2023 IEEE International Conference on Robotics and Automation (ICRA 2023)
May 29 - June 2, 2023. London, UK

Discovering Multiple Algorithm Configurations

Leonid Keselman[1] and Martial Hebert[1]

***Abstract*— Many practitioners in robotics regularly depend on classic, hand-designed algorithms. Often the performance of these algorithms is tuned across a dataset of annotated examples which represent typical deployment conditions. Automatic tuning of these settings is traditionally known as algorithm configuration. In this work, we extend algorithm configuration to automatically discover multiple modes in the tuning dataset. Unlike prior work, these configuration modes represent multiple dataset instances and are detected automatically during the course of optimization. We propose three methods for mode discovery: a post hoc method, a multi-stage method, and an online algorithm using a multi-armed bandit. Our results characterize these methods on synthetic test functions and in multiple robotics application domains: stereoscopic depth estimation, differentiable rendering, motion planning, and visual odometry. We show the clear benefits of detecting multiple modes in algorithm configuration space.**

I. Introduction

Autonomous integrated systems often depend on a multitude of algorithms interacting with each other and their external environment. Despite the recent popularity of deep, end-to-end trained models [1], robotic systems often depend on hand-designed algorithms in several parts of the processing stack. They include motion planning [2], algorithms involved in sensing [3] and simultaneous location and mapping [4]. As systems become more sophisticated, they often accumulate more methods and with them, more parameters that need to be set and configured by the system designers.

Often the developers of these methods can discover viable configurations by hand but leave many settings open to configuration by eventual users. Intuitively, these can include settings can control smoothing, performance and run-time. Ideal settings in noise-free environments can vary dramatically from those required in noisy settings. Likewise, different configurations may exist for optimal online and offline performance. Tuning such settings to work well in a deployment environment remains a challenge for many autonomous systems. Without proper tuning, components that are expected to be reliable may unexpectedly fail.

While it is possible to tune settings by hand, it is also possible to use automated methods to find potential configurations. In classic computer science literature, this was done to optimize runtime performance [5] and was known as the algorithm selection (or configuration) problem. Researchers have shown how automated tuning can quickly and robustly improve the performance of even common tools such as compilers [6]. In an era with many benchmarks [7], [8], [9] and challenge competitions [10], algorithm tuning is performed on a validation set made to approximate the performance of the final test set, and ensure the best possible performance for proposed techniques. To demonstrate consistency and generalizability, researchers often report performance under a single configuration.

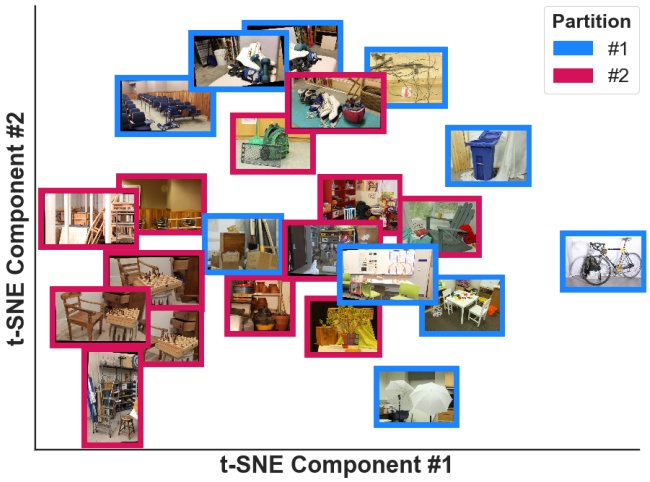


Fig. 1: **Partitioned Configurations.** Instead of finding a single algorithm configuration for an entire dataset, we partition the dataset during the process of optimization and find a different configuration for each partition.

Considering multiple configurations can greatly expand the applicability of a particular method [11]. In the case of multi-objective optimization, a Pareto set exists, where progress on any one objective regresses performance on another objective. As such, providing multiple configurations is often done by hardware vendors [3], and selecting between multiple configurations in robotics is also well studied [12], [13], [14], along with selecting from ensembles of solvers [15] and motion-planners [16].

In this work, we propose to discover multiple viable configurations while an algorithm configuration is being automatically tuned over a dataset by some black-box optimizer [17], [18]. By noticing correlations between data in response to new configurations, we detect multiple algorithm modes. Working in algorithm configuration space enables generalization across several problem domains, as they do not depend on domain specific features. This allows our method, with fixed settings, to show benefits in multiple application areas (see Section IV). We show how multiple modes can be found online (Section III-E) and how they can be used to guard against outliers (Section IV-B).

[1]Leonid Keselman and Martial Hebert are with the Robotics Institute, School of Computer Science, Carnegie Mellon University, Pittsburgh PA, 15232 USA {lkeselma,hebert}@cs.cmu.edu

Further info at https://leonidk.github.io/modecfg

II. Related Work

There is a long history of work in algorithm configuration and related areas. Originally studied for performance optimization [5], algorithm configuration has been well studied in the SAT solver community [19]. These extensions include large evaluations [20], taxonomies of methods [21] and methods which adapt the algorithm configuration over a series of time-steps [22], [23], [24], assuming the algorithms used have different temporal properties.

Since robotics applications and datasets are reasonably expensive operations (compared to purely synthetic tasks), our work is related to work on extremely few function evaluations, typically on the order of 100 [25]. Typically this is studied in the Machine Learning community as hyperparameter search, finding configurations for ideal neural network configuration and training [26], [18].

