.fr

Frederic Lerasle
LAAS-CNRS,
Université de Toulouse, CNRS, UPS
Toulouse, France
lerasle@laas.fr

Abstract

This paper evaluates and compares different hyper-parameters optimization tools that can be used in any vision applications for tuning their underlying free parameters. We focus in the problem of multiple object tracking, as it is widely studied in the literature and offers several parameters to tune. The selected tools are freely available or easy to implement. In this paper we evaluate the impact of parameter optimization tools over the tracking performances using videos from public datasets. Also, we discuss differences between the tools in term of performances, stability, documentation, etc.

1. Introduction

Multiple Object Tracking (MOT) is a popular topic due to its applicability in different areas such as human-machine interaction, activity recognition, robotics, surveillance, among others. Many proposals with different schemes and strategies have emerged to solve the challenges that the topic has to offer. At the same time, some communities provide frameworks, *e.g*. MOTChallenge [11], to evaluate, fairly, multiple people tracking algorithms. Among all the ranked strategies, it can be difficult to determine whenever it is an improvement of the tracking strategy or a better tuning of the free parameters.

The evaluation and comparison of the tracking approaches can be done through several performance evaluation metrics such as CLEAR-MOT [3]. However, the tracker's performances are tuning dependent [13]. Finding the optimal free parameters that optimize the evaluation metrics is an issue. When the number of parameters is small, tuning could be achieved manually. However, when more parameters are involved in the process, hand-tuning is not satisfactory due to time consumption and the requirement of expert knowledge. They are multiple approaches that studied the influence of free parameters (online/offline) in their proposals. Besides, we focus here on the evaluation and comparison of several well-known hyper-parameters optimization tools. Particle filter based tracking proposals commonly involve the usage of several free parameters [1]. The performances of such approaches can be characterized by one or several metrics, which could be considered as cost functions to be minimized or maximized following an optimization scheme. The literature exhibits different schemes such as stochastic optimization, model-based optimization, among others that can help to find the optimal hyper-parameters for different kind of tracking approaches.

In this paper, we analyze some well-known hyper-parameter optimization tools for tracking purpose. Parameter tuning is relevant for the majority of tracking proposals since, in most cases, parameter setting is done manually, which can lead to sub-optimal or even biased results. We present tools, some of free code and others easy to implement, for the different methods of optimization shown, which can be used for anyone. We evaluate their performances and make a quantitative comparison between the different tools considering the computational time, CLEAR-MOT based performances, stability and their ease of use. To our best knowledge, no studies have been proposed in the visual tracking community while these tools are essential for a fine tuning.

The paper is organized as follows: section 2 describes the considered application *i.e*. multi-camera MOT tracking, and the set of parameters to be optimized. Section 3 describes the optimization methods selected for the study. In section 4 these tools are described from the implementation point of view. In section 5 we show both quantitative and qualitative evaluations. Conclusions are drawn in section 6.

2. Study case

In this paper, we compare different hyper-parameters optimization tools with respect to a specific study case. This study could be done for any other applications with any cost functions, *e.g*. CLEAR-MOT [3] metrics, to evaluate their

978-1-5386-2939-0/17/$31.00

performances. Nevertheless, the relevance is more reflected when they are enough parameters to optimize. Therefore, we focus on MOT for pedestrian tracking in a multiple overlapping cameras environment. This is due to this kind of application offers several parameters to optimize as well as being widely discussed in the literature [9]. The parameter set is optimized with respect to the tracker accuracy measure defined in CLEAR-MOT [3].

Given an input video stream from k-synchronized cameras, our framework is as follows: first we perform people detection on each camera frame, illustrated as d1, d2, . . . , dl on Fig. 1. Then, those detections are projected to the ground plane thanks to the predefined camera calibration. Due to detector precision problems and camera calibration quality, different detections of a same person are not projected at the same position on the floor. For this reason we perform a fusion over the detections.Fig. 1 illustrates the following case: three people are detected by the cameras, but seven detections, represented by colored crosses, are projected to the ground plane. The color of each cross indicates the camera in which the individual has been detected, and the circles represent their fusion.

