On the optimization of imaging pipelines

Schoonhoven, R.A.

Curriculum Vitae

Richard Schoonhoven was born in 1995 in Utrecht, the Netherlands. He completed his secondary education at RSG Broklede in Breukelen, the Netherlands. Afterwards, he completed bachelor's degrees (both cum laude) in mathematics and physics at Utrecht University in 2016, and obtained master's degrees (both cum laude) in mathematics and computer science in 2019 from Utrecht University. The master's thesis with the title "Improving cryo-ET reconstructions of ER-associated ribosomes with tomographic reconstruction methods and deep learning" was supervised by Dr. Tristan van Leeuwen. In 2019, he started as a PhD candidate at Centrum Wiskunde & Informatica (the national research institute for mathematics and computer science in Amsterdam) under the supervision of Prof.dr. K.J. Batenburg.

Acknowledgments

First, I thank my advisors, Prof. Joost Batenburg and Dr. Daniël Pelt, for the time and energy they have put into providing motivating, educational, and fun supervision of our research projects. In particular, I would like to thank them for providing me with large amounts of freedom to pursue different research directions, which has made the past four years a thoroughly enjoyable part of my career.

In addition, I would like to thank Dr. Ben van Werkhoven for his energetic and stimulating approach to research and supervision, which has made several of our projects a great pleasure to collaborate on.

I would also like to thank my colleague Dr. Alexander Skorikov for his companionable collaboration, and his tireless energy for dealing with my ceaseless stream of thoughts, and at times, tumultuous working style.

A special thank you goes out to my co-authors at the ESRF who have hosted me for several weeks on many occasions, and for their efforts to make me feel welcome and show me around their impressive facilities. I have to mention in particular Dr. Alexandra Pacureanu who was kind enough to allow me to stay in their city apartment in Grenoble during these trips.

I would like to thank my co-authors Allard Hendriksen, Bram Veenboer, and Jan-Willem Buurlage, who have greatly helped me with my research projects and spent time and effort teaching me more about new topics.

In particular, I would like to thank Willem Jan Palenstijn for his expert and patient help on countless problems I encountered.

I thank the colleagues whom I shared an office with, Vladyslav Andriiashen, Mathé Zeegers, Adriaan Graas, Poulami Ganguly, Francien Bossema, Jordi Minnema, Dirk Schut, Maximilian Kiss, Tianyuan Wang, Rien Lagerwerf for providing stimulating conversations and a pleasant work environment.

Many others contributed to a great research environment, among which Floris-Jan, Jiayang, Serban, Alex, Nicola, Henri, Maureen, Dzemila, Georgios, Sophia, Felix, Rob, Robert, Hamid, Roozbeh, Ajynkya, Jan, and Tristan.

I would like to thank my family and friends for their support, and camaraderie over the years.

Finally, I would like to thank Sasha for her love, infinite patience, and wholehearted support.

A
Appendices

A.1 Appendix: (LEAN) graph-based pruning for convolutional neural networks by extracting longest chains

A.1.1 Datasets

In this appendix we discuss some more details on the datasets used for experimentation.

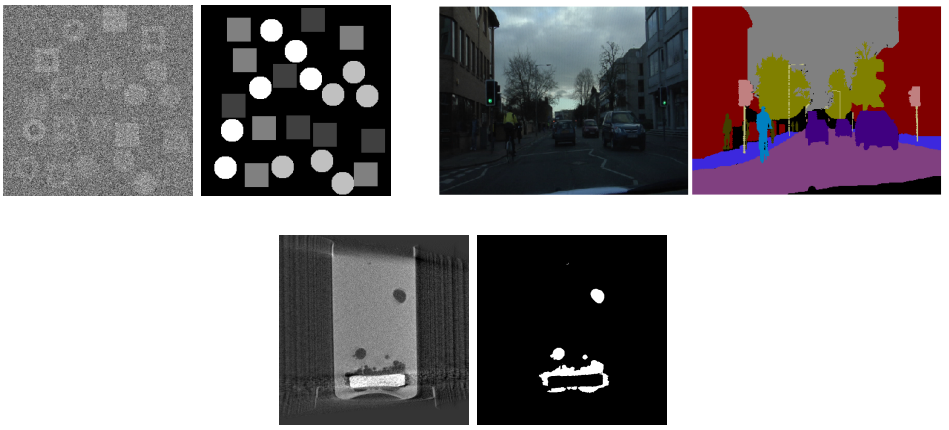
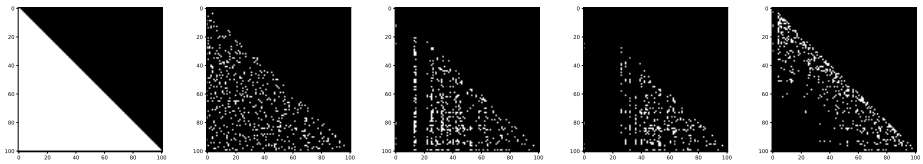


Figure A1: Example input and target images of the (top left) Circle-Square (CS), (top right) CamVid, (bottom) real-world dynamic CT datasets.

Simulated Circle-Square (CS) dataset: We used a simulated high-noise 5-class segmentation dataset containing 256×256 images of randomly placed squares and circles (CS dataset) [183] (see Figure A1). The objects were assigned a random grey value and Gaussian noise was added to the images. In total, we generated 1000 training, 250 validation, and 100 test images. Experimental results on the CS dataset are quantified using global accuracy, i.e., the ratio of correctly classified pixels, regardless of class, to the total number of pixels.

CamVid: The Cambridge-driving Labeled Video Database (CamVid) [24, 25] is a collection of videos with labels, captured from the perspective of a driving automobile. In total, 700 labeled frames are split into 367 training, 100 validation, and 233 test images. As there are few training images, we combined the training and validation datasets and trained for a fixed 500 epochs. Similar to other papers that apply CNNs to CamVid [9, 180], we use 11 classes, and a single class representing unlabeled pixels (see Figure A1).

We used median frequency balancing [56] to balance classes for training, and set the unlabeled class weights to zero. During training, we used data augmentation by cropping and (horizontally) flipping input images.



Unpruned Random Magnitude Operator norm LEAN

Figure A2: Adjacency matrices of active convolutions (in white) after pruning. All pruned network were pruned to a ratio of 10%. From left to right, we have the unpruned matrix of a 100-layer MS-D network trained on the real-world dynamic CT dataset, randomly pruned convolutions, structured magnitude pruning, structured operator norm pruning, and LEAN.

Real-world dynamic CT dataset: The real-time dynamic X-ray CT dataset contains images of a dissolving tablet suspended in gel [38, 39]. The bubbles are to be segmented within a glass container filled with gel [209] (see Figure A1). The dataset consists of 512×512 images, split into 9216 training images, 2048 validation images, and 1536 test images. As in [209], we use the F1-score because the large amount of background pixels make global accuracy an unsuitable metric.