Our work is most closely related to portfolio-based algorithm configuration literature [27], [28], [29]. These methods often design a different configuration for each instance of the problem in their dataset (e.g. each data example) [30], [31], [32]. Similar to our work, they cluster methods into groups. However, their clusters are derived from domain-specific feature spaces, while we use the response of the instances to new configurations. Our use of domain-specific features is limited to test time deployment, using supervised training to target our obtained algorithm configuration clusters.

In the Machine Learning community, the problem of coreset discovery [33] is related to our approach. Coreset discovery finds representative examples from a dataset to focus training time on a subset of the data. Most related, methods exist for online discovery of such sets [34]. Of note, our contribution is orthogonal to coreset research, as our method could benefit from coreset methods, which would serve to give us a smaller subset of data to evaluate at each iteration. Specifically, coreset methods for clustering [35] could enable more function evaluation steps in a fixed budget of time and give better resulting minima.

In computer vision, some recent work has explored estimating algorithm configurations for classic algorithms on a local level, operating on patches within an image [36].

III. Method

Our approach consists of evaluating algorithm configurations from a black box optimizer (Section III-B) across a dataset of examples for the given algorithm (Algorithm 1). Building upon this baseline, we propose three methods of partitioning the data during optimization: Post hoc (Algorithm 2), Staged (Algorithm 3), and Online (Algorithm 4).

For our experiments, we always use two partitions, even when there are more known modes (Section IV-A). This avoids exploring the area of instance-specific algorithm configuration and minimizes our risk of overfitting to the dataset and overstating our performance. We typically report results between the initialization (the known defaults for an algorithm) and the oracle. Our oracle is defined as awarding the best known configuration for each individual datum across all optimization runs.

A. Partitioning

The optimal partition for a given number of partitions K, with M algorithm configurations over N datapoints can be formulated via 0-1 Integer Linear Programming where $c_{i,j}$ corresponds to the quality of datum i with configuration j.

$$\begin{aligned} \min \quad & c_{i,j}\, x_{i,j} \\ \text{subject to} \quad & \\ & x_{i,j} \in \{0,1\} \\ & \sum_{j=1}^{M} x_{i,j} = 1 \\ & \sum_{j=1}^{M} \mathbb{1}\left[(\sum_{i=1}^{N} x_{i,j}) > 0 \right] \leq K \end{aligned}$$

One can also exhaustively evaluate all $\binom{M}{K}$ partitions. Our experiments do exhaustive evaluation for $K = 2$ (as with all results in this paper) and use the optimization formulation with larger numbers of partitions. We solve the optimization problem with a recent solver [37] and implement the indicator variables using the BigM modeling trick.

As an alternative, we evaluate using a clustering method such as k-Means [38] on a normalized matrix $\tilde{X}$, where each row has zero mean and unit variance. Cluster centers are in the space of the history of evaluated configurations and each row is a datum's response to the history of evaluated configurations. Clustering approaches treat the algorithm configuration history as a feature and group results which behave similarly, but may not be optimally partitioned.

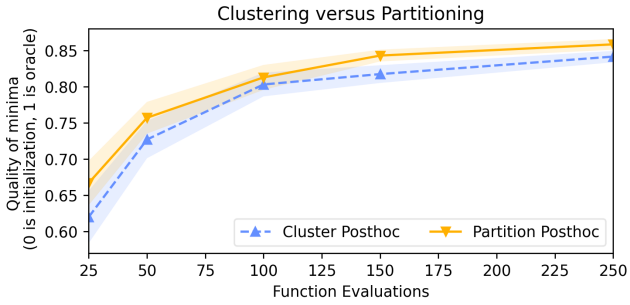


Fig. 2: **Partitioning vs K-Means Clustering** on the stereoscopic depth experiments described in Section IV-B.

B. Black Box Optimizer

Our method is generic to the choice of black box optimizer, also known as gradient-free optimization. For example, one could use random search, which is known to be a strong baseline in higher dimensional optimization [39]. On the other extreme, if one has expensive function evaluations, one could fit a surrogate function to the data and optimize its expected minima, as is done in Bayesian Optimization [40]. Effective black box optimizers in practice often combine a plethora of optimizers [6], [41].

We use CMA-ES, an evolutionary method with a multivariate Gaussian model [17]. CMA-ES is known for algorithm configuration search in robotics [42] and is widely used in other algorithm configuration comparisons [22]. CMA-ES is convenient in only requiring an initial configuration and a σ in parameter space. When tuning existing algorithms, reasonable prior configurations are often available. Approaches which focus on modeling a bounded volume [40], [41]) can be wasteful. All of our parameter search is for non-negative parameters, so we transform our search space with $\log(x)$ to perform unconstrained optimization, making our search operate on order-of-magnitude scale for all parameters.

C. Post hoc Partitioning

The post hoc method is simple and straightforward: perform black box on the dataset as a whole, noting each datum's response to each configuration. Afterwards, partition the data following Section III-A. This approach allows clearest evaluation against the non-partitioned CMA-ES baseline, which it outperforms in all of our experiments. Algorithm 2 outlines the *Post hoc* method in detail. As a method with no change in exploration, it can be be run on existing single mode optimization or simply using coherently evaluated random configurations [39], as in Section IV-F.

