# Re-Evaluating Chemical Synthesis Planning Algorithms

**Anonymous Author(s)**
Affiliation
Address
email

## Abstract

Computer-Aided Chemical Synthesis Planning (CASP) algorithms have the potential to help chemists predict how to make molecules, and decide which molecules to prioritize for synthesis and testing. Recently, several algorithms have been proposed to tackle this problem, reporting large performance improvements. In this work, we re-examine current and prior State-of-the-Art synthesis planning algorithms under controlled and identical conditions, providing a holistic view using several previously un-reported evaluation metrics which cover the common use-cases of these algorithms. In contrast to prior studies, we find that under strict control, differences between algorithms are smaller than previously assumed. Our findings can guide users to choose the appropriate algorithms for specific tasks, as well as stimulate new research in improved algorithms.

## 1 Introduction/Motivation

The synthesis of small organic molecules (SMOLs) is an essential part of drug and materials discovery. To make a desired target molecule, one follows a synthesis route (also called plan), which consists of usually multiple reactions one has to run starting from commercially available building block molecules to obtain the target. Thus, a route resembles a cooking recipe, where the target molecule is the desired final dish, and the the building blocks are ingredients. The most common approach to come up with a route is by decomposing the target molecule recursively using formally reversed chemical reactions, until one reaches the building blocks. In chemistry, this technique is called *retrosynthesis*.

In contrast to peptides, proteins, or DNA/RNA, which can be made by a small number of possible chemical reaction types, the structural diversity of SMOLs requires thousands of different reaction types. This makes route planning challenging, as reaction types correspond to different options that can be taken at each step in the recursive decomposition, which makes retrosynthesis hard even for the best human chemists.

Computational approaches for synthesis planning have been studied for at least six decades [Todd, 2005, Strieth-Kalthoff et al., 2020]. Recently, the use of machine learning for synthesis planning has led to considerable progress in particular in the quality of the proposed solutions, and to renewed interest in the field [Segler et al., 2018, Coley et al., 2019, Kishimoto et al., 2019, Schwaller et al., 2020, Genheden et al., 2020, Chen et al., 2020, Kim et al., 2021, Xie et al., 2022].

Conceptually, synthesis planning algorithms have four components: 1) A reaction prediction module, to propose single-step retrosynthetic disconnections that correspond to feasible reactions in the

Submitted to NeurIPS 2022 AI for Science Workshop.

forward direction, 2) a search or planning algorithm that can handle large branching factors, 3) a building block library, 4) ranking criteria for the returned routes, for example by cost.

In this work, we re-evaluate component 2), the search algorithms. We report preliminary findings which suggest that the performance of different search algorithms may be much more similar than previously reported. Specifically, after re-benchmarking two MCTS variants against the Retro* algorithm [Chen et al., 2020] on four distinct sets of test molecules, we find that both the time at which a solution is found and the diversity of solutions found is essentially indistinguishable for all algorithms. We believe these results raise some important questions for the community, and help to set best practice for future researchers in this area.

## 2 Experimental Procedure

**Algorithm classes considered.** We focus our experiments on two algorithm classes: MCTS [Browne et al., 2012] and Retro* [Chen et al., 2020], which is inspired by A*. MCTS is chosen because of its initial successful results for synthesis planning [Segler et al., 2017], and the subsequent attention and adoption that it received from the chemistry community (e.g. [Segler et al., 2018, Coley et al., 2019, Genheden et al., 2020]). Retro* [Chen et al., 2020] is selected as it reported a significant improvement over MCTS , and because the algorithm operates on an AND/OR tree instead of an OR tree from Segler et al. [2018], Coley et al. [2019], Genheden et al. [2020], which is a more natural data structure for synthesis trees. We did not run initial experiments using proof number search [Kishimoto et al., 2019] at this time due to the lower performance of this method reported in Genheden and Bjerrum [2022], although we intend to run these experiments at a later date.

**Algorithms used.** From these algorithm classes, we chose three specific algorithms: Retro* on AND/OR trees, MCTS as implemented by Segler et al. [2018], Coley et al. [2019], Genheden et al. [2020] on OR trees, and a novel modified version of MCTS which uses an AND/OR tree (AO-MCTS). All algorithms are implemented in a similar way as part of a larger python package for retrosynthesis, ensuring that performance differences due to implementation style are minimal. All algorithms use the same reaction model and policy, and we run each algorithm with two different value functions: a constant value function, and a pre-trained neural network value function from Chen et al. [2020]. This is an important departure from previous papers, which used different value functions for different algorithms (often rollouts for MCTS), making it unclear how much of performance differences are due to the algorithm and how much are due to the value function. Further details can be found in Appendix A.