The fusion of different detections, corresponding to the same person seen by multiple cameras, is done based on the Euclidean distance on the ground plane. The appearance similarity is measured using Bhattacharya distance [4]. The fusion is done using an association strategy. The idea is to associate the detections that have the minimum distance, in terms of Euclidean distance and appearance similarity. We use the Hungarian algorithm [10] to find minimum cost matches. In order to reduce the computational cost, we only consider the detections that are close to each other. In the case of two detections being sufficiently remote from one another, above a previously defined threshold for each distance, they are not considered in the association process. When all detections from the k-cameras are merged on the ground floor, they are linked to existing tracks. We solve this association problem in a similar way than before. Once again, we use the Euclidean and an appearance-based distance thresholds to measure the score affinity between detections and trackers. Tuning of those four threshold parameters is identified to be crucial regarding the tracking performances.

Once this process is done, we use the associations as observations for the tracking module, see Fig. 1. We follow a decentralized particle filter strategy in the vein of [6]. When an observation is not associated with an existing track, we create a new tracker. The particles are propagated on the ground plane. The appearance model is learned by projecting the track position back into the k images. The likelihood of particles is calculated in the same way. However, if back projection is outside of the image place or if the target is occluded by other, by calculating the overlapping between

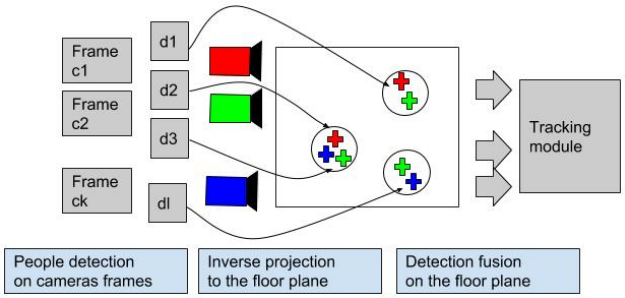


Figure 1: Workflow of our multi-cameras pre-tracking process.

projections and leaving only the closer one, then likelihood estimation is omitted in that camera. Thus, we have as parameters to tune: the number of particles, the distance considered for their propagation, and two weights in the likelihood computation. This study case provides parameters related to the data fusion of multiple cameras in addition to parameters specific to the tracker. To summarize, 8 parameters are being optimized, and the 8-components vector is called λ in the next section.

3. Hyper-optimization methods

In our analysis, we focus on complementary tuning methodologies: model free *vs*. model based.

3.1. Model-free optimization methods: example of MCMC

Model-free configuration techniques are generally classical methods that benefit from a large theoretical study since the 90s [14]. Those are relatively simple and can be applied out-of-the-box [8]. This is a decent base since our evaluation focuses on the influence of tools over performances and also their handiness.

Markov Chain Monte Carlo optimization

The Markov Chain Monte Carlo (MCMC) is a stochastic method based on sampling that can determine, by constructing a Markov Chain, the optimal values of a given function. These methods are widely used to calculate numerical approximations of complex functions because it only requires a prior distribution to estimate a new set of parameters from the previous ones. They do not require to estimate the gradient, which depends on the parameters, and can handle local minima/maxima. In literature, they are different strategies to apply the MCMC method. We estimate the optimal parameters by following the Metropolis-Hastings (MH) algorithm [5]. In a Bayesian context, the parameters follow a

unknown distribution which it can be approximated using a prior distribution. In our case, we use a normal distribution.

The Metropolis-Hastings strategy works as follow: in an iteration, a proposal λ^* is generated from the given Gaussian distribution, with mean in the current estimated parameters λ and a fixed variance. Then, a given function f is evaluated with the new parameters. If the proposal improves function value, those parameters are accepted and used in the next iteration. Otherwise, we accept the parameters with a certain probability (given by a uniform distribution). More details about the formalism can be found in [5].

3.2. Model-based optimization methods

More recent methods propose to build a model to select the optimal hyper-parameters, in a defined search space, based on previous measurements. All through evaluation iterations, new values for the parameters are defined based on this model. This class of methods is called Sequential Model Based Optimization (SMBO) [8].

