# A Time-Efficient Optimisation Framework for Parameters of Optical Flow Methods

Michael Stoll(✉), Sebastian Volz, Daniel Maurer, and Andrés Bruhn

Institute for Visualization and Interactive Systems,
University of Stuttgart, Stuttgart, Germany
{stoll,volz,maurer,bruhn}@vis.uni-stuttgart.de

**Abstract.** Due to the increase of optical flow benchmark data, concerning both amount and resolution, learning parameters from training sequences with ground truth has become significantly more challenging in recent years. Moreover, optical flow methods are much more complex than a few years ago resulting in a larger amount of model parameters and a noticeably increased runtime. As a consequence, even optimising a small set of suitable parameters may take hours or even days which makes hand tuning infeasible. Hence, time-efficient strategies for automatic parameter optimisation become more and more important. In this context, our work addresses three important aspects. First, we provide an overview of different optimisation strategies and juxtapose them in the context of different optical flow methods and different evaluation benchmarks. Second, we focus on choosing a suitable subset of the training data to speed up the computation while still obtaining meaningful results. Finally, we also consider different strategies for distributing the evaluation on hardware infrastructures which allows to further reduce the run time. Experiments show that the proposed methodology allows to obtain good results while keeping the overall effort reasonably low.

**Keywords:** Performance evaluation · Parameter optimisation · Distributed optimisation · Adaptive scheduling · Optical flow

## 1 Introduction

Among the variety of computer vision tasks, optical flow estimation sticks out in some important aspects. First of all, it is applied to sequences of images, in contrast to single images. Secondly, it covers a 2D search space (in contrast to e.g. 1D stereo problems). Additionally, there is already a variety of benchmarks which allow for a comparison of different methods.

Evaluating an optical flow method is usually done by comparing its performance quantitatively to other methods. Recent benchmarks provide training datasets containing image sequences *with* ground truth data and testing datasets containing only image sequences. The estimation results are evaluated against the given ground truth by computing error measures. Depending on the benchmark, different error measures such as the *average angular error* (AAE), the *average endpoint error* (AEE) or the *average bad pixel* (BP3, using a 3 pixel threshold) are chosen for comparison. Hence, the typical optimisation task comes down to choosing a benchmark and the corresponding error measure and finding the parameters that minimise the overall error value.

In the early days of quantitative evaluation, there was a single artificial image sequence with ground truth, the *Yosemite* sequence [2] having a resolution of $316 \times 252$ pixels. In 2007, the Middlebury benchmark was presented providing 8 training and 8 testing datasets with resolutions from $316 \times 252$ to $640 \times 480$ pixels [1]. Most of these image sequences have been created synthetically, not providing realworld challenges such as significant illumination changes, considerable out-of-plane motion or disturbances in data such as lens flares, under- and oversaturations or noise. These types of challenges are provided by the KITTI Vision benchmark which was proposed in 2012 [10]. It contains 194 training and 195 testing datasets covering resolutions from $1226 \times 370$ to $1238 \times 374$ (training data) and $1242 \times 375$ pixels (testing data), respectively. In the same year, 2012, also the MPI Sintel Flow dataset was proposed, which contains artificial data on the one hand, but provides tough challenges by creating specular reflections, motion blur, defocus blur and atmospheric effects [6]. There are 1064 training and 564 testing image datasets at a resolution of $1024 \times 436$ pixels. The Middlebury benchmark provides two important rankings, one w.r.t. the $AAE$ and another w.r.t. the $AEE$. The MPI Sintel benchmark makes use of the $AEE$ while the KITTI benchmark ranks methods according to their average BP3 error.

There is an increasing amount of data – provided by publicly available benchmarks – and there are optical flow methods with increasing complexity (e.g. [23–26]) – developed by researchers. But to the best of our knowledge there is no easy-to-use publicly available framework that connects data and methods by finding suitable parameters in a reasonable amount of time. An open-source variant that provides a basis of common parameter optimisation algorithms which can be extended through contributions by experienced researchers could be valuable for the whole community.