D. Staged Partitioning

In staged partitioning, we spend half of the function evaluations exploring the space to find adequate minimia, and we spend half of the function evaluations exploiting the discovered partitions in isolation. While the scale of a particular problem may suggest different balance of exploration and exploitation stages, we use a ratio of $\frac{1}{2}$ for our experiments. Algorithm 3 performs partitioning in the middle of optimization and tunes the results for each partition. This enables more explicit exploitation of the partitions, at the expense of less exploration time to find good partitions.

E. Online Partitioning

To balance exploration and exploitation in an online fashion, one could use a multi-armed bandit. Algorithm 4 dynamically assigns data points to partitions during the course of optimization. Since CMA-ES only evaluates relative order, we can readily switch data assigned to each partition during the course of evaluation. For the online method, we setup a multi-armed bandit (MAB) for each datum. Since our function evaluations are unbounded, we use classic Thompson Sampling [43], [44] with a Gaussian distribution. We perform one CMA-ES step to sample the space, and use that to initialize the distributions for each arm of the bandits identically. Each iteration, we sample a partition assignment for each bandit. That datum then evaluates the configuration given by that optimizer and records its result for that arm. The optimizers record the mean cost of the data assigned to them that iteration.

This approach allows us to simultaneously perform multiple optimizations and partition assignments on the fly.

Algorithm 1 Optimize algorithm configuration over a dataset

Input: x_0 Initial Configuration
Input: $f_{1\ldots N}(x)$ dataset queries for algorithm
Input: M Maximum number of function evaluations
procedure OPTIMIZE($x_0, f_{1\ldots N}(x), M$)
  for $i \leftarrow 1$ to N **do**
    $x_i \leftarrow$ OPTIMIZER CANDIDATE()
    for $j \leftarrow 1$ to N **do** ▷ Evaluate all data
      $X_{i,j} \leftarrow f_j(x_i)$
    end for
    $g_i \leftarrow$ MEAN(X_i)
    OPTIMIZER TELL(g_i) ▷ Report average
  end for
  $Y_i \leftarrow$ MEAN($X_{i,j}$) ▷ Per configuration scores
  $x \leftarrow x_{\text{argmin}(Y_i)}$ ▷ Best configuration
end procedure

Algorithm 2 Finding modes with post hoc partitioning

Input: K Number of partitions
Output: $x_{1..K}$ Per partition configurations
Output: $c_{1..N}$ Per datum partition assignments
procedure POSTHOC($x_0, f_{1\ldots N}(x), M, K$)
  OPTIMIZE($x_0, f_{1\ldots N}(x), M$)
  $c_j \leftarrow$ PARTITION(X^T, K) ▷ Get partitions for data
  $Y_k \leftarrow$ MEAN($X_{i,(c_j=k)}$) ▷ Per partition scores
  $x_k \leftarrow x_{\text{argmin}(Y_k)}$ ▷ Configuration for partitions
end procedure

Algorithm 3 Finding modes with staged partitioning

procedure STAGED($x_0, f_{1\ldots N}, M, K$)
  POST HOC($x_0, f_{1\ldots N}(x), \frac{M}{2}, K$)
  for $k \leftarrow 1$ to K **do** ▷ Separate optimization
    OPTIMIZE($x_0, f_{c_j=k}(x), \frac{M}{2}$)
  end for
end procedure

Algorithm 4 Finding modes with online partitioning

procedure ONLINE($x_0, f_{1\ldots N}, M, K$)
  $B_k \leftarrow$ BANDIT(N) ▷ K arms for each datum
  $OPT_k \leftarrow$ OPTIMIZER() ▷ K optimizers
  for $i \leftarrow 1$ to M **do**
    for $j \leftarrow 1$ to N **do**
      $b_j \leftarrow$ BANDIT PULL(B_j) ▷ Sample bandit
    end for
    for $m \leftarrow 1$ to K **do** ▷ Separate Evaluation
      $x_{i,k} \leftarrow OPT_k$CANDIDATE()
      $y_{i,k} \leftarrow$ MEAN($f_{b_j=k}(x_{i,k})$)
      OPT_kTELL($y_{i,k}$)
    end for
  end for
  $c_j \leftarrow$ BEST ARM(B_j)
  $y_k \leftarrow$ BEST CONFIG(OPT_k)
end procedure

IV. Experimental Results

We evaluate our approach on several application domains. We start with a synthetic function whose structure and modes are known and is quick to evaluate. This enables us to characterize our different methods of finding partitions. We then proceed to show successful benefits to robotics methods like stereoscopic depth generation [3], differentiable rendering [45], motion planning [46], and visual odometry [4].

A. Synthetic Function

A synthetic function allows us to characterize our methods across arbitrary many dimensions and modes. Our synthetic function has K modes, each the sum of N hard-to-optimize functions, leading to a simulation of KN data points. We use four hard-to-optimize functons: Ackley, Griewank, Rastrigin, Zakharov (for details of these functions and their visualizations see [47]), and rescale them to have a minima of value zero, a random rotation, and to have a maximum value of around one near the minima. This paradigm and these functions can be generated in arbitrary many dimensions, allowing us to understand how these partitioning methods scale as algorithm hyper-parameters scale from two to forty dimensions.

For the synthetic function optimization, the staged method works best across most dimensions and number of function evaluations. Close behind, especially with fewer evaluations, is the post hoc method. In contrast, our online bandit method is typically only slightly better than the single mode baseline. Of note is that all methods begin to perform better with hundreds of function evaluations, suggesting that the improved performance of the partitioning may come from improved efficiency in low numbers of evaluations, and not the multi-modal nature of the synthetic function.