A.1.2 Reducing the size of the pruning graph

The procedure outlined in Section 4.4 can lead to large pruning graphs, but the size of the graph can be reduced. First, according to Equation 4.3, the operator norm of ReLU is 1. Therefore, the combination of a convolution followed by a ReLU can be combined into a single edge whose weight equals the norm of the convolution.

Batch normalization often succeeds a convolution. Batch-normalization scaling is applied with different learned parameters per input channel, and output a single channel. Therefore, the input convolution edge and the following batch normalization edges can be combined. The edges can be combined into a single edge whose weight is the product of the two edge weights, preserving the path length.

A.1.3 Structure of pruned MS-D networks

To investigate the structure of pruned networks we plotted the adjacency matrices of pruned networks where an entry is 0 if it is pruned (black) and 1 if it is still active (white). Here, we show the adjacency matrices of MS-D networks pruned to a ratio of 10% in Figure A2. After pruning, LEAN retains only connections linked to nearby layers in the densely connected MS-D network. Compared to individual filter pruning, LEAN exposes a distinct structure which may suggest that LEAN could be used for architecture discovery.

A.2 Appendix: Benchmarking optimization algorithms for auto-tuning GPU kernels

A.2.1 Tunable parameters per GPU kernel

In Table A.1 we show the tunable parameters per kernel, and the values each parameter could take. For the convolution kernel, the MI50 GPU (the only AMD model) required a different problem setup due to hardware constraints.

Kernel	parameter to tune	list of values	number of possible values
Convolution (except MI50)	`block_size_x`	1, 2, 4, 8, 16, 32, 48, 64, 80, 96, 112, 128	12
	`block_size_y`	1, 2, 4, 8, 16, 32	6
	`tile_size_x`	1, 2, 3, 4, 5, 6, 7, 8	8
	`tile_size_y`	1, 2, 3, 4, 5, 6, 7, 8	8
	`use_padding`	0, 1	2
	`read_only`	0, 1	2
Convolution (MI50)	`block_size_x`	16, 32, 48, 64, 80, 96, 112, 128	8
	`block_size_y`	1, 2, 4, 8, 16, 32	6
	`tile_size_x`	1, 2, 4	3
	`tile_size_y`	1, 2, 4	3
	`use_padding`	0, 1	2
GEMM	`MWG`	16, 32, 64, 128	4
	`NWG`	16, 32, 64, 128	4
	`MDIMC`	8, 16, 32	3
	`NDIMC`	8, 16, 32	3
	`MDIMA`	8, 16, 32	3
	`NDIMB`	8, 16, 32	3
	`VWM`	1, 2, 4, 8	4
	`VWN`	1, 2, 4, 8	4
	`SA`	0, 1	2
	`SB`	0, 1	2
Point-in-polygon	`block_size_x`	32, 64, 96, 128, 160, 192, 224, 256, 288, 320, 352, 384, 416, 448, 480, 512, 544, 576, 608, 640, 672, 704, 736, 768, 800, 832, 864, 896, 928, 960, 992	31
	`tile_size`	1, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20	11
	`between_method`	0, 1, 2, 3	4
	`use_precomputed_slopes`	0, 1	2
	`use_method`	0, 1, 2	3

Table A.1: Tunable parameters per kernel, and list of possible values for each parameter.

A.2.2 Alternative splits for competition heatmaps

In Figures A1 and A6 we show the algorithm competition heatmaps such as in Figures 5.1, 5.2 and 5.3, but when split at 100 and 400 budgets instead of 200.

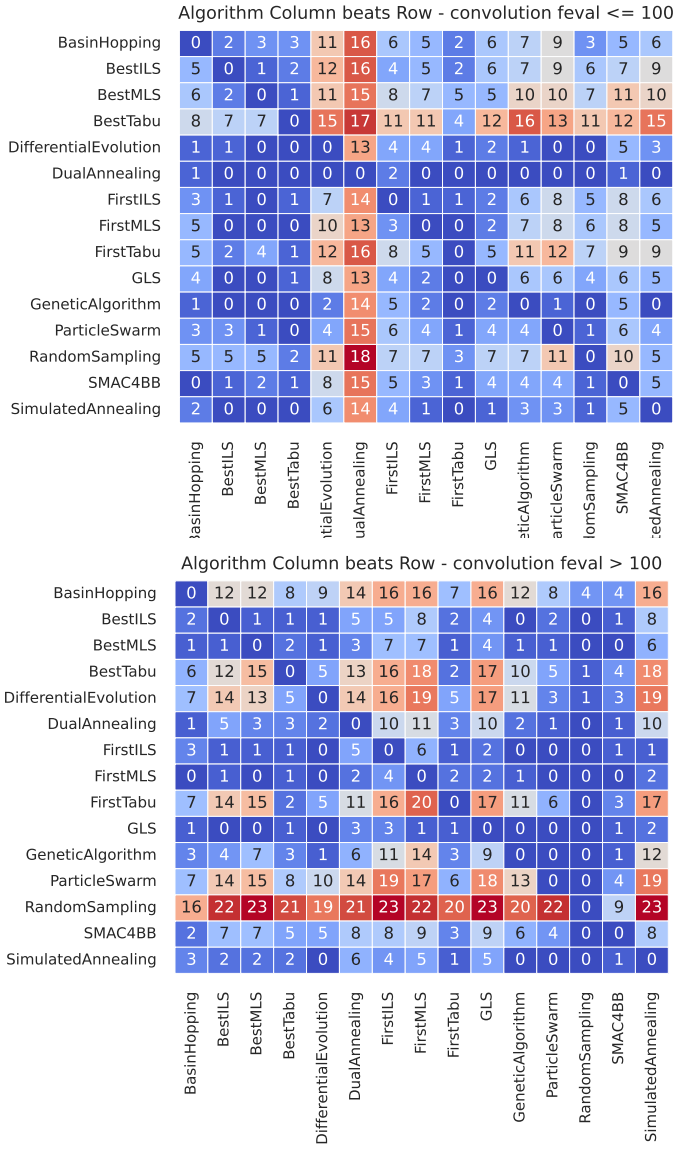


Figure A1: (Convolution:) Occurrences when the column algorithm found better solutions than the row algorithm. An occurrence is counted when 50 runs for a budget are statistically significantly better according to a two-sample independent t-test ($\alpha = 0.05$). (Top): Heatmap for low ≤ 100 budgets (25, 50, 100). (Bottom): Heatmap for mid and high > 100 budgets (200, 400, 800, 1600). Algorithms with low values (blue) in their rows were not often beaten for those budgets, and algorithms with high values in their column (red) often beat other algorithms.