**Contributions.** We present such a framework which does not require any specific properties of the underlying objective function (such as e.g. its derivatives) and by design can be generalised to other computer vision tasks comprising a scalar error measure. In this context our contributions are fourfold: First, we briefly introduce different classes of derivative-free algorithms for parameter optimisation which are juxtaposed later in the evaluation. Second, we reduce the computational burden by applying a selection strategy in order to determine suitable subsets of the datasets for evaluation. Third, we argue how all the individual sampling steps can be distributed efficiently among different computers within the typical hardware infrastructure of a research institute. Finally, we provide an open-source framework for both, single-machine- as well as distributed parameter optimisation with easy adaptability and extendibility regarding both the interfaces to the optical flow methods and the implementation of derivative-free algorithms for parameter optimisation. The framework is publicly available at http://go.visus.uni-stuttgart.de/cvis-optimizer/.

**Related Work.** While there are a few methods that estimate their parameters jointly with the optical flow [12,17], most of the current algorithms rely on some kind of a-priori parameter selection based on training data; see e.g. [5,9,15,22,26]. Surprisingly, most of these approaches do not comment on the underlying optimisation strategy. The few exceptions can essentially be divided into two classes: On the one hand, there are *derivative-based* approaches that rely on gradient descent or Newton-like techniques to estimate the optimal parameters as minimiser of a cost functional that relates the obtained results to ground truth data [15,22,26]. However, since the underlying optical flow models are rather complex – in contrast to parameter estimation for image restoration [14,21] – the functional gradient is typically not computed analytically but approximated stochastically [15,22,26]. On the other hand, there are *derivative-free* techniques that are solely based on multiple evaluation runs for different parameter settings. Although some of those methods implicitly look for the downhill direction, they can also be applied in those cases where no analytic or stochastic gradient is available. Typical representatives for this class of method are the multidimensional sampling [8] as well as the Downhill Simplex method [9,16]. Moreover, also evolutionary algorithms such as the Covariance Matrix Adaptation Evolution Strategy [11,20] or nature inspired algorithms [18] belong to this class.

The only related work regarding the comparison of parameter optimisation methods for optical flow estimation is given by [18]. However, the current paper improves upon this work by addressing significantly more aspects. In particular, it considers three different classes of derivative free methods (fixed sampling, geometric sampling, stochastic sampling), it evaluates the performance for two optical flow algorithms, it considers all major benchmarks, it investigates the usage of suitable subsets of image sequences and it elaborates on the distribution of evaluation runs on homogeneous and heterogeneous hardware infrastructures.

There is also a lot of work in the context of auto-tuning of image processing and computer vision algorithms like e.g. [7,13,19] which are helpful to split up an image processing algorithm into parallelisable pieces and schedule them on a multicore system. Our goal, however, is not to split up an algorithm and schedule its parts in the context of parallelization. We rather want to distribute multiple, independent executions of an algorithm (i.e. optical flow estimations) among the available hardware, measured by the number of CPU cores in one or more computing systems. Regarding distributed optimisation, [3] provides domain decomposition schemes for general optimisation problems, discusses such examples in statistics and machine learning and gives hints on the implementation of general distributed optimisation. It does, however, not focus on the scheduling of the distributed tasks. [4] gives an overview of different scheduling heuristics to map independent tasks on heterogeneous distributed computing systems.

**Organisation.** Section 2 reviews different derivative-free strategies for parameter optimisation. Moreover, it discusses how to speed up the computation by selecting a suitable subset of the training sequences. Section 3 then sketches the distributed computation, while Sect. 4 presents an evaluation of the different strategies. Section 5 concludes with a summary.

## 2 The Optimisation Process

For an optimisation problem with objective function $f(\mathbf{x})$, the goal is to find the parameters $\mathbf{x}_{\min} \in \mathbb{R}^P$ that minimise $f$ where $P$ is the number of parameters. In case of optical flow, this objective function depends on the optical flow method, its parameters $\mathbf{x}$, the training data with ground truth and the respective error measure $d$, e.g. AEE, AAE or BP(3). Please note that the objective function is often nonconvex w.r.t. the parameters such that one typically relaxes the task to finding a suitable local minimum. In the following, the whole optimisation process consists of many *optimisation runs*, each evaluating $f$ on the whole dataset with a different parameter set $\mathbf{x}$. Each optimisation run, in turn, consists of several *evaluation tasks*, each evaluating $d$ on a single element $i$ of the dataset yielding the deviation $d_i(\mathbf{x})$. The value of the objective function is then given by $f(\mathbf{x}) = \operatorname{avg}_i d_i(\mathbf{x})$, which we call the *error value* of the optimisation run.