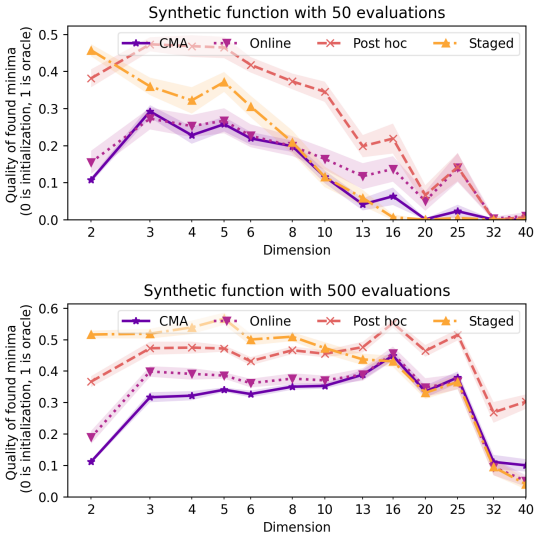


Fig. 3: **Synthetic Function Partitioning** (Section IV-A). Graphs shows the quality of the best found minima for all methods, between the initial configuration and an oracle. Shaded regions indicate standard error of the mean.

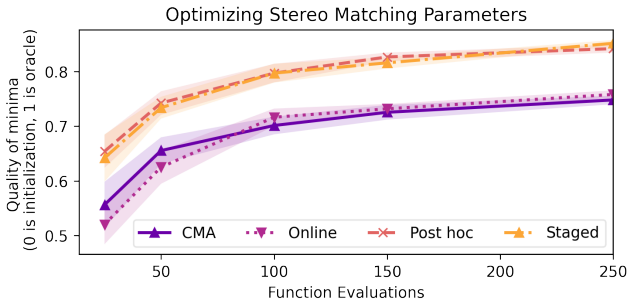


Fig. 4: **Dense Stereo Matching Partitioning** quality on the training set. The posthoc and staged methods perform well while the online method is indistinguishable from CMA-ES.

B. Dense Stereo Matching

Robotics applications often use stereoscopic depth sensors. Here we optimize the performance a classic Dense Stereo Matching method, namely Semi-Global Block Matching (SGBM) [48] as implemented by OpenCV [49]. We obtain 47 image pairs by combining the Middlebury 2014 and 2021 Stereo datasets [9]. We split the data into 23 training examples and 24 test examples, shown in Section IV-B. The algorithm settings control the regularization of the SGBM algorithm, the post-processing filters used to cleanup the data, and the block size used for initial matching. Results of the four methods on the training set are shown in Fig. 4.

In deploying the discovered configurations to new data, we show the efficacy of a simple supervised classifier.The classifier used is k-nearest neighbors with a $k = 1$, returning the partition index to be used. We use a pre-trained neural network's top level feature space as the feature space. Specifically we use SqueezeNet 1.1 [50] pre-trained on ImageNet in PyTorch [51] and its 512 dimensional space for images.

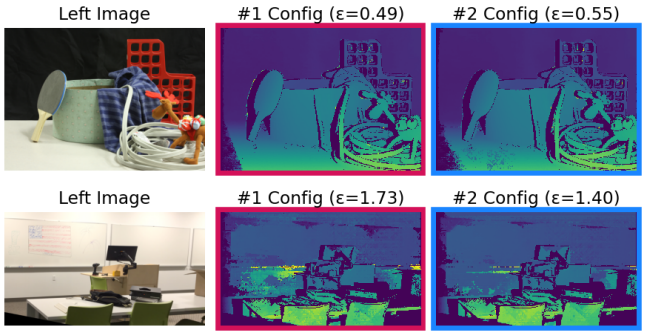
The test set performance is improved with partitioning, as shown in Fig. 5c with quantitative estimates and Fig. 5b with two qualitative examples from the test set.

We find that the optimal hyperparameter configuration typically focuses on regularization and filtering. The first configuration usually has less regularization, but a more aggressive filter to discard bad matches, while the second configuration has more regularization and less aggressive filters to discard bad data.

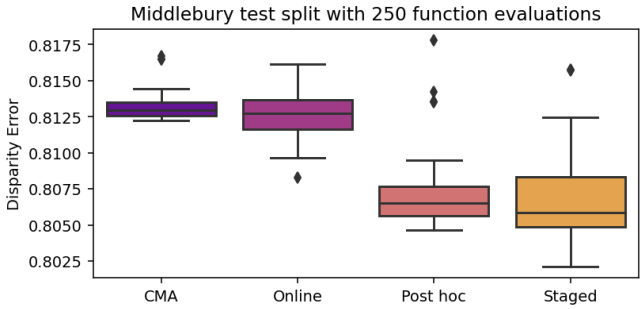
C. Differentiable Rendering

We optimize the parameters of a recent differentiable renderer [45]. Our dataset includes 20 sequences from the KITTI odometry dataset [7] and 20 synthetic shapes. The KITTI sequences use k-Means to build a quick model of 10 consecutive LIDAR frames, from the center of the scene looking out. In contrast, the synthetic sequences all have the object densely in front camera. The differentiable render has four hyperparameters, two controlling the sharpness of the silhouettes, one controlling surface smoothness and one controlling how opaque objects are. We optimize all four for depth and silhouette accuracy, similar to the original paper.

(a) **Middlebury Stereo Data** with training data shown as partitioned by our staged method after 250 evaluations. The test set is shown with the predicted classification colors for each image.



(b) **Qualitative examples of test set results** with our classifier correctly assigning the better configuration to each example. Disparity error shown in parenthesis for each configuration and example.