**Evaluation metrics.** We chose to evaluate performance with two metrics: the time at which a solution is found, and the number of distinct solutions found over time. The first metric was chosen as an improvement over previous papers, which simply report the fraction of molecules which have at least one solution by a given time, which does not yield any insight about *when* within that time window solutions were found. The second metric was chosen to represent route diversity. Although chemists generally want CASP algorithms to propose multiple diverse synthesis routes, most previous works have not evaluated diversity. Our metric, which specifically reports the number of synthesis routes found at time $t$ such that no molecules in any two synthesis routes overlap, was chosen based on the diversity-metric guidance from Xie et al. [2021]. Among other things, this measure of diversity guarantees that adding an additional solution to an existing set of solutions will never result in the diversity decreasing: a property which does not hold for other proposed measures of diversity [Genheden and Bjerrum, 2022].

**Use of caching.** When calculating the metrics above, we have followed previous works in using the number of calls to the reaction model as a measure of time, since this tends to be the most computationally-intensive part of the search. However, unlike previous works, we use a cache for the reaction model, meaning that after the reactions for a molecule $M$ are predicted once, if $M$ is encountered again its reactions are retrieved from the cache instead of calling the model. This may

sound like a minor implementation detail, but it actually has a significant impact on the evaluation. Often very large sub-trees can occur in multiple places during search:[1] without a cache, expanding each occurrence of these subtrees will count against an algorithm's time budget, whereas with a cache these expansions are effectively free. Given that any realistic implementation of these algorithms would use caching to minimize computational cost, we think that it is important to use caching when making any sort of real-world comparison between algorithms.

**Experiments run.** For all experiments, each algorithm is allowed to run for a maximum of 300 s on each molecule tested. While previous papers tend to focus on a single testing setup (i.e. using a single reaction model, set of purchasable molecules, and set of test molecules), we think that testing on multiple setups is helpful to obtain a comprehensive picture of the performance differences of each algorithm. Therefore we use the following setups.

1. **Retro* Hard:** 190 hard molecules used in Chen et al. [2020], using the reaction model and set of purchasable molecules from Chen et al. [2020].
2. **PaRoutes:** 200 randomly chosen molecules from the $n = 5$ setting of PaRoutes [Genheden and Bjerrum, 2022], using their pre-trained reaction model as the policy for each algorithm.
3. **ChemBL Hard:** 200 randomly chosen "hard" molecules chosen from the ChemBL dataset, with the reaction model and purchasable molecules from Chen et al. [2020]. To create a set of molecules equally or more challenging than the Retro* set, molecules were chosen by applying the pre-processing script from [Brown et al., 2019], keeping only molecules whose molecular weight, Bertz coefficient, logP and TPSA were larger the mean of the respective values in the Retro* hard set, and removing all purchasable molecules.
4. **GDB17:** 200 randomly chosen molecules from gdb17 [Ruddigkeit et al., 2012], using the reaction model and set of purchasable molecules from Chen et al. [2020].

**Summary.** Overall, the most important parts of our experimental procedure are:

- Algorithm Implementation:
  - All algorithms implemented by us in a similar way as part of a common code base.
  - Use the same policies and value functions for all algorithms.
- Analysis of performance
  - Show *when* each algorithm finds a solution, not just whether a solution is found at an arbitrary end time.
  - Measure and evaluate diversity of routes found in a principled way
  - Use caching of the reaction model and account for this when measuring algorithm runtime.
  - Test in 4 different scenarios

## 3 Results and Discussion

The results from our experiments are summarized in Figure 1. The most important observation in this figure is the high amount of overlap between the evaluation metrics for all different algorithms. Looking at the left column, for all four test sets, the box plots for time of first solution for all different algorithms almost entirely overlap. Looking at the right column, for all test sets the algorithms all find approximately the same number of non-overlapping routes, although they seem to find additional routes at slightly different rates (with no clear pattern across the different test sets). This result may seem surprising, because it directly contradicts the findings of Chen et al. [2020], who report that MCTS performs far worse than Retro*. However, our findings are very similar to those of Genheden and Bjerrum [2022], who also report very little difference between MCTS and Retro*.

[1] For example, if the reactions $M \rightarrow A + B$ and $M \rightarrow A + C$ are possible, then any subsequent reactions on molecule $A$ will be repeated multiple times.