### 2.1 Continuous Parameters

In order to find optimal parameters that can be sampled from a continuous space, different derivative-free parameter optimisation methods are known that solely rely on the values of the objective function $f(\mathbf{x})$ at specific sampling points $\mathbf{x}_i$. These methods can be further classified according to the heuristics they use. Examples for methods without a heuristic are equidistant (cascadic) sampling or logarithmic cascadic sampling [8]. Given a sampling interval, they compute sample points and determine the best result. In the cascadic case, they repeat this procedure with a smaller interval around the best sampling point so far. An example for a method with a geometric heuristic is the so-called Downhill Simplex method, a.k.a. Nelder-Mead method [16], which constructs an initial simplex of sampling points in $\mathbb{R}^P$ and moves this simplex in a downhill manner towards a minimum by different geometrically inspired steps which are called *reflection*, *expansion*, *contraction* and *reduction*. The Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [11] is an example for a method with a stochastic heuristic. It belongs to the class of evolutionary algorithms, i.e. is based on populations of parameter sets and consists of two iterated main steps: *mutation* and *recombination*.

We will use these methods exemplarily in the evaluation. However, any other derivative-free method is also implementable in our system.

### 2.2 Discrete Parameters

Besides the continuous parameters like e.g. the weights of different assumptions in a model, there can also be discrete parameters like booleans or enumerations which can only take a finite, a priori known set of values. These discrete parameters are separated from the continuous ones, all possible combinations are computed and for each set of discrete parameters the remaining continuous parameters are optimised using one of the strategies from above.

### 2.3 Reducing the Amount of Evaluations

Instead of using all datasets from provided image and ground truth data, it may be sufficient to use a subset of them for the parameter optimisation. The question then arises which datasets to keep.

As the total error value is the average error of each dataset w.r.t. the chosen error measure, a promising approach is to take a look at those datasets that have the highest variance among the error values for an initial set of optimisation runs, thus being sensitive to parameter variations and influencing the variations of the average error the most. Hence, we evaluate an initial set of optimisation runs (i.e. the first level of a cascade, the first simplex or the first population) on all given datasets, determine the range $r_i$ of error values for each dataset $i$ and - given a percentage value $q$ as a parameter - we select those $q$ percent of datasets with the highest $r_i$ and re-compute the average error values using only the selected datasets. After the parameter optimisation strategy has terminated, we compute the errors for the ignored datasets using the estimated parameters and re-compute the average error using all datasets in order to obtain an average error for the complete given dataset.

## 3 Distributed Computation of Evaluation Tasks

After each step of the chosen parameter optimisation algorithm there is a set of evaluation tasks to be performed. Each of these tasks can be computed independently from the others and thus may be distributed to a separate thread. These threads can either run on the same computer or in a distributed fashion on different computers. Taking into account only one computer means that each thread of a thread pool can simply fetch a task, evaluate it and fetch the next one as long as there are evaluation tasks left.

Using more computers, however, makes things more complicated. On the one hand, one needs a central instance managing the optimisation process and clients that take care of the evaluations as well as a communication layer between them. On the other hand, different computers may show different performance. Hence, the question arises how to distribute the tasks such that they are computed as fast as possible. In case each client is allowed to evaluate tasks as long as there are any left - named as *Opportunistic Load Balancing (OLB)* [4] - it may happen that the slowest client fetches the last task while faster clients stay unemployed. For an optimisation run, this leads to a longer computation time than necessary.

### 3.1 Minimal Completion Time Algorithm

The *Minimal Completion Time (MCT)* algorithm [4] is a heuristic that uses an estimation of the run time for a task in order to assign it to the client which is assumed to have the smallest time until it completes all computations. As the highest completion time (CT) among all clients coincides with the completion of the current bunch of evaluation tasks, achieving a minimal CT avoids unnecessary waiting for the tasks to be finished. To this end, we need to keep track of the performance of all clients, measured by the average run time $t_{\text{avg}}$.