SMBO algorithms have been used in particular when the evaluation of the cost function f is expensive, as in our case since we need to apply the tracking system over a sequence. We choose two SMBO tools based on their popularity and availability of open-source code. Both are described below.

Sequential Model-based Algorithm Configuration

Sequential Model-based Algorithm Configuration [8] (SMAC) is a machine learning optimization method. The main idea is to make progressively better estimations of the parameters, in contrast of MCMC which can accept parameters that decrease the performance of the cost function.

In general, SMAC builds a probabilistic model $p(f \mid \lambda)$ that captures the dependence between function f and the hyper-parameter λ. This model is constructed using a regression random forest. This allows SMAC to work with noisy functions regardless the dimensionality of the parameter. At first, SMAC construct a random forest with a set of regression trees. For each tree, a set of proposal parameters are sampled from a uniform distribution in a given finite range. Then, the tree is built by splitting the samples according to one (randomly selected) parameter of λ. This is done until a minimum number of samples per branch is reached. The standard "expected improvement (EI)" criterion is used to select promising configurations and then a local search is used to find the configuration with the highest EI values. Finally, the best configuration is compared to the previous one. If the new proposal improves the cost function, then it is accepted and the built tree is used in the next iteration. SMAC is robust to different kind of functions and parameters. This is important since we can not warranty the smoothness of the cost function.

Tree-structured Parzen Estimator

The Tree-structured Parzen Estimator [2] (TPE) is also a SMBO-based approach. The main idea is similar to the two previous method, but this proposal does not require initial values or training sets. At each iteration, TPE draws new parameter samples and decides which set to try in the next iteration. At the beginning, samples λ are drawn, uniformly over the search area, and evaluated according to function f. Then, the collected samples are divided in two groups according to their score in the cost function f: the first group contains samples that improve the current score estimation and the second the remaining. These groups are used to model the likelihood probability : $g(\lambda)$ and $l(\lambda)$ respectively. Both density estimator have a hierarchical structure and are modeled using a 1-D Parzen estimator.

The key idea is to find the set of parameters that are most likely to be in the first group. Therefore, at each iteration, new samples of $g(\lambda)$ are extracted and the one with the highest improvement is used in the next iteration. The Expected Improvement is defined by $EI(x) = \frac{l(x)}{g(x)}$. We can observe that TPE uses the distribution of the best samples instead of relying on the best estimated parameters.

4. Associated tools

SMAC [8] is a publicly available tool to optimize configuration parameters. We use the stable release SMAC (v2) in Java [1]. Authors provide a quick installation guide to setup the environment efficiently, as well as an extensive manual. This tool is widely used in different contexts with up to 412 citations at this time. In this tool we need to define two files: a parameter file and a scenario file. The parameter file defines the parameters set (named λ) to evaluate, defining their type, range and initial value. The scenario file specifies: (1) the target algorithm to run, (2) the target parameter file and (3) optimization options, *e.g.* the number of iterations.

The **TPE** [2] tool is publicly available in the Optunity library [2]. It can be installed for a wide range of environments such as: Python, MATLAB, Octave, R, Julia, and Java. Optunity has a dependency on Python and it must be installed beforehand. Both Python 2.7 and 3.x versions are compatible. The TPE tool dependents on Hyperopt, other Python library. A Python wrapper example is provided in order to optimize a non-Python functions and the maximization of the tracking performance is done through a function call. The publication related to this tool has up to 316 citations at this time.

The **MCMC** tool is an C++ implementation, done by us, of the classic Metropolis-Hastings algorithm [5]. This tool requires a variance (for the Gaussian distribution) and the

[1] http://www.cs.ubc.ca/labs/beta/Projects/SMAC/v2.10.03/quickstart.html
[2] http://optunity.readthedocs.io/

Figure 2: Examples of sequence S2L1 from PETS 2009 dataset [7]. Left-to-right: views from cameras 1,2,5,6 at frame 49.