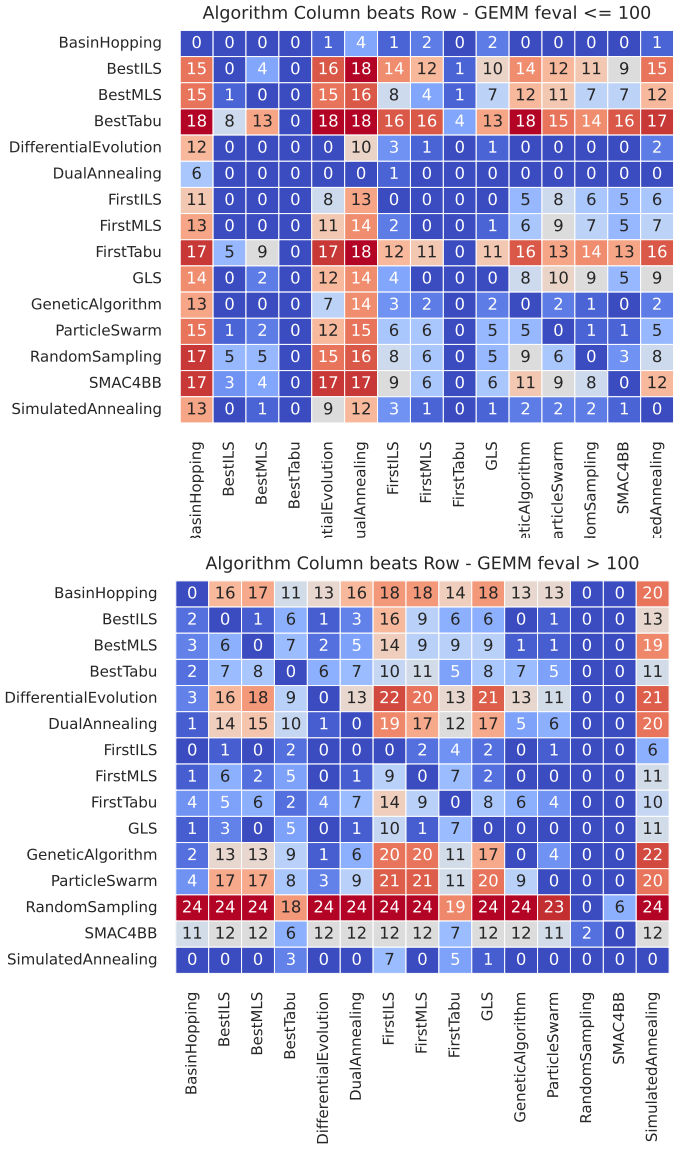


Figure A2: (GEMM:) Occurrences when the column algorithm found better solutions than the row algorithm. An occurrence is counted when 50 runs for a budget are statistically significantly better according to a two-sample independent t-test ($\alpha = 0.05$). (Top): Heatmap for low ≤ 100 budgets (25, 50, 100). (Bottom): Heatmap for mid and high > 100 budgets (200, 400, 800, 1600). Algorithms with low values (blue) in their rows were not often beaten for those budgets, and algorithms with high values in their column (red) often beat other algorithms.

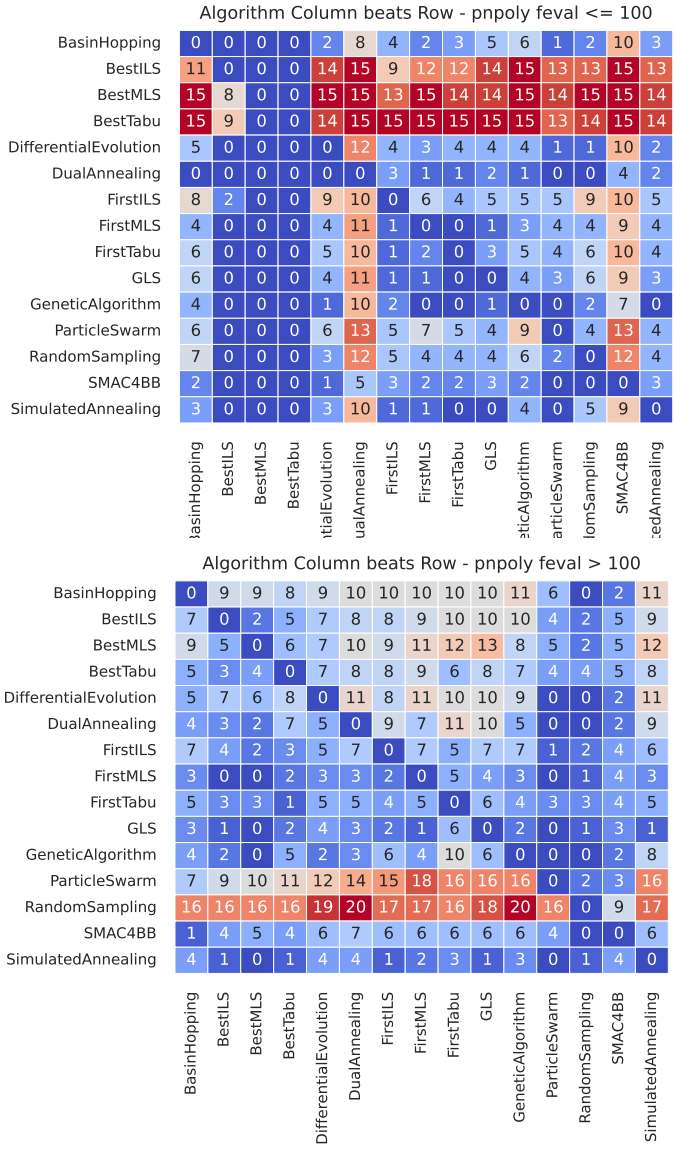


Figure A3: (PnPoly:) Occurrences when the column algorithm found better solutions than the row algorithm. An occurrence is counted when 50 runs for a budget are statistically significantly better according to a two-sample independent t-test ($\alpha = 0.05$). (Top): Heatmap for low ≤ 100 budgets (25, 50, 100). (Bottom): Heatmap for mid and high > 100 budgets (200, 400, 800, 1600). Algorithms with low values (blue) in their rows were not often beaten for those budgets, and algorithms with high values in their column (red) often beat other algorithms.