The algorithm is a greedy algorithm. It iterates through all available tasks and assigns each task to the client whose estimated CT - including the task it is currently computing - would be the smallest one. The CT of the client is the maximum CT among all its threads. For the computation of the CT, the current task is virtually assigned to the thread with the smallest CT as we expect this thread to first acquire the next task. This way, a small amount of (remaining) tasks is assigned to fast clients instead of slow clients.

**Timing Statistics.** The central quantity is the average evaluation time $t_{\text{avg}}$ of each client. Upon the first acquisition of a task by the client, it is initialised with a small value bigger than zero in order to get a reasonable first distribution of the tasks. Otherwise, tasks would probably be assigned to only the first client or the one with $t_{\text{avg}} = 0$. On the first submission of an evaluation result, $t_{\text{avg}}$ is set to the evaluation time $t_{\text{curr}}$ of the submitted task. In subsequent submissions, we update $t_{\text{avg}}$ with a weighted averaging:

$$t_{\text{avg}} = \left(1 - \frac{1}{q}\right) \cdot t_{\text{avg}} + \frac{1}{q} \cdot t_{\text{curr}} \tag{1}$$

We set $q = 6$ in order to make the estimation of $t_{\text{avg}}$ robust against outliers.

If the run times are not expected to be rather equal for all datasets (e.g. due to a different resolution of the data), a so-called run time factor can be assigned to those datasets that lead to a significantly different evaluation effort. The evaluation time $t_{\text{curr}}$ is then normalised before the computation of $t_{\text{avg}}$.

The performance of a client may undergo variations, e.g. due to workload, thermal issues etc. In particular, it might become significantly slower than estimated. In order to be robust against this, we also keep track of the time since the last task acquisition or submission, respectively, which we call $t_{\text{silence}}$. In the worst case, a client evaluates only tasks with the maximal run time factor and may thus be expected to be busy for $t_{\text{busy}} = \max(R) \cdot t_{\text{avg}}$ (if $R$ denotes the set of all run time factors). In case, it is not interacting for a longer time (i.e. $t_{\text{silence}} > t_{\text{busy}}$) we assume a decrease in performance, thus replacing $t_{\text{avg}}$ by an effective average evaluation time $t_{\text{eff}} = \frac{t_{\text{silence}}}{\max(R)}$.

If there are multiple optimisations at the same time, they can involve different methods and/or different data leading to different average run times. Computing only a single, joint average run time $t_{\text{avg}}$ could lead to strong over- or underestimations of the run times of different evaluation tasks, thus leading to inappropriate task distributions to the available clients. Hence, the average run time $t_{\text{avg}}$ is measured for each optimisation separately.

## 4 Evaluation

In order to perform an evaluation of our optimiser, we applied it to the publicly available methods of Brox and Malik (LDOF) [5] and Weinzaepfel et al. (DF)

[26]. The datasets are the eight sequences of the Middlebury training dataset (MB) [1], a subset of 20 sequences of the KITTI training dataset (seq. 0–19) [10] and a subset of 69 sequences of the Sintel training dataset [6] (final version, using the middle image pair as well as the pairs five frames before and five frames after the middle of each scene). The restriction to these subsets of KITTI and Sintel, which we expect to be representative, had to be performed in order to make the evaluation, which incorporates a large amount of optimisation processes, feasible. Otherwise, each single optimisation process on KITTI or Sintel would have already taken several days of run time. Please note, that we only evaluate on these subsets and they thus serve as starting point for the proposed selection of even smaller subsets in Sect. 4.2. Furthermore, we restrict the optimisation to those parameters that define weights for different terms in the functional (3 parameters for LDOF, 4 parameters for DF). The initial search interval for all parameters was $[0, 200]$ (or they were initialised with 100 for CMA-ES which constitutes the centre of the interval $[0, 200]$) except for the matching term weights which take the discrete values 200, 300 or 400. Both equidistant cascadic sampling and logarithmic cascadic sampling use five samples per parameter and four cascade levels.