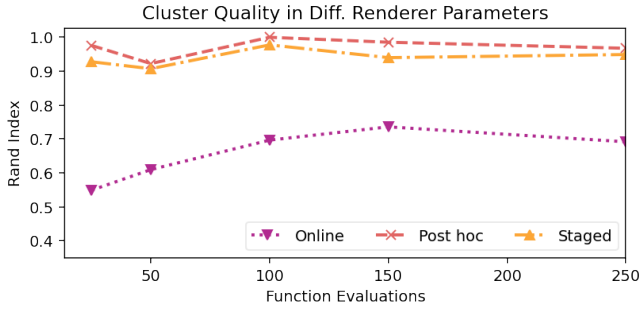
(c) **Test set performance** based on the configurations preferred by our supervised classifier described in Section IV-B.

Fig. 5: **Dense Stereo Matching Test Set Performance**.

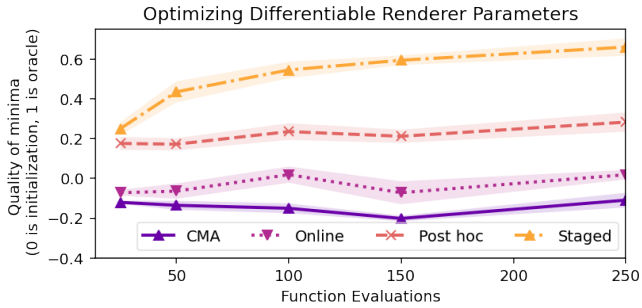
In rendering, the optimizer is unable to find a better single mode configuration than the initialization. However, all proposed methods show a statistically significant improvement over the baseline, with the staged method performing the best. In addition, we report the ability of the methods to properly partition the disparate datasets in Fig. 6a.

D. Motion Planning

We evaluate a popular motion planning method, Informed RRT* [52] in the Sampling-Based Motion Planner Testing Environment [53]. We setup three start-goal pairs for three testing environments. We use the geometric mean of runtime (as estimated by the number of expanded notes) and performance (as estimated by the quality of the first found solution). This balance of runtime and quality is essential to obtaining an interested configuration. Results are shown in Fig. 7, with only the post hoc method outperforming the baseline. Other methods perform poorly, and we suspect the problem is insufficient samples and lack of exploration in the online and staged methods; especially as RRT*-based planners are stochastic, making evaluations noisy.



(a) **Partitioning Accuracy** in splitting the KITTI data from synthetic data in Section IV-C. All methods show some success.



(b) **Differentiable Renderer Partitioning** hyper-parameter optimization for depth and silhouette fidelity. Section IV-C. In this case, reasonable defaults restrict the progress of the baseline while the exploitation-focused staged optimization performs best.

Fig. 6: **Differentiable Renderer Experiments**

We find that the optimal partition finds different parameters focused on the goal sampling frequency (0.2 and 0.3; single mode 0.26) and the rewiring radius (1500 vs 7500; single mode 6000). Of our three start/goal pairs in each of three different environments, optimal partitioning typically grouped one environment together.

We also performed some experiments with RRdT* [54], which we report briefly report. Often, one partition would focus on a configuration that frequently spawned new trees, while the other focused on expanding existing trees.

As it is unclear how to parameterize motion planning goals and environments for supervised classification, we were unable to do experiments on a hold-out test set.

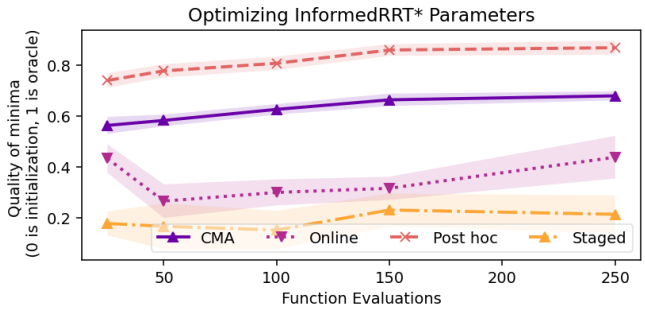


Fig. 7: **Motion Planner Partitioning** on a set of environments and planar planning tasks. Section IV-D for details.

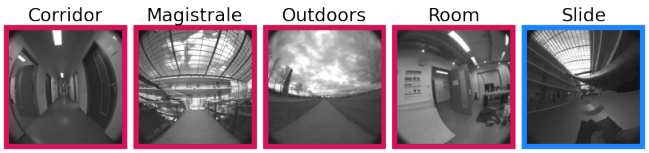


Fig. 8: **Partitioned TUM-VI Visual Odometry Dataset**

TABLE I: **Visual Odometry Partitioning** for the DM-VIO on the TUM-VI dataset. See Section IV-E for details.

Method	**Func. Evals.**	**Quality (0 is init., 1 is oracle)**
CMA	50	0.34 ±0.14
CMA	100	0.40 ±0.19
CMA	150	0.49 ±0.07
Post hoc	50	0.84 ±0.02
Post hoc	100	0.84 ±0.08
Post hoc	150	0.90 ±0.01

E. Visual Odometry

We perform experiments on a subset of the TUM VI Visual-Inertial Dataset [55] using DM-VIO [56]. DM-VIO has many parameters but we focus our optimization on just five (number of points, number of immature points, min frames, max frames, max optimization steps). TUM VI has 5 environments, and we select the third sequence from each environment as our dataset.