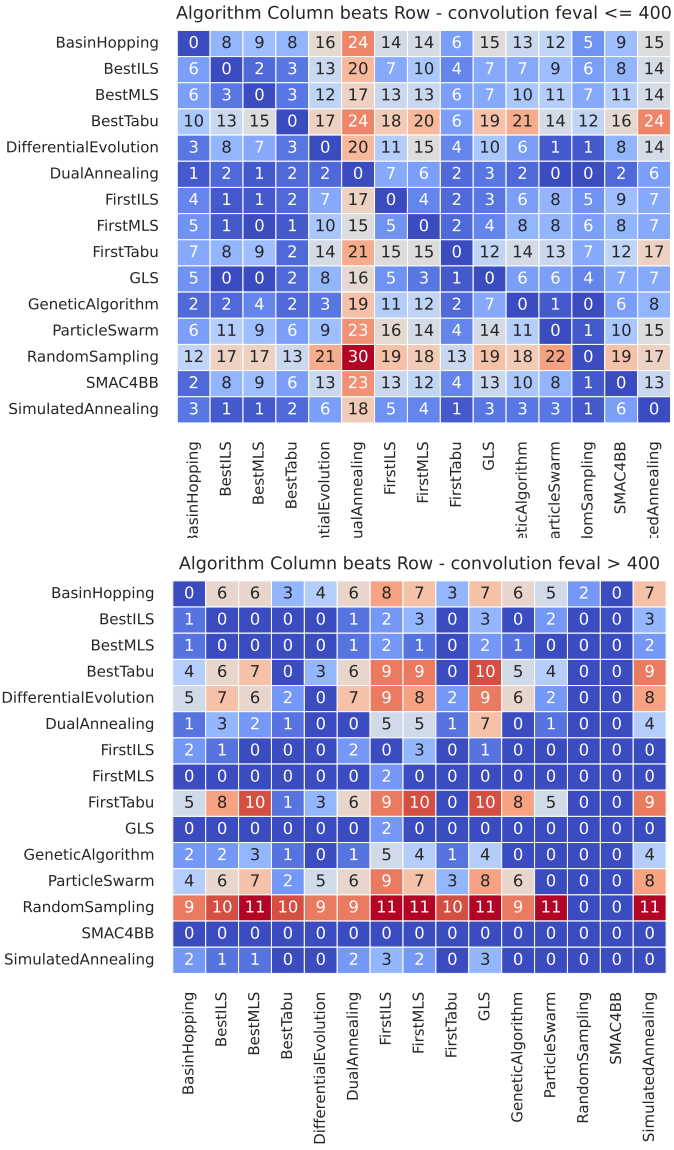


Figure A4: (Convolution:) Occurrences when the column algorithm found better solutions than the row algorithm. An occurrence is counted when 50 runs for a budget are statistically significantly better according to a two-sample independent t-test ($\alpha = 0.05$). (Top): Heatmap for low ≤ 400 budgets (25, 50, 100, 200, 400). (Bottom): Heatmap for mid and high > 400 budgets (800, 1600). Algorithms with low values (blue) in their rows were not often beaten for those budgets, and algorithms with high values in their column (red) often beat other algorithms.

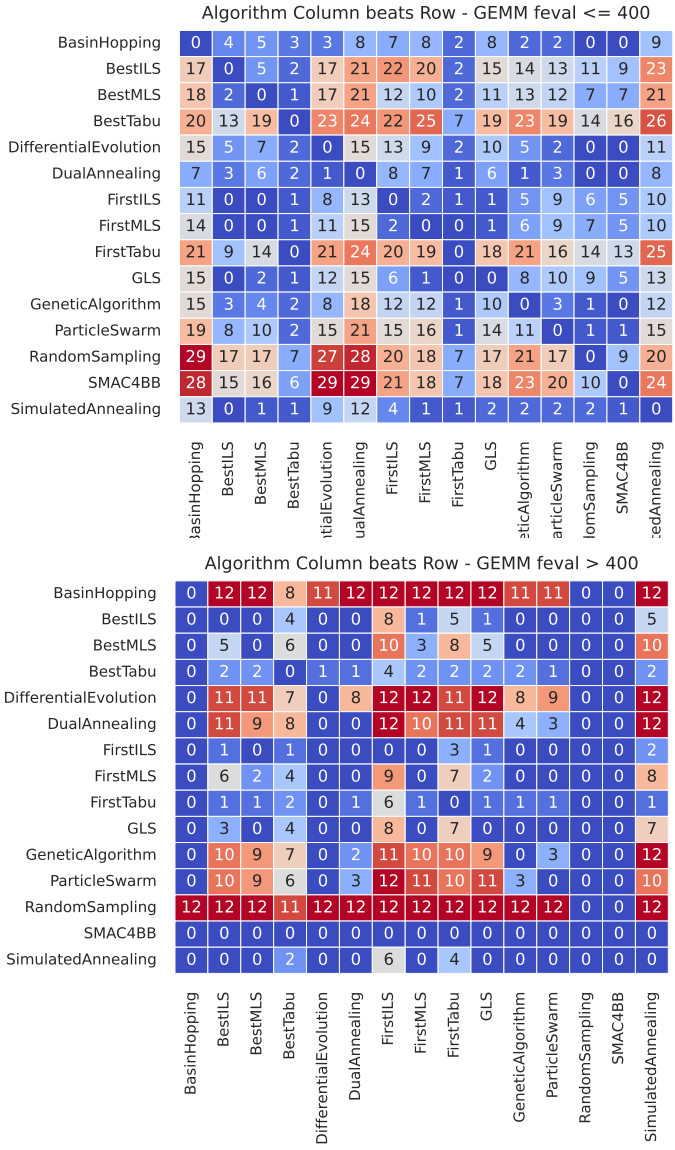


Figure A5: (GEMM:) Occurrences when the column algorithm found better solutions than the row algorithm. An occurrence is counted when 50 runs for a budget are statistically significantly better according to a two-sample independent t-test ($\alpha = 0.05$). (Top): Heatmap for low ≤ 400 budgets (25, 50, 100, 200, 400). (Bottom): Heatmap for mid and high > 400 budgets (800, 1600). Algorithms with low values (blue) in their rows were not often beaten for those budgets, and algorithms with high values in their column (red) often beat other algorithms.