Figure 3: Examples of sequence scenario11-1 from UvA Multi-Camera Multi-person dataset [12]. Left-to-right, top-to-bottom: views from cameras 1,2,3 at frame 0.

number of iterations. The latter has to be selected wisely. This is due to an small value will not warranty the convergence to an stationary value and high number of iterations could generate an over-fitting problem.

5. Evaluations

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

5.1. Dataset description

The tracker is evaluated on representative sequences extracted from two public datasets: PETS 2009 [7] and UvA Multi-Camera Multi-Person Benchmark [12]. PETS 2009 challenge aims to test existing or new systems for crowds video surveillance. For PETS 2009 S2L1 sequence, there are up to ten people present simultaneously recorded by eight cameras. Four cameras are used in our evaluation: two are place at approximatively 4 meters in a elevated position and relatively far from targets, and two others are placed at about 2 meters above the ground, significantly closer to the targets. An example of these cameras views is shown on Fig. 2. This scenario has a difficulty from medium to high with a semi-crowded space and 20 pedestrians. Besides, the UvA MCMPB dataset [12] contains various scenarios recorded on a platform or in the hall of a train station. In the considered sequence, there are two people recorded by three cameras at a close distance. Frames from each of the three cameras are shown on Fig. 3. Regarding the number of people presents in the scene and their trajectories, this sequence is considered as easy.

Both datasets are recorded by calibrated overlapping cameras and provide ground truth of object tracking. In total, 1296 frames are considered, 795 frames from PETS 2009 S2L1 sequence, and 501 frames from scenario11-1 out of UvA MCMPB.

5.2. Experimental setup

This section describes our evaluation protocol and criteria used in order to compare the aforementioned hyper-optimization tools.

Implementation

The tracking framework is implemented as in [6] and it is turned into a black-box function. All parameters exhibited in section 2 (λ) are set as inputs. The output is a set of optimal parameters that maximize a performance evaluation metric, defined hereafter. All the three tools require to define a search space, either with lower and upper bounds, or covariances. Additionally, SMAC and MCMC require to set initial values, we choose the middle point of the space search. The tools are able to call our tracking function with any test parameters within the specified search space.

Evaluations and comparison protocol

The cost function that we use to measure the performance is the Multiple Object Tracking Accuracy, MOTA, as defined by [3]. The metric MOTA is derived from the ratio of missed detections, false positives and mismatches over all detected objects in the sequence with respect to a ground-truth. This is commonly used as the key metric when comparing tracking methods. The MOTChallenge [11] ranked the results, by default, in decreasing order using MOTA metrics. In the top ten of the 2D MOT 2015 challenge [3], all approaches achieved similar tracking precision (MOTP), with a standard deviation of: 0.9 over 100, whereas the difference lies in the tracking accuracy, with a standard deviation of: 5.3. For these reasons we optimize the MOTA score. As we deal with multiple cameras and track people on the floor plane, we compute a variation of the MOTA metric for ground plane tracker provided by [12].

We use cross validation, where we optimize the parameters over a training set and test them over validation sets to avoid over-fitting and boost configurations who generalize well. Each sequence is divided into equal parts, one part is used for training and the rest for testing. We divide the sequences into ten equal parts and the first is used for training. In our dataset, they represent fairly well the rest of the sequences in terms of number of people and distance between each others. Due to the stochastic nature of the approaches, we launch several executions of each tool, with a different number of iterations, to evaluate its stability and speed of convergence. Given that the function cost is expensive to

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

		Train	Test	
Sequence	Tool	MOTA±SD	MOTA	MOTP
UvA 11-1	SMAC	0.83±**0.02**	0.85	120
UvA 11-1	TPE	0.86±0.06	**0.94**	120
UvA 11-1	MCMC	0.86±0.05	0.90	**115**
PETS S2L1	SMAC	0.33±**0.03**	0.36	410
PETS S2L1	TPE	0.33±0.05	0.35	406
PETS S2L1	MCMC	0.32±0.05	0.35	**405**

Table 1: MOTA and MOTP (mm) results for SMAC [8], TPE [2] and MCMC tools on the dataset.

evaluate, the speed of convergence is a very important factor. The test machine is a HP ZBook 15 with an Intel Core i7-4800 MQ CPU with 8 cores clocked at 2.8 GHz, 15 GB of RAM and Linux Ubuntu 14.04.