We prioritize a geometric mean of runtime (frame time) and quality (best-aligned absolute pose error [57]), while penalizing trajectories which do not complete successfully.Results are shown in Table I and Fig. 8. Reliably, the algorithm partitions a separate configuration for the *slide3* sequence, which includes fast motion through a closed pipe. We find that this configuration typically has a large set of immature points (400 vs 200; single mode of 200) and a larger minframe size (6 vs 2; single mode of 2) to handle the high-velocity, highly occluded part of the sequence.

As our VO partitions depend on properties of the sequence, we were unable to construct a reasonable test set based on the first frame. Instead, our multiple configurations may be used in on-the-fly configuration selection [13],

F. Commercial Depth Sensor

Lastly, we demonstrate our partitioning method on a Intel RealSense D435 [3] and its 35 parameters for estimating depth. We generate a set of 500 randomly configurations. We evaluate all configurations on 10 scenes, for which we have collected their pseudo ground truths using a moving laser pattern [3]. We partition the configurations using the post hoc method. Results for four scenes are shown in Fig. 9.

The optimal $K = 2$ partition included the best single mode configuration as well, allowing us to show it and the alternative configuration. The single mode configuration produced small holes, but dense results outdoors. While the alternative configuration produced smoother, denser walls in indoor environments in exchange for more artifacts outdoors.

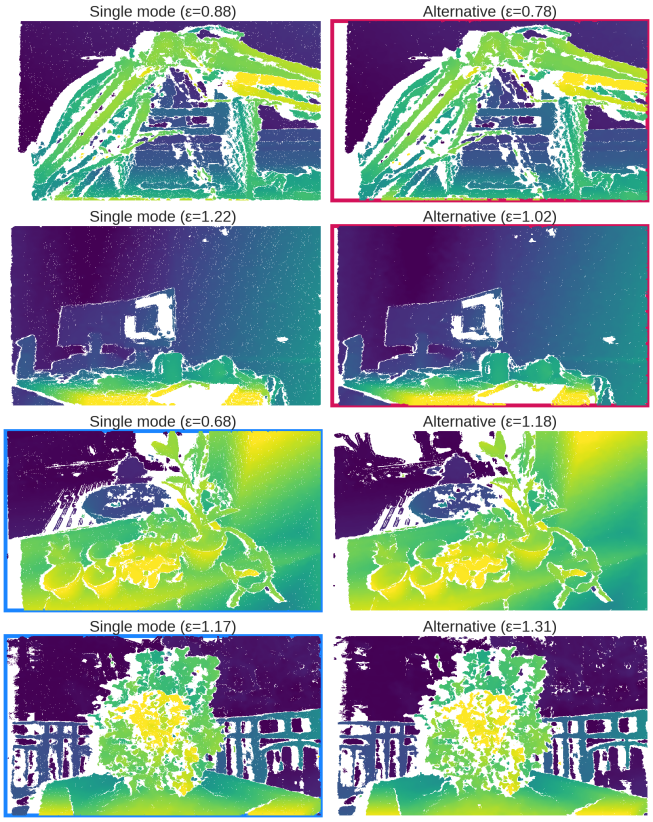


Fig. 9: **Intel RealSense D435 Partitioning** with 500 randomly generated configurations on 10 scenes. Two are shown, with both configurations (and its error). Section IV-F.

V. Discussion

Many algorithms in robotics operate in environments with multiple modes. These natural partitions are easy to understand, and can be discovered naturally by analyzing how different datums respond to different algorithm configurations. The modes were found because they affected algorithm response, not because they happened to be grouped together in some domain-specific feature space.

All the proposed methods for partitioning show some efficacy. Across the board, the post hoc method works well. This is likely due our extremely small number of evaluations for algorithm configuration [25], leading to more benefits for exploration. The online method typically performs poorly in this setting and it is possible that more sophisticated bandit algorithms [58] could perform better.

Our experiments focused on two partitions for all methods. Even when problems had more modes by construction, two partitions were able to clearly improve performance.

VI. Conclusion

Automatically finding modes during the course of algorithm configuration is a viable way to improve algorithm performance in several different areas of robotics. More study is needed to understand what typical algorithm modes exist and how such modal configurations might be used long-term deployed autonomous systems.

References

[1] OpenAI, M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba, "Learning dexterous in-hand manipulation," 2019.

[2] D. Coleman, I. Sucan, S. Chitta, and N. Correll, "Reducing the barrier to entry of complex robotic software: a moveit! case study," 2014. [Online]. Available: https://arxiv.org/abs/1404.3785

[3] L. Keselman, J. I. Woodfill, A. Grunnet-Jepsen, and A. Bhowmik, "Intel realsense stereoscopic depth cameras," 2017. [Online]. Available: https://arxiv.org/abs/1705.05548

[4] J. Engel, V. Koltun, and D. Cremers, "Direct sparse odometry," 2016. [Online]. Available: https://arxiv.org/abs/1607.02565

[5] J. R. Rice, "The algorithm selection problem," *Adv. Comput.*, vol. 15, pp. 65–118, 1976.