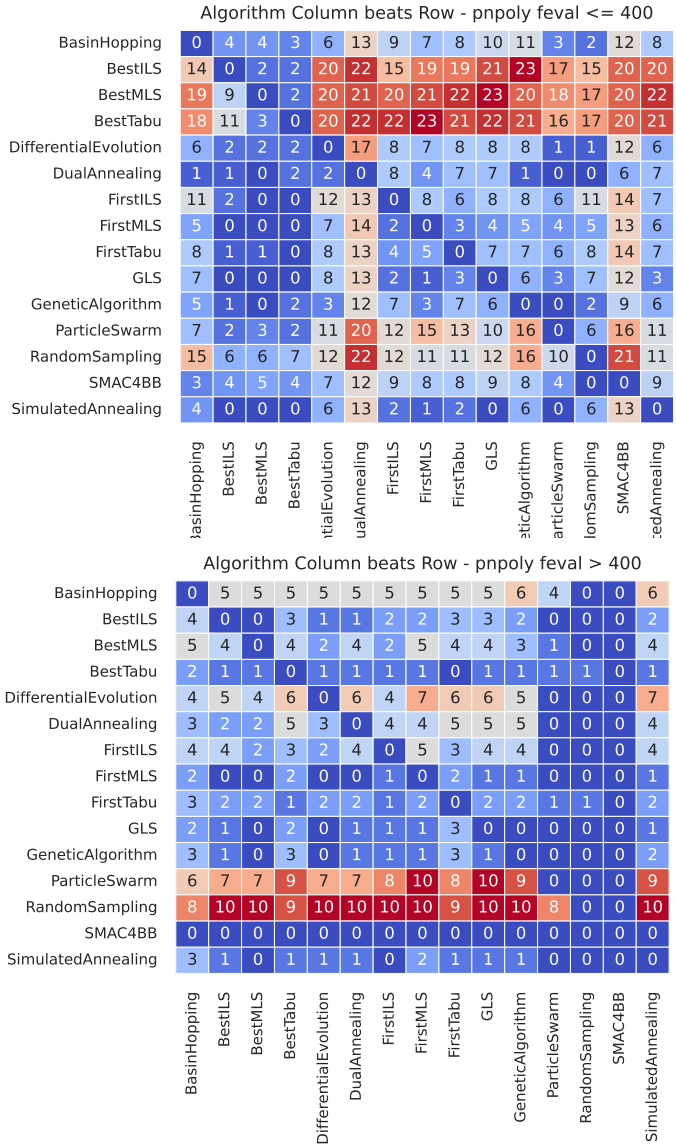


Figure A6: (PnPoly:) Occurrences when the column algorithm found better solutions than the row algorithm. An occurrence is counted when 50 runs for a budget are statistically significantly better according to a two-sample independent t-test ($\alpha = 0.05$). (Top): Heatmap for low ≤ 400 budgets (25, 50, 100, 200, 400). (Bottom): Heatmap for mid and high > 400 budgets (800, 1600). Algorithms with low values (blue) in their rows were not often beaten for those budgets, and algorithms with high values in their column (red) often beat other algorithms.

A.2.3 Per kernel graphs of experimental results

In Figures A7 to A15 we show plots of algorithm performance in terms of fraction of optimal fitness found for certain budget used (per GPU).

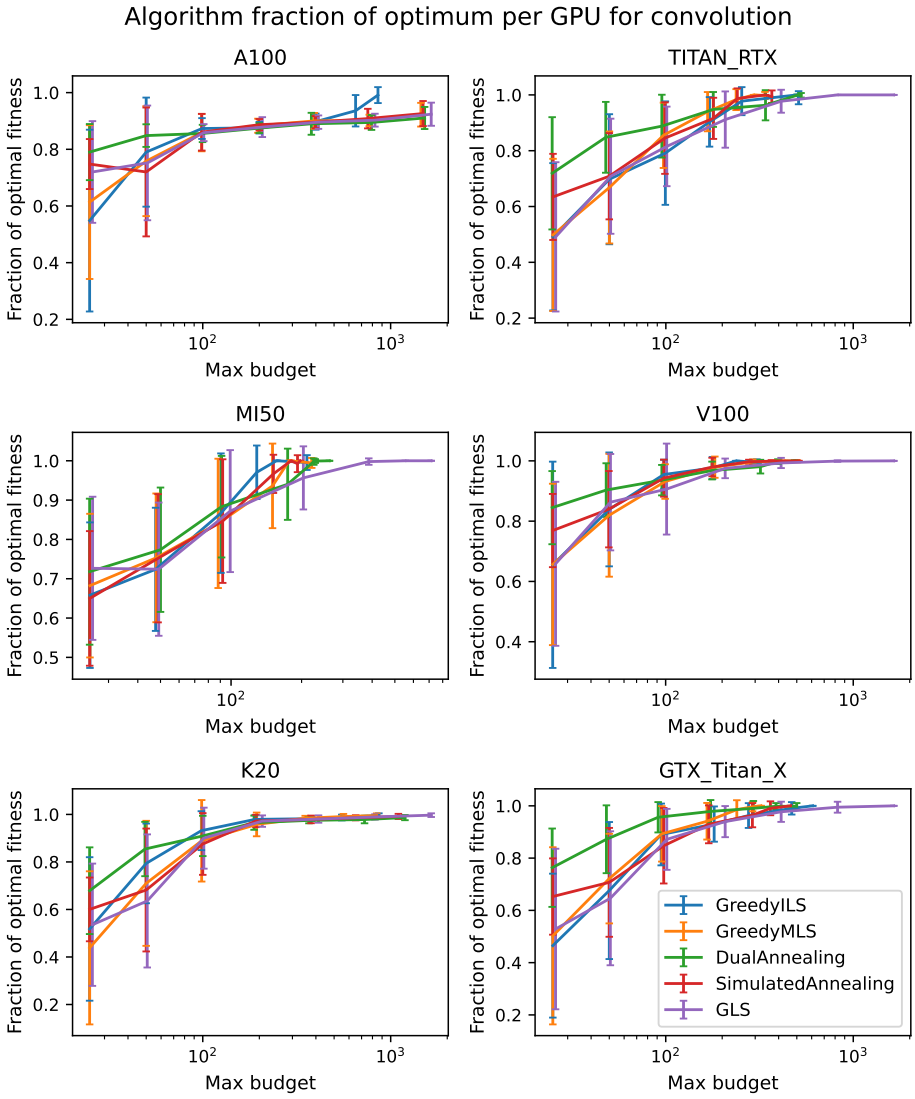


Figure A7: **Convolution:** Fraction of optimal runtime per GPU for FirstILS, FirstMLS, dual annealing, simulated annealing, and GLS over 50 runs. Each point is the mean fraction of optimal runtime found (y-axis) for mean budget used (logarithmic x-axis), with error bars indicating the standard deviation in fraction of optimum.