## 4.1 Comparison of the Parameter Optimisation Methods

In our first experiment, we compare the different combinations of optical flow methods, optimisation methods and benchmarks in terms of both the achieved error and the required number of optimisation runs. Table 1 shows both values where the number of optimisation runs can be found in brackets. As one can see, CMA-ES performs best for DeepFlow (DF) but it gets trapped in a local minimum for LDOF. In contrast, the logarithmic cascadic approach avoids this minimum and provides the best result for this optical flow method. Regarding the number of optimisation runs, all parameter optimisation strategies stay in the same order of magnitude for LDOF (3 parameters) while things change for DeepFlow (4 parameters). Here, the equidistant and logarithmic sampling strategies need one order of magnitude more runs than Downhill Simplex (DS). In all cases CMA-ES requires a medium amount of runs. Regarding both error and number of optimisation runs DS is the favourable method: It does not achieve the absolute best results but stays within a range of less than 1% from the top results while only requiring at most 25% of the number of runs compared to CMA-ES. Moreover, its results are superior to the baseline that has been computed using the specified parameters in the paper. The only slight exception is DeepFlow on the Middlebury dataset but the baseline parameters ignore the - in this setting - inappropriate matching term which we however keep active.

For the DeepFlow algorithm on the KITTI training subset, Fig. 1 displays how the individual algorithms converge to their final error value while the number of runs is growing. Please note, that we only show a single instance of each algorithm, i.e. for the best discrete choice of the matching weight (being either 200, 300 or 400). We can observe that logarithmic cascadic sampling passes a

**Table 1.** Error values and number of optimisation runs for different methods and benchmarks.

| | Equid. (C) | Log. (C) | DS | CMA-ES | Baseline |
|---|---|---|---|---|---|
| LDOF (MB, *AAE*) | 4.349 (240) | **3.976** (300) | 3.993 (106) | 4.318 (763) | 4.100 |
| DF (MB, *AEE*) | 0.252 (1344) | 0.251 (1500) | 0.252 (138) | 0.251 (552) | **0.250** |
| DF (KITTI, *BP*) | 8.340 (1341) | 8.335 (1500) | 8.296 (201) | **8.292** (880) | 9.303 |
| DF (Sintel, *AEE*) | 8.732 (1341) | 8.743 (1500) | 8.739 (144) | **8.720** (848) | 11.026 |

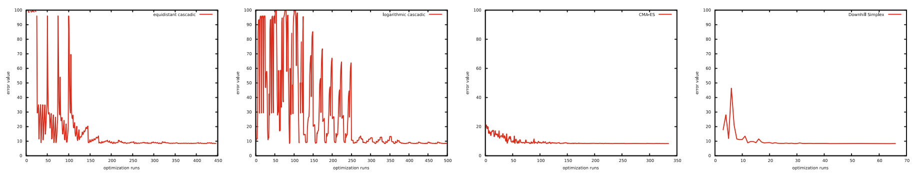

**Fig. 1.** Convergence of the parameter optimisation strategies. **From left to right:** equidistant cascadic sampling, logarithmic cascadic sampling, CMA-ES, DS.

large variety of error values indicating that it tests many different orders of magnitude of ratios of the different weighting parameters. In contrast, equidistant cascadic sampling only covers a small range of error values despite some single peaks. CMA-ES already starts at a low error level and successively oszillates to the minimum while DS already converges after a few optimisation runs.

## 4.2 Influence of Using Subsets

Our second experiment evaluates the effect of reducing the amount of evaluation tasks per optimisation run by selecting a suitable subset of the dataset for evaluation. To this end, we automatically choose this subset according to the highest variance (cf. Sect. 2.3) and consider both, the resulting errors and the corresponding number of evaluations. Table 2 shows that using around 10% of all datasets leads to only a moderate degradation. In this context, the loss in quality is considerably higher if either the baseline dataset is already small (only 20 KITTI sequences) or the method does not perform many optimisation runs like DS, i.e. if the total number of evaluations has already been small. In the other cases, the loss in quality is not that significant. When using around 20% of the baseline dataset for evaluation, there is hardly any degradation - in particular for CMA-ES or Sintel. It is worth noting that the amount of optimisation runs can change depending on the subset due to different adaptions of the respective algorithm to the intermediate errors. In some cases, using less sequences may even lead to more evaluations (cf. Table 2, KITTI (DS), last two rows).