5.3. Results

We present the results of each hyper-optimization tool with respect to the MOTA cost function f. We compare their overall performance as well as their CPU cost, convergence speed, stability of the optimum value and available documentation. Finally given these evaluations we propose a ranking of the tools in Tab. 2.

An overview of the qualities observed in each tool is shown in the summary Tab. 2 in Mean MOTA and Global MOTA columns. In Tab. 1, we observe that MOTA maxima, on the train sequences, are very similar using any of the three tools. Each optimization tool suggests different parameter configurations. Nevertheless, in the optimized λ, they generally set a larger Euclidean distance threshold for the fusion of detections than our initial guess. This tends to address the accuracy problems in the camera calibration mentioned in section 2. There are no significant changes in the MOTP results on Tab. 1. In our case, it depends mainly on the camera calibration precision.

The tools do not perform similarly on the test sequences, we obtain better performances on the test set with parameters optimized by TPE, see Tab. 1. An overview of the qualities observed in each tool is shown in the summary Tab. 2 in Mean MOTA and Global MOTA columns. The mean MOTA and the associated standard deviation vs. the number of iterations are plotted on Fig. 4. Both SMAC and TPE achieve convergence after about 100 iterations, which is not the case for MCMC on Fig. 4c. Additional experiments highlight that MCMC needs more iterations to converge, about 800. These observations are reported on the summary Tab. 2 in Nb of iterations.

Given Tab. 1, Fig. 4a and Fig. 4b, we observe that the optimal parameters found by SMAC are more stable than those found by TPE because the standard deviation is lower. This is expressed in the column MOTA SD in Tab. 2.

Hyper-optimization tools should be easy to use and not require much inherent tuning. In this context, we can observe that SMAC and MCMC require to define the search space of the parameters as well as an initial value. TPE does not require to set initial values but lower and upper limits and the type of distribution. In this aspect, the use of TPE is easier. The CPU cost mentioned in Tab. 2 is related to the execution time of 1 run of 150 iterations. The TPE and MCMC optimization tools have the advantage that can be used like a function call. Whereas, in SMAC we need to set the folders and files to define the search space parameters, the scenarios and training or testing instances. Nevertheless, SMAC is considered as the easiest on Tab. 2 thanks to its documentation. In addition, the wide range of examples provided facilitates their adaptation in different approaches.

Unlike the SMAC tool, where outputs of important data are automatically recorded in a folder with details on each run, the TPE tool in the Hyperopt library, requires that we handle the logging values separately.

5.4. Discussion

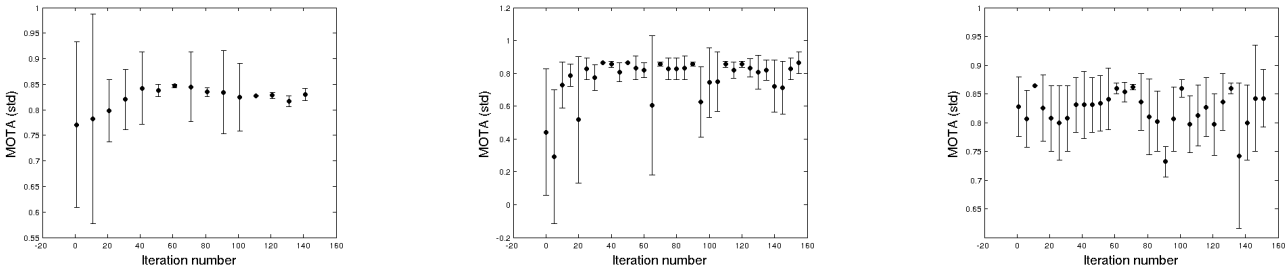
Among the tools, we rank SMAC as the best compromise with respect to our evaluation criteria detailed in Tab. 2. We consider SMAC as the easiest to use despite the number of files to configure. Its extensive documentation and example codes provided make it easy to set up. Also, after the fixed number of iterations, SMAC is the tool which obtain the smallest standard deviation in the optimization of MOTA in the train sequence. This is important for the repeatability of the results.