[6] J. Ansel, S. Kamil, K. Veeramachaneni, J. Ragan-Kelley, J. Bosboom, U.-M. O'Reilly, and S. Amarasinghe, "Opentuner: An extensible framework for program autotuning," in *International Conference on Parallel Architectures and Compilation Techniques*, Edmonton, Canada, Aug 2014. [Online]. Available: http://groups.csail.mit.edu/commit/papers/2014/ansel-pact14-opentuner.pdf

[7] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[8] M. Menze and A. Geiger, "Object scene flow for autonomous vehicles," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[9] D. Scharstein, H. Hirschmüller, Y. Kitajima, G. Krathwohl, N. Nešić, X. Wang, and P. Westling, "High-resolution stereo datasets with subpixel-accurate ground truth," in *Pattern Recognition*, X. Jiang, J. Hornegger, and R. Koch, Eds. Cham: Springer International Publishing, 2014, pp. 31–42.

[10] E. Krotkov, D. Hackett, L. Jackel, M. Perschbacher, J. Pippine, J. Strauss, G. Pratt, and C. Orlowski, "The darpa robotics challenge finals: Results and perspectives," in *The DARPA Robotics Challenge Finals: Humanoid Robots To The Rescue*. Springer, 2018, pp. 1–26.

[11] L. Makatura, M. Guo, A. Schulz, J. Solomon, and W. Matusik, "Pareto gamuts: Exploring optimal designs across varying contexts," *ACM Transactions on Graphics (SIGGRAPH)*, vol. 40, no. 4, pp. 1–17, 8 2021. [Online]. Available: https://doi.org/10.1145/3450626.3459750

[12] H. Hu and G. Kantor, "Introspective evaluation of perception performance for parameter tuning without ground truth," in *Proceedings of Robotics: Science and Systems*, Cambridge, Massachusetts, July 2017.

[13] ——, "Efficient automatic perception system parameter tuning on site without expert supervision," in *Proceedings of the 1st Annual Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, S. Levine, V. Vanhoucke, and K. Goldberg, Eds., vol. 78. PMLR, 13–15 Nov 2017, pp. 57–66. [Online]. Available: https://proceedings.mlr.press/v78/hu17a.html

[14] ——, "Compensating for context by learning local models of perception performance," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 4629–4634.

[15] S. Choudhury, S. Arora, and S. Scherer, "The planner ensemble: Motion planning by executing diverse algorithms," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 2389–2395.

[16] A. Tallavajhula, S. Choudhury, S. Scherer, and A. Kelly, "List prediction applied to motion planning," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 213–220.

[17] N. Hansen, "The cma evolution strategy: A tutorial," 2016.

[18] I. Loshchilov and F. Hutter, "Cma-es for hyperparameter optimization of deep neural networks," 2016.

[19] C. Ansótegui, Y. Malitsky, H. Samulowitz, M. Sellmann, and K. Tierney, "Model-based genetic algorithms for algorithm configuration," in *Proceedings of the 24th International Conference on Artificial Intelligence*, ser. IJCAI'15. AAAI Press, 2015, p. 733–739.

[20] N. Hansen, A. Auger, R. Ros, O. Mersmann, T. Tušar, and D. Brockhoff, "Coco: a platform for comparing continuous optimizers in a black-box setting," *Optimization Methods and Software*, vol. 36, no. 1, pp. 114–144, 2021. [Online]. Available: https://doi.org/10.1080/10556788.2020.1808977

[21] E. Schede, J. Brandt, A. Tornede, M. Wever, V. Bengs, E. Hüllermeier, and K. Tierney, "A survey of methods for automated algorithm configuration," 2022. [Online]. Available: https://arxiv.org/abs/2202.01651

[22] T. Eimer, A. Biedenkapp, M. Reimer, S. Adriaensen, F. Hutter, and M. Lindauer, "Dacbench: A benchmark library for dynamic algorithm configuration," 2021. [Online]. Available: https://arxiv.org/abs/2105.08541

[23] F. Ye, C. Doerr, and T. Bäck, "Leveraging benchmarking data for informed one-shot dynamic algorithm selection," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, ser. GECCO '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 245–246. [Online]. Available: https://doi.org/10.1145/3449726.3459578

[24] S. Adriaensen, A. Biedenkapp, G. Shala, N. Awad, T. Eimer, M. Lindauer, and F. Hutter, "Automated dynamic algorithm configuration," 2022. [Online]. Available: https://arxiv.org/abs/2205.13881

[25] C. Ansotegui, M. Sellmann, T. Shah, and K. Tierney, "Learning how to optimize black-box functions with extreme limits on the number of function evaluations," 2021. [Online]. Available: https://arxiv.org/abs/2103.10321

[26] L. Kotthoff, C. Thornton, H. H. Hoos, F. Hutter, and K. Leyton-Brown, *Auto-WEKA: Automatic Model Selection and Hyperparameter Optimization in WEKA*. Cham: Springer International Publishing, 2019, pp. 81–95. [Online]. Available: https://doi.org/10.1007/978-3-030-05318-5_4

[27] K. Leyton-Brown, E. Nudelman, G. Andrew, J. McFadden, and Y. Shoham, "A portfolio approach to algorithm select," in *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, ser. IJCAI'03. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003, p. 1542–1543.

[28] A. Guerri and M. Milano, "Learning techniques for automatic algorithm portfolio selection," in *Proceedings of the 16th European Conference on Artificial Intelligence*, ser. ECAI'04. NLD: IOS Press, 2004, p. 475–479.

[29] K. Leyton-Brown, E. Nudelman, and Y. Shoham, "Empirical hardness models: Methodology and a case study on combinatorial auctions," *J. ACM*, vol. 56, no. 4, jul 2009. [Online]. Available: https://doi.org/10.1145/1538902.1538906

[30] Y. Malitsky and M. Sellmann, "Stochastic offline programming," in *2009 21st IEEE International Conference on Tools with Artificial Intelligence*, 2009, pp. 784–791.