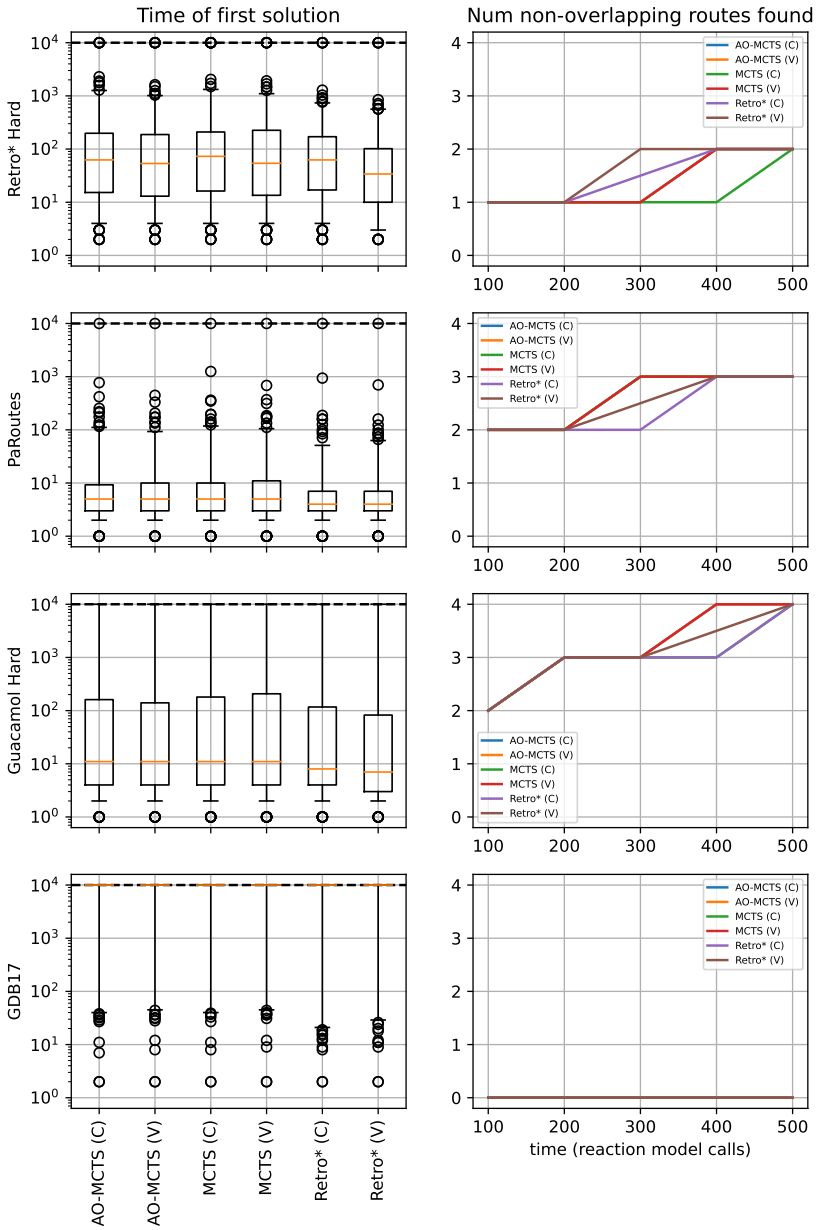


Figure 1: Evaluation metrics for different algorithms, where time is measured by number of calls to the reaction model (with caching). Each row represents a different test set, explained in Section 2. **Left:** The box plots in the left column represent the times at which the first solution was found for all tested molecules. The orange line represents the median, the top/bottom of the box represent the 25th/75th percentile, and the whiskers represent 5th and 95th percentile respectively. Points outside this range are plotted as dots. The black dashed line represents the value assigned to molecules that were not solved. For this metric, *lower is better*. **Right:** The plots in the right column plot the median number of distinct solutions found for each molecule (i.e. a reading of $n$ at time $t$ implies that 50% of the test molecules had $\geq n$ distinct solutions at time $t$). For this metric, *higher is better*.

125

Further examining Figure 1 yields several additional insights:

- **Difficulty of different test sets:** The easiest set seems to be that of PaRoutes [Genheden and Bjerrum, 2022], where all algorithms find a solution for 75% of the test molecules within just 10 calls to the reaction model. The hardest set seems to be GDB17, where over 75% of the molecules are not solved at all. The Retro* Hard and ChemBL Hard test sets are somewhere in between.
- **Effect of the value function:** Comparing each algorithm run with a constant value function (C) to that same algorithm run with a neural network value function (V) reveals that the value function seems to help, but only slightly. This is most visible on the Retro* Hard test set. It is consistent with the results reported by Chen et al. [2020] that setting the value function to 0 for all molecules resulted in only a mild decrease in performance.