**Table 2.** Using the sequences with the **highest** variances of the error values for optimisation. The entries have the format *error (#runs, #evaluations)*.

| | KITTI (CMA-ES) | KITTI (DS) | Sintel (CMA-ES) | Sintel (DS) |
|---|---|---|---|---|
| 100% | 8.292 (880, 17600) | 8.296 (201, 4020) | 8.720 (848, 58512) | 8.739 (144, 9936) |
| 50% | 8.311 (960, 9690) | 8.429 (198, 2030) | 8.722 (864, 29691) | 8.737 (159, 5581) |
| 20% | 8.322 (920, 3824) | 8.920 (138, 632) | 8.760 (832, 11320) | 8.737 (129, 1957) |
| 10% | 8.734 (936, 2034) | 9.031 (189, 652) | 8.777 (840, 5607) | 8.931 (165, 1305) |

## 4.3 Task Distribution Strategies

In our final experiment, we compare the average optimisation times for DeepFlow with the Downhill Simplex method for different levels of parallelism. To this end, we compute the times of an optimisation process on a typical office computer (4 cores), a workstation (24 cores) and a combination of a workstation and some office PCs (9 additional cores). The latter is evaluated using both, opportunistic load balancing (OLB) and load balancing using the minimal completion time algorithm (MCT). We investigate the two cases: First, having less evaluation tasks than CPU cores which are evaluated in parallel (using the 20 sequences of the KITTI subset) and second, having more evaluation tasks than CPU cores (using the 69 sequences of the Sintel subset). The DS algorithm is - besides its fast convergence - an interesting algorithm for investigation because it is mostly sequential in terms of optimisation runs. Hence, a bad distribution of evaluation tasks leads to unnecessary waiting times at almost each optimisation run which sum up in the total optimisation time.

**Table 3.** Optimisation times (given in hours) for DS on DeepFlow w.r.t. different distribution strategies in a heterogeneous infrastructure. 1 and 2 indicate different sets of additional computers (each giving 9 additional cores, i.e. 33 cores in total).

| | 4 cores | 12 cores | 24 cores | OLB$^1$ | MCT$^1$ | OLB$^2$ | MCT$^2$ |
|---|---|---|---|---|---|---|---|
| KITTI (20 seq.) | 08:43 | 03:54 | 02:55 | 02:56 | 02:43 | 02:33 | **02:32** |
| Sintel (69 seq.) | - | - | 06:38 | 05:04 | 05:04 | 05:01 | **04:49** |

From Table 3, we can see that using more cores on a single machine is of course beneficial, but the gain in performance might not always be as high as expected. One reason is the sequential nature of DS regarding the optimisation runs. But there is another important reason: the behaviour of some modern many-core CPUs. When using only a small subset of all cores, these cores run at a higher frequency compared to the case when all cores are busy. In the latter case, an individual task is evaluated more slowly. Hence, even when the single machine with 24 cores is not working to capacity, using more cores on different machines and distributing the workload among those is beneficial despite of the overhead due to communication and synchronisation.

When comparing the different distribution strategies, the benefit of the MCT heuristic becomes evident. Either it is better than OLB or at least they show similar performance. The latter can be explained by the already mentioned behaviour of some modern CPUs. For them, the estimation of a task's expected run time - which is the basis for the MCT heuristic - is nontrivial. If these CPUs are assigned many tasks, they become slow. As a consequence, they are assigned only a few tasks afterwards, making them faster again. Using an estimation that respects this behaviour is future work and may improve the outcome of the MCT heuristic. Nevertheless, MCT can already reduce the run time by up to 8%.