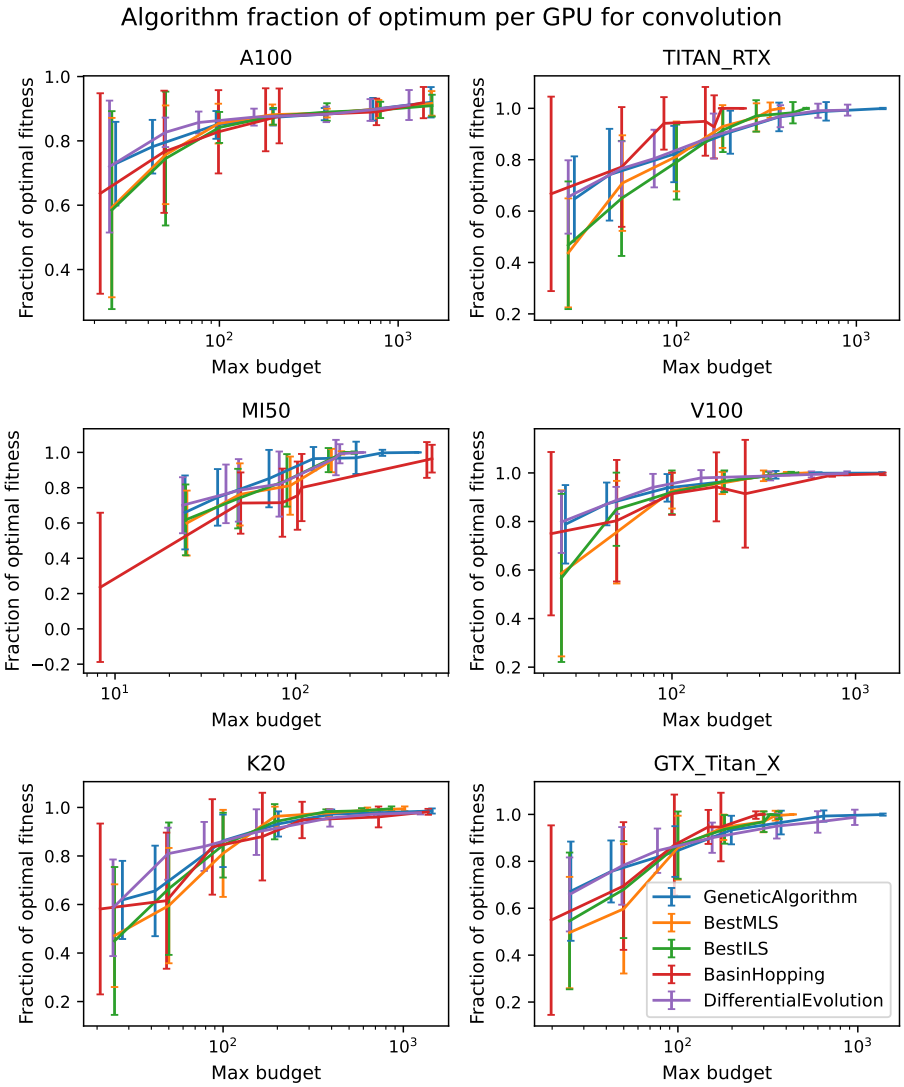


Figure A8: **Convolution:** Fraction of optimal runtime per GPU for GA, BestMLS, BestILS, basin hopping, and differential evolution over 50 runs. Each point is the mean fraction of optimal runtime found (y-axis) for mean budget used (logarithmic x-axis), with error bars indicating the standard deviation in fraction of optimum.

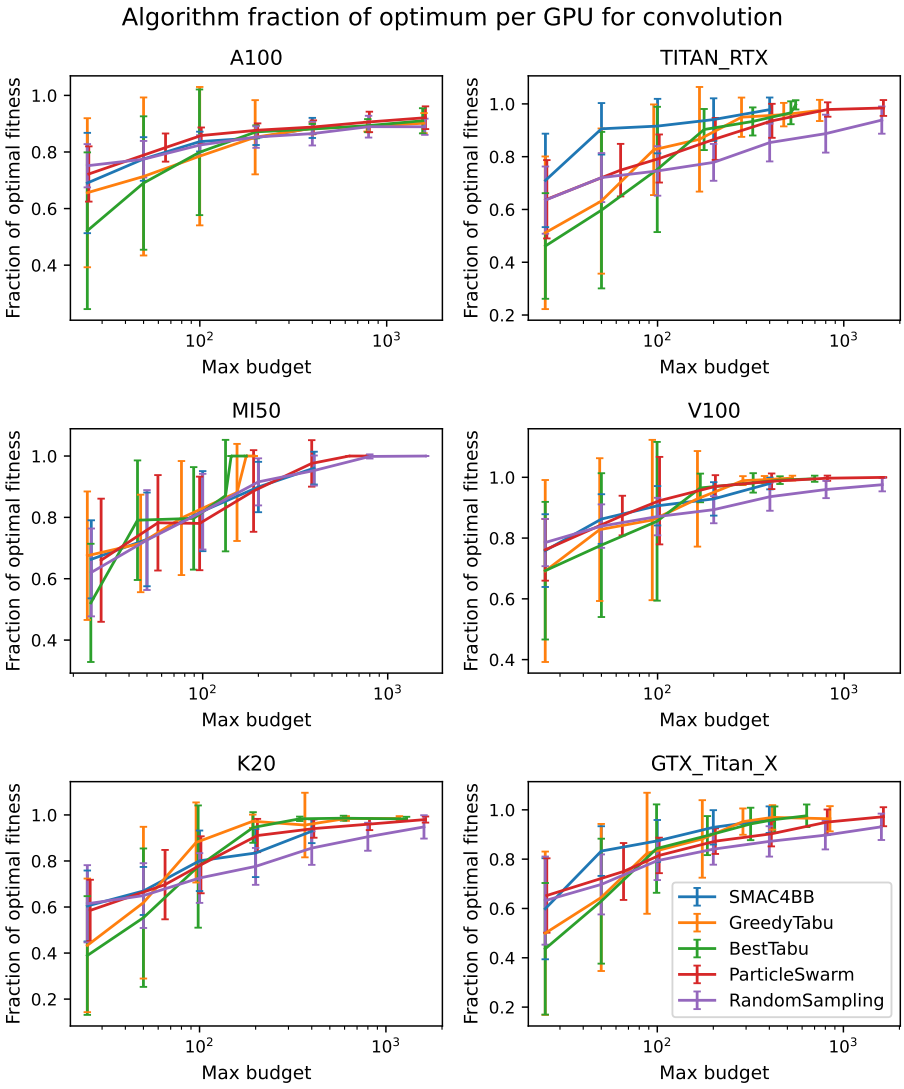


Figure A9: **Convolution:** Fraction of optimal runtime per GPU for SMAC, FirstTabu, BestTabu, PSO, and random sampling over 50 runs. Each point is the mean fraction of optimal runtime found (y-axis) for mean budget used (logarithmic x-axis), with error bars indicating the standard deviation in fraction of optimum.