We also analyzed the results measuring time instead by wall-clock time (Figure 2) and by number of calls to the value function (Figure 3). In general, the conclusions measuring time with wall-clock are very similar to those measuring time with number of calls to the reaction model: perhaps unsurprising since calling the reaction model is the most time-consuming part of search. However, when time is measured by number of calls to the value function, Retro* is decisively worse than MCTS: solutions are found later, and fewer solutions are generally found by any given time. We believe this is because for MCTS, only a handful of molecules are evaluated after the reaction model is run (since a single random action is taken), however for Retro* all molecules produced by the value function are evaluated with the reaction model (because the minimum cost path must be computed). This suggests that Retro* would not perform well if an expensive value function were used (e.g. rollouts).

## 4 Conclusions and Future Work

After a careful re-implementation of existing methods, our main finding in Section 3 is that the performance of different CASP algorithms is much more similar than previously reported. This seems to suggest either that newer algorithms are needed to improve upon the deficiencies of existing algorithms, or that the choice of search algorithm is less important than previously thought – perhaps the most important choice is a good policy and value function. Future work will include adding additional algorithms such as proof number search [Kishimoto et al., 2019] to the benchmark, and expanding our evaluation metrics to also include route quality.

## References


Nathan Brown, Marco Fiscato, Marwin HS Segler, and Alain C Vaucher. Guacamol: benchmarking models for de novo molecular design. *Journal of chemical information and modeling*, 59(3): 1096–1108, 2019.

Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

Binghong Chen, Chengtao Li, Hanjun Dai, and Le Song. Retro*: learning retrosynthetic planning with neural guided a* search. In *International Conference on Machine Learning*, pages 1608–1616. PMLR, 2020.

Connor W Coley, Dale A Thomas III, Justin AM Lummiss, Jonathan N Jaworski, Christopher P Breen, Victor Schultz, Travis Hart, Joshua S Fishman, Luke Rogers, Hanyu Gao, et al. A robotic platform for flow synthesis of organic compounds informed by ai planning. *Science*, 365(6453): eaax1566, 2019.

Samuel Genheden and Esben Bjerrum. Paroutes: towards a framework for benchmarking retrosynthesis route predictions. *Digital Discovery*, 1(4):527–539, 2022.

Samuel Genheden, Amol Thakkar, Veronika Chadimová, Jean-Louis Reymond, Ola Engkvist, and Esben Bjerrum. Aizynthfinder: a fast, robust and flexible open-source software for retrosynthetic planning. *Journal of cheminformatics*, 12(1):1–9, 2020.

Junsu Kim, Sungsoo Ahn, Hankook Lee, and Jinwoo Shin. Self-improved retrosynthetic planning. In *International Conference on Machine Learning*, pages 5486–5495. PMLR, 2021.

Akihiro Kishimoto, Beat Buesser, Bei Chen, and Adi Botea. Depth-first proof-number search with heuristic edge cost and application to chemical synthesis planning. *Advances in Neural Information Processing Systems*, 32, 2019.

Lars Ruddigkeit, Ruud Van Deursen, Lorenz C Blum, and Jean-Louis Reymond. Enumeration of 166 billion organic small molecules in the chemical universe database gdb-17. *Journal of chemical information and modeling*, 52(11):2864–2875, 2012.

Philippe Schwaller, Riccardo Petraglia, Valerio Zullo, Vishnu H Nair, Rico Andreas Haeuselmann, Riccardo Pisoni, Costas Bekas, Anna Iuliano, and Teodoro Laino. Predicting retrosynthetic pathways using transformer-based models and a hyper-graph exploration strategy. *Chemical science*, 11(12):3316–3325, 2020.

Marwin Segler, Mike Preuß, and Mark P Waller. Towards" alphachem": Chemical synthesis planning with tree search and deep neural network policies. *arXiv preprint arXiv:1702.00020*, 2017.

Marwin HS Segler, Mike Preuss, and Mark P Waller. Planning chemical syntheses with deep neural networks and symbolic ai. *Nature*, 555(7698):604–610, 2018.

Felix Strieth-Kalthoff, Frederik Sandfort, Marwin HS Segler, and Frank Glorius. Machine learning the ropes: principles, applications and directions in synthetic chemistry. *Chemical Society Reviews*, 49(17):6154–6168, 2020.