[31] L. Xu, H. H. Hoos, and K. Leyton-Brown, "Hydra: Automatically configuring algorithms for portfolio-based selection," in *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, ser. AAAI'10. AAAI Press, 2010, p. 210–216.

[32] S. Kadioglu, Y. Malitsky, M. Sellmann, and K. Tierney, "Isac – instance-specific algorithm configuration," in *Proceedings of the 2010 Conference on ECAI 2010: 19th European Conference on Artificial Intelligence*. NLD: IOS Press, 2010, p. 751–756.

[33] B. Mirzasoleiman, J. Bilmes, and J. Leskovec, "Coresets for data-efficient training of machine learning models," 2019. [Online]. Available: https://arxiv.org/abs/1906.01827

[34] J. Yoon, D. Madaan, E. Yang, and S. J. Hwang, "Online coreset selection for rehearsal-based continual learning," in *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=f9D-5WNG4Nv

[35] O. Bachem, M. Lucic, and A. Krause, "Scalable k-means clustering via lightweight coresets," 2017. [Online]. Available: https://arxiv.org/abs/1702.08248

[36] B. Wronski, "Procedural kernel networks," 2021. [Online]. Available: https://arxiv.org/abs/2112.09318

[37] Q. Huangfu and J. A. J. Hall, "Parallelizing the dual revised simplex method," *Mathematical Programming Computation*, vol. 10, no. 1, pp. 119–142, Mar 2018. [Online]. Available: https://doi.org/10.1007/s12532-017-0130-5

[38] D. Sculley, "Web-scale k-means clustering," in *Proceedings of the 19th International Conference on World Wide Web*, ser. WWW '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 1177–1178. [Online]. Available: https://doi.org/10.1145/1772690.1772862

[39] D. Golovin, B. Solnik, S. Moitra, G. Kochanski, J. E. Karro, and D. Sculley, Eds., *Google Vizier: A Service for Black-Box Optimization*, 2017. [Online]. Available: http://www.kdd.org/kdd2017/papers/view/google-vizier-a-service-for-black-box-optimization

[40] P. I. Frazier, "A tutorial on bayesian optimization," 2018. [Online]. Available: https://arxiv.org/abs/1807.02811

[41] J. Rapin and O. Teytaud, "Nevergrad - A gradient-free optimization platform," https://GitHub.com/FacebookResearch/Nevergrad, 2018.
[42] J. Zhang, P. Fiers, K. A. Witte, R. W. Jackson, K. L. Poggensee, C. G. Atkeson, and S. H. Collins, "Human-in-the-loop optimization of exoskeleton assistance during walking," *Science*, vol. 356, no. 6344, pp. 1280–1284, 2017. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.aal5054
[43] W. R. Thompson, "On the likelihood that one unknown probability exceeds another in view of the evidence of two samples," *Biometrika*, vol. 25, no. 3/4, pp. 285–294, 1933. [Online]. Available: http://www.jstor.org/stable/2332286
[44] D. Russo, B. Van Roy, A. Kazerouni, I. Osband, and Z. Wen, "A tutorial on thompson sampling," 2017. [Online]. Available: https://arxiv.org/abs/1707.02038
[45] L. Keselman and M. Hebert, "Approximate differentiable rendering with algebraic surfaces," in *European Conference on Computer Vision (ECCV)*, 2022.
[46] S. M. Lavalle, "Rapidly-exploring random trees: A new tool for path planning," Tech. Rep., 1998.
[47] X. Lv, Y. Wang, J. Deng, G. Zhang, and L. Zhang, "Improved particle swarm optimization algorithm based on last-eliminated principle and enhanced information sharing," *Computational Intelligence and Neuroscience*, vol. 2018, p. 5025672, Dec 2018. [Online]. Available: https://doi.org/10.1155/2018/5025672
[48] H. Hirschmuller, "Stereo processing by semiglobal matching and mutual information," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 2, pp. 328–341, 2008.
[49] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.
[50] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size," 2016. [Online]. Available: https://arxiv.org/abs/1602.07360
[51] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035.
[52] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 2997–3004.
[53] T. Lai, "sbp-env: A python package for sampling-based motion planner and samplers," *Journal of Open Source Software*, vol. 6, no. 66, p. 3782, 2021. [Online]. Available: https://doi.org/10.21105/joss.03782
[54] T. Lai, F. Ramos, and G. Francis, "Balancing global exploration and local-connectivity exploitation with rapidly-exploring random disjointed-trees," in *Proceedings of The International Conference on Robotics and Automation (ICRA)*. IEEE, 2019.
[55] D. Schubert, T. Goll, N. Demmel, V. Usenko, J. Stueckler, and D. Cremers, "The tum vi benchmark for evaluating visual-inertial odometry," in *International Conference on Intelligent Robots and Systems (IROS)*, October 2018.
[56] L. von Stumberg and D. Cremers, "DM-VIO: Delayed marginalization visual-inertial odometry," *International Conference on Robotics and Automation (ICRA)*, vol. 7, no. 2, pp. 1408–1415, 2022.
[57] M. Grupp, "evo: Python package for the evaluation of odometry and slam." https://github.com/MichaelGrupp/evo, 2017.
[58] P. Ménard and A. Garivier, "A minimax and asymptotically optimal algorithm for stochastic bandits," *Algorithmic Learning Theory*, vol. 76, 2017. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01475078