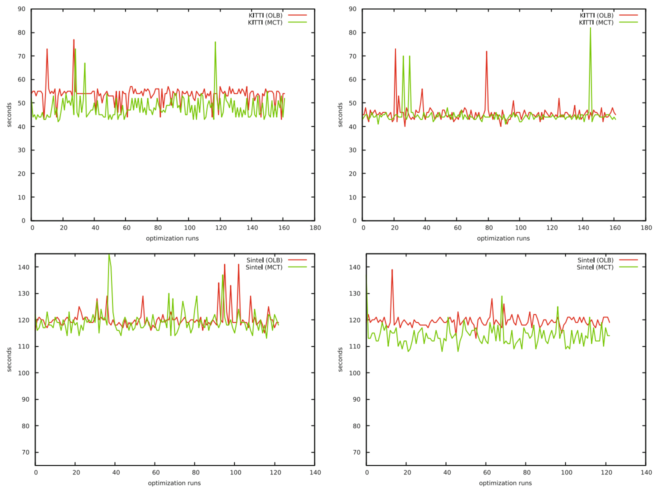

**Fig. 2.** Run times for subsequent optimisation runs. Red stands for the OLB heuristic, green for the MCT heuristic. **From left to right:** The sets 1 and 2 of additional computers according to Table 3. **From top to bottom:** KITTI, Sintel. (Color figure online)

In this context, we also compared the run times of the individual, subsequent optimisation runs by the example of DS. To this end, we considered only those steps of DS that led to an evaluation of only a single optimisation run (i.e. reflection, expansion and contraction steps) in order to have comparable time values. The graphs in Fig. 2 confirm the tendency of the total optimisation times in Table 3. These graphs show that the findings that hold for the total run time are also valid for the individual run times of the optimisation runs. When comparing the graphs for OLB (red) and MCT (green), one can see that the MCT heuristic leads to smaller run times for configuration 1 of additional computers on the KITTI dataset and for configuration 2 on the Sintel dataset. In the other two cases, both graphs indicate similar run times. The sparse peaks in all graphs are related to peripheral effects which do not depend on the heuristic.

## 5 Conclusion

In this work, we investigated three important aspects of time-efficient parameter optimisation for optical flow. To this end, we first compared different optimisation methods w.r.t. quality and workload. In this context, the Downhill Simplex method showed up to be a favourable compromise as it achieves good results using only a small amount of optimisation runs. Furthermore, we proposed a variance based strategy to reduce the number of training datasets (thus reducing the number of evaluation tasks per optimisation run) while retaining a good level of quality - in particular for large baseline datasets. Finally, we adopted a promising heuristic for distributing the evaluation tasks among different computers which allowed us to further reduce the overall optimisation time. To encourage the use of our entire optimisation framework, we provide a publicly available implementation as open source.

**Acknowledgements.** We thank the German Research Foundation (DFG) for financial support within project B04 of SFB/Transregio 161.

## References

1. Baker, S., Roth, S., Scharstein, D., Black, M.J., Lewis, J.P., Szeliski, R.: A database and evaluation methodology for optical flow. In: Proceedings of IEEE International Conference on Computer Vision (ICCV). IEEE Computer Society Press (2007)
2. Barron, J.L., Fleet, D.J., Beauchemin, S.S.: Performance of optical flow techniques. Int. J. Comput. Vis. **12**(1), 43–77 (1994)
3. Boyd, S., Parikh, N., Chu, E., Peleato, B., Eckstein, J.: Distributed optimization and statistical learning via the alternating direction method of multipliers. Found. Trends Mach. Learn. **3**(1), 1–122 (2011)
4. Braun, T.D., Siegel, H.J., Beck, N., Bölöni, L.L., Maheswaran, M., Reuther, A.I., Robertson, J.P., Theys, M.D., Yao, B., Hensgen, D., Freund, R.F.: A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. J. Parallel Distrib. Comput. **61**(6), 810–837 (2001)
5. Brox, T., Malik, J.: Large displacement optical flow: descriptor matching in variational motion estimation. IEEE Trans. Pattern Anal. Mach. Intell. **33**(3), 500–513 (2011)
6. Butler, D.J., Wulff, J., Stanley, G.B., Black, M.J.: A naturalistic open source movie for optical flow evaluation. In: Fitzgibbon, A., Lazebnik, S., Perona, P., Sato, Y., Schmid, C. (eds.) ECCV 2012. LNCS, vol. 7577, pp. 611–625. Springer, Heidelberg (2012). doi:10.1007/978-3-642-33783-3_44
7. Datta, K., Murphy, M., Volkov, V., Williams, S., Carter, J., Oliker, L., Patterson, D., Shalf, J., Yelick, K.: Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures. In: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, pp. 4:1–4:12. IEEE Press (2008)
8. Demetz, O.: Feature Invariance versus Change Estimation in Variational Motion Estimation. Ph.D. Thesis, Faculty of Mathematics and Computer Science, Saarland University (2015)
9. Drayer, B., Brox,T.: Combinatorial regularization of descriptor matching for optical flow estimation. In: British Machine Vision Conference (BMVC). BMVA Press (2015)