Matthew H Todd. Computer-aided organic synthesis. *Chemical Society Reviews*, 34(3):247–266, 2005.

Shufang Xie, Rui Yan, Peng Han, Yingce Xia, Lijun Wu, Chenjuan Guo, Bin Yang, and Tao Qin. Retrograph: Retrosynthetic planning with graph search. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 2120–2129, 2022.

Yutong Xie, Ziqiao Xu, Jiaqi Ma, and Qiaozhu Mei. How much of the chemical space has been covered? measuring and improving the variety of candidate set in molecular generation. *arXiv preprint arXiv:2112.12542*, 2021.

# A Details on Algorithm Implementations

All code was written in python. For retro*, the cost of all reactions was the negative log probability from the template classifier, as in [Chen et al., 2020]. The constant value function was given a value of 0, and the outputs of the neural network value function was used without adjustment. For both AND/OR and OR MCTS, the P-UCT bound function from Segler et al. [2018] was used with $C = 10$, and $P$ values obtained by applying softmax normalization to the log probabilities from the template classifier with a softmax temperature $T = 3$. The values of $T$ and $C$ were chosen from a hyperparameter sweep on a small set of sample molecules. The constant value function had a value of $0.5$, and the values from the neural network value function were adjusted by $f(x) = \exp(-x)$ (because this neural network predicts a *cost*, it needed to be adjusted so that higher cost implied lower value). To prevent infinite expansion, the search trees for all algorithms were limited to a maximum depth of 20 for all tree types.

AND/OR MCTS is a simple variant of MCTS on AND/OR tree. Like OR MCTS, it is an iterative algorithm where one starts at the root node, descends the tree to a leaf node, either expands or evaluates this node, then propagates a series of values upwards. At an OR node, an AND node is selected using the same P-UCT method as in OR MCTS (whichever node has the highest value + exploration bound). At an AND node, *all* of its children are visited, and the value propagated upwards is the *minimum* value of the children. This is the key difference to normal MCTS, where only a single child of one node is visited.