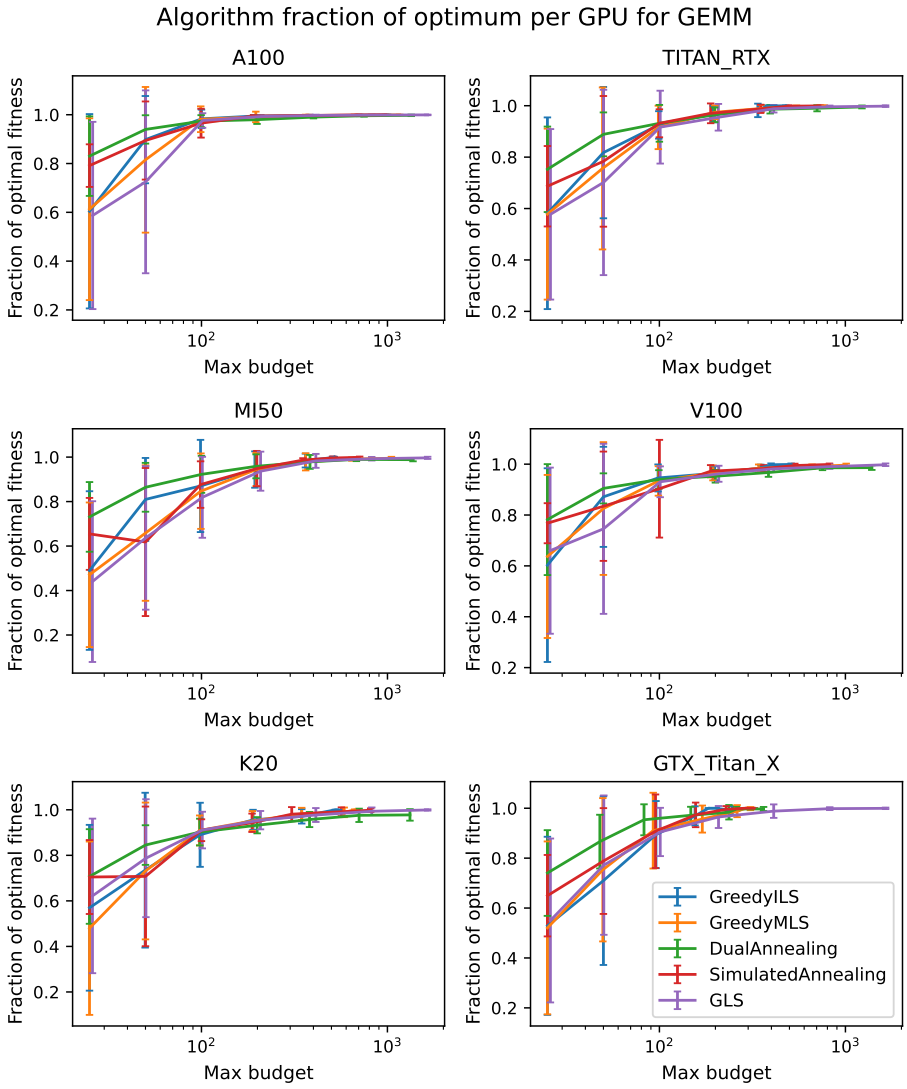


Figure A10: **GEMM:** Fraction of optimal runtime per GPU for FirstILS, FirstMLS, dual annealing, simulated annealing, and GLS over 50 runs. Each point is the mean fraction of optimal runtime found (y-axis) for mean budget used (logarithmic x-axis), with error bars indicating the standard deviation in fraction of optimum.

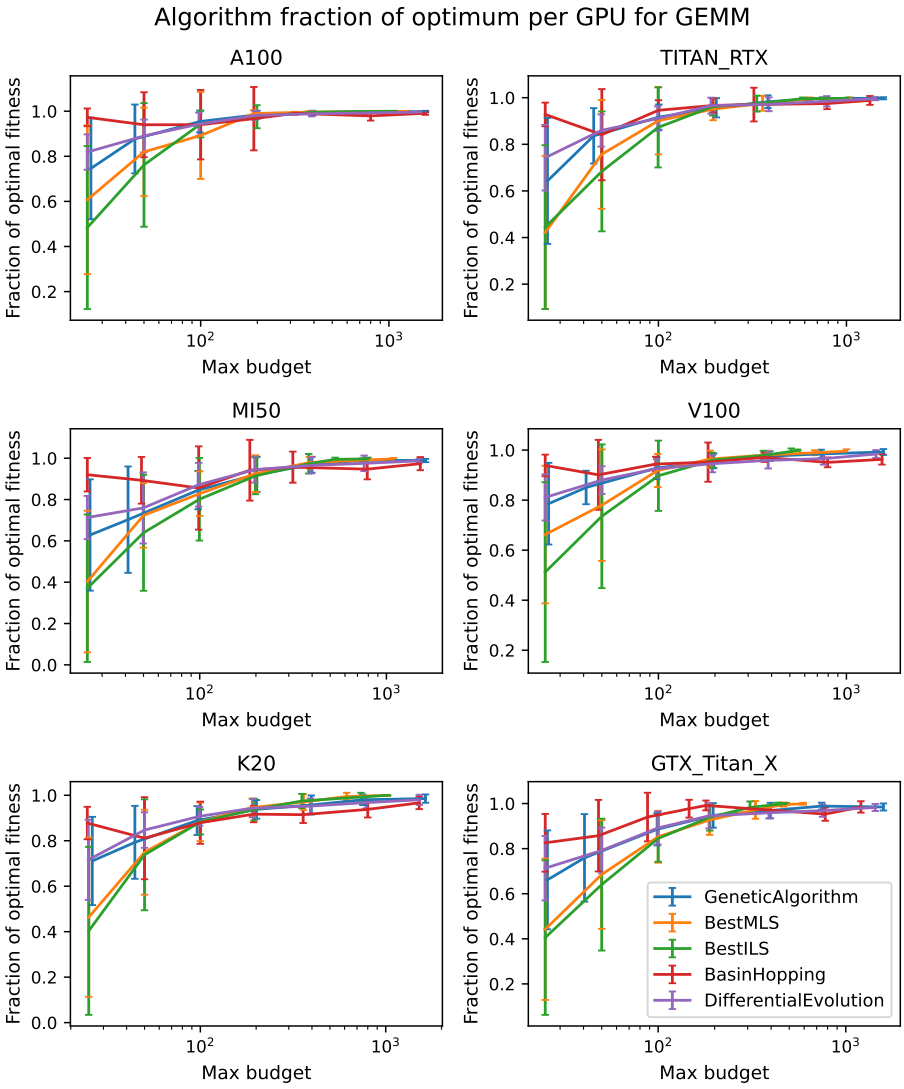


Figure A11: **GEMM:** Fraction of optimal runtime per GPU for GA, BestMLS, BestILS, basin hopping, and differential evolution over 50 runs. Each point is the mean fraction of optimal runtime found (y-axis) for mean budget used (logarithmic x-axis), with error bars indicating the standard deviation in fraction of optimum.

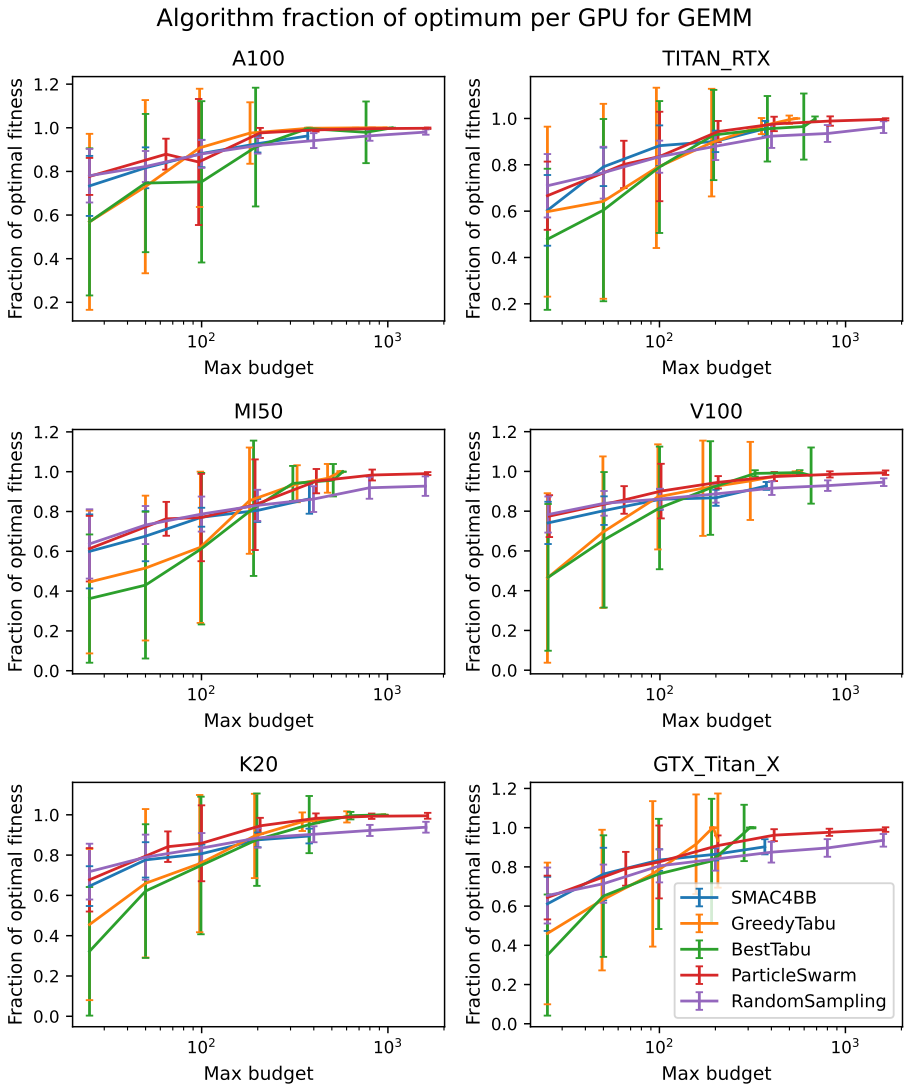


Figure A12: **GEMM:** Fraction of optimal runtime per GPU for SMAC, FirstTabu, BestTabu, PSO, and random sampling over 50 runs. Each point is the mean fraction of optimal runtime found (y-axis) for mean budget used (logarithmic x-axis), with error bars indicating the standard deviation in fraction of optimum.

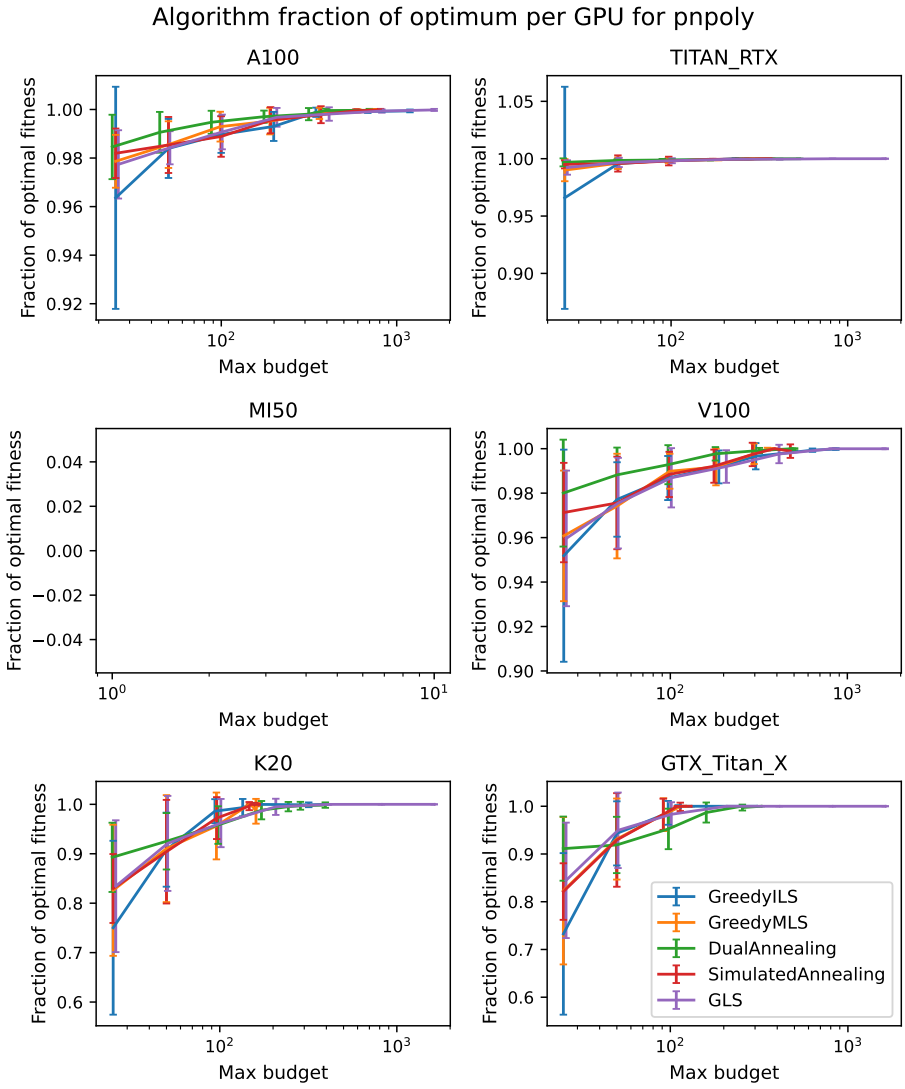


Figure A13: **Point-in-polygon:** Fraction of optimal runtime per GPU for FirstILS, FirstMLS, dual annealing, simulated annealing, and GLS over 50 runs. Each point is the mean fraction of optimal runtime found (y-axis) for mean budget used (logarithmic x-axis), with error bars indicating the standard deviation in fraction of optimum. The point-in-polygon kernel was not implemented for the MI50 GPU.