10. Geiger, A., Lenz, P., Urtasun, R.: Are we ready for autonomous driving? The KITTI vision benchmark suite. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 3354–3361. IEEE Computer Society Press (2012)
11. Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution strategies. Evol. Comput. **9**(2), 159–195 (2001)
12. Krajsek, K., Mester, R.: A maximum likelihood estimator for choosing the regularization parameters in global optical flow methods. In: Proceedings of IEEE International Conference on Image Processing (ICIP), pp. 1081–1084. IEEE Computer Society (2006)
13. Kulkarni, T., Kohli, P., Tenenbaum, J.B., Mansinghka, V.: Picture: a probabilistic programming language for scene perception. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 4390–4399. IEEE Computer Society Press (2015)
14. Kunisch, K., Pock, T.: A bilevel optimization approach for parameter learning in variational models. SIAM J. Imaging Sci. **6**(2), 938–983 (2013)
15. Li, Y., Huttenlocher, D.P.: Learning for optical flow using stochastic optimization. In: Forsyth, D., Torr, P., Zisserman, A. (eds.) ECCV 2008. LNCS, vol. 5303, pp. 379–391. Springer, Heidelberg (2008). doi:10.1007/978-3-540-88688-4_28
16. Nelder, J.A., Mead, R.: A simplex method for function minimization. Comput. J. **7**(4), 308–313 (1965)
17. Memin, E., Heas, P., Herzet, C.: Bayesian inference of models and hyperparameters for robust optic-flow estimation. IEEE Trans. Image Process. **21**(4), 1437–1451 (2012)
18. Perreira, D.R., Delpiano, J., Papa, J.P.: On the optical flow model selection through metaheuristics. EURASIP J. Image Video Process. **2015**, 11 (2015)
19. Ragan-Kelley, J., Barnes, C., Adams, A., Paris, S., Durand, F., Amarasinghe, S.: Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. In: Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 519–530. ACM (2013)
20. Salmen, J., Caup, L., Igel, C.: Real-time estimation of optical flow based on optimized haar wavelet features. In: Takahashi, R.H.C., Deb, K., Wanner, E.F., Greco, S. (eds.) EMO 2011. LNCS, vol. 6576, pp. 448–461. Springer, Heidelberg (2011). doi:10.1007/978-3-642-19893-9_31
21. Samuel, K.G.G., Tappen, M.F.: Learning optimized map estimates in continuously-valued MRF models. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 477–484. IEEE Computer Society Press (2009)
22. Sun, D., Roth, S., Lewis, J.P., Black, M.J.: Learning optical flow. In: Forsyth, D., Torr, P., Zisserman, A. (eds.) ECCV 2008. LNCS, vol. 5304, pp. 83–97. Springer, Heidelberg (2008). doi:10.1007/978-3-540-88690-7_7
23. Sun, D., Sudderth, E.B., Black, M.J.: Layered segmentation and optical flow estimation over time. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1768–1775. IEEE Computer Society Press (2012)
24. Sun, D., Roth, S., Black, M.J.: A quantitative analysis of current practices in optical flow estimation and the principles behind them. Int. J. Comput. Vis. **106**(2), 115–137 (2013)

25. Volz, S., Bruhn, A., Valgaerts, L., Zimmer, H.: Modeling temporal coherence for optical flow. In: Proceedings of IEEE International Conference on Computer Vision (ICCV), pp. 1116–1123. IEEE Computer Society Press (2011)
26. Weinzaepfel, P., Revaud, J., Harchaoui, Z., Schmid, C.: DeepFlow: large displacement optical flow with deep matching. In: Proceedings of IEEE International Conference on Computer Vision (ICCV), pp. 1385–1392. IEEE Computer Society Press (2013)