# B Additional Results

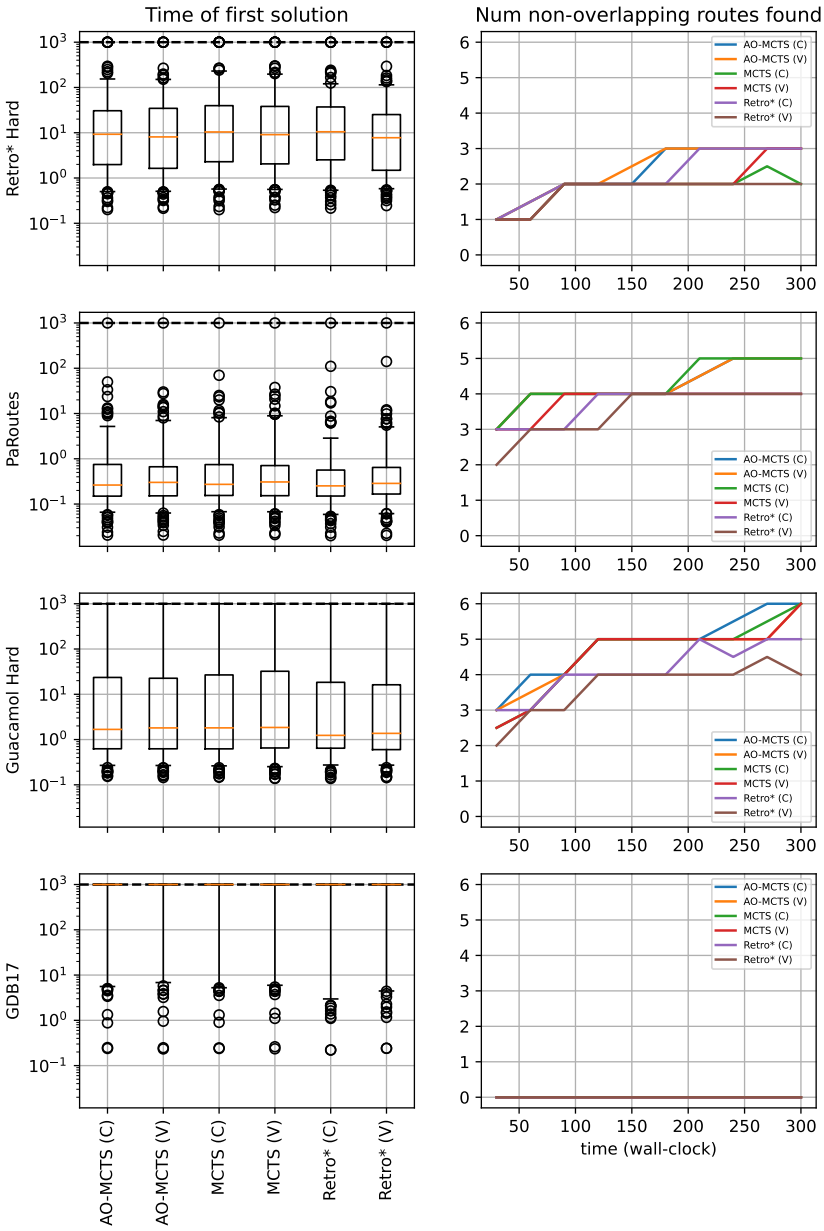


Figure 2: Identical to Figure 1 but with time measured by wallclock time (in seconds)

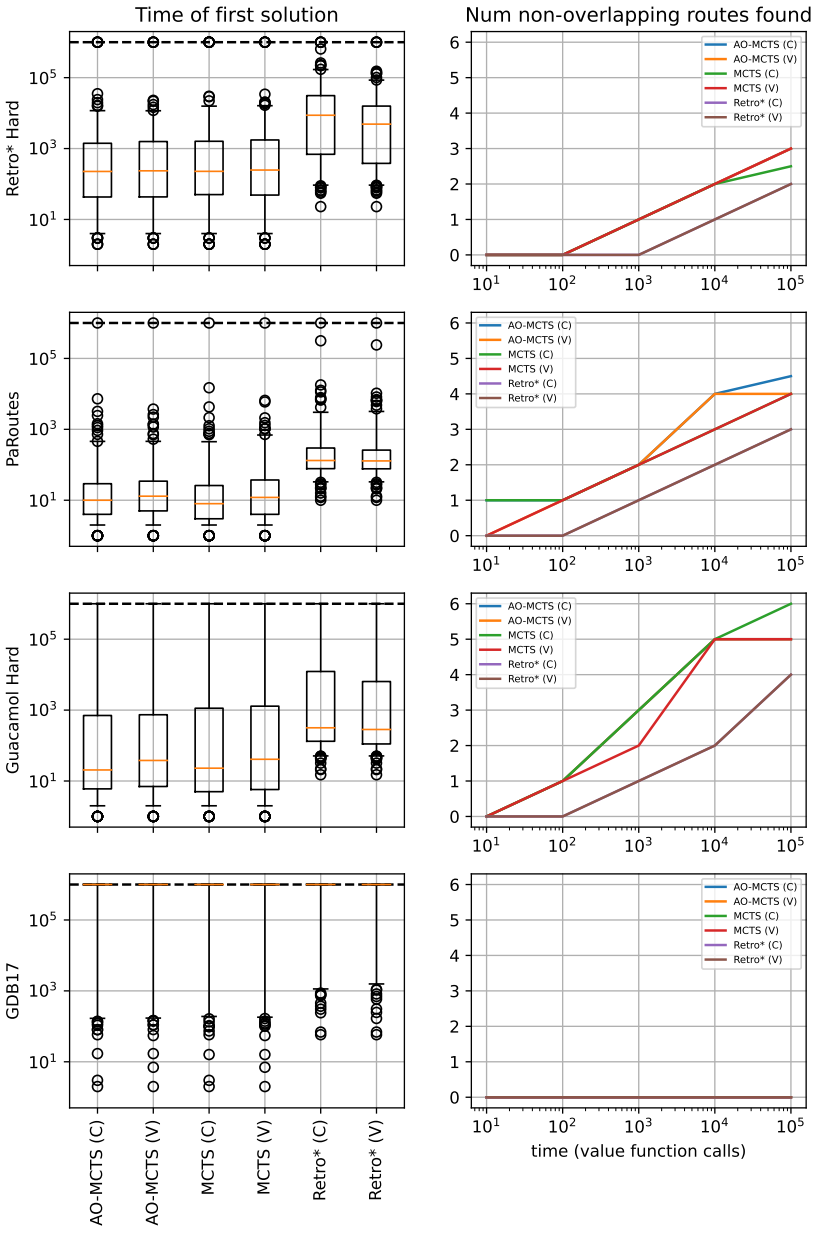


Figure 3: Identical to Figure 1 but with time measured by number of calls to the value function.