The TPE is ranked number two, this tool from the Hyperopt library is less accessible. It offers a wide range of possibilities but as they are less documented it makes the tool more difficult to use. TPE achieved the convergence with about the same number of iterations as SMAC. This fact is crucial because each iteration is costly. Indeed, we run at each iteration the multi-camera tracking program with parameters on the training sequence.

Finally, the MCMC tool is ranked number three. It is not described as accessible, because it is not available in an optimization library and requires some implementation work. It achieved fair results, but the main issue is regarding the number of iterations required. Even though the execution of one iteration almost only requires to run the tracking program, unlike SMAC and TPE which need more computational time, the number of iterations required makes MCMC slower.

6. Conclusions and future works

To the best of our knowledge, there are studies about the tuning influence over the tracking performances but



(a) SMAC [8] (b) TPE [2] (c) MCMC

Figure 4: Mean MOTA and standard deviation over 5 runs of 150 iterations

Tool	Training phase				Testing phase	Accessibility	Rank
	Mean MOTA	MOTA SD	Nb of iterations	CPU cost	Global MOTA		
SMAC [8]	+	+++	++	+	++	+++	1
TPE [2]	++	+	++	+	+++	++	2
MCMC	++	+	+	++	++	+	3

Table 2: Evaluation summary

no comparative studies of the optimization tools involved. In this paper, we propose a comparative study of hyper-parameters optimization tools for a multi-camera MOT application with respect to performances criteria and also accessibility. This application is an example but it can be extended to other applications.

As future work we will investigate the influence of the optimization over the metrics included in MOTA such as the number of mismatches, false positives, among others. We also would like to measure the robustness of the aforementioned tools to the amount of data in the training set.

Acknowledgments

This work has been supported by LAAS-CNRS within the project SERVAT....

References

[1] J. Berclaz, E. Turetken, F. Fleuret, and P. Fua. Multiple object tracking using k-shortest paths optimization. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 2011.

[2] J. S. Bergstra, , R. Bardenet, , Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, volume 24.

[3] K. Bernardin and R. Stiefelhagen. Evaluating multiple object tracking performance: the clear mot metrics. *EURASIP Journal on Image and Video Processing*, 2008(1):1–10, 2008.

[4] A. Bhattachayya. On a measure of divergence between two statistical population defined by their population distributions. *Bulletin Calcutta Mathematical Society*, 35(99-109):28, 1943.

[5] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[6] M. D. Breitenstein, F. Reichlin, B. Leibe, E. Koller-Meier, and L. Van Gool. Online multiperson tracking-by-detection from a single, uncalibrated camera. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 33(9):1820–1833, 2011.

[7] J. Ferryman and A. Shahrokni. Pets2009: Dataset and challenge. In *Twelfth IEEE Int. Workshop on Performance Evaluation of Tracking and Surveillance*, pages 1–6, Dec 2009.

[8] F. Hutter, , H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proc. of LION-5*, pages 507–523, 2011.

[9] K. Kim and L. Davis. Multi-camera tracking and segmentation of occluded people on ground plane using search-guided particle filtering. *Computer Vision–ECCV 2006*, pages 98–109, 2006.

[10] H. W. Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[11] L. Leal-Taixé, A. Milan, I. Reid, S. Roth, and K. Schindler. MOTChallenge 2015: Towards a benchmark for multi-target tracking. Apr. 2015.

[12] M. C. Liem and D. M. Gavrila. A comparative study on multi-person tracking using overlapping cameras. In *International Conference on Computer Vision Systems*, pages 203–212. Springer, 2013.

[13] E. Maggio, M. Taj, and A. Cavallaro. Efficient multitarget visual tracking using random finite sets. *IEEE Transactions on Circuits and Systems for Video Technology*, 18(8):1016–1027, Aug 2008.

[14] S. Minton, M. D. Johnston, A. B. Philips, and P. Laird. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58(1-3):161–205, 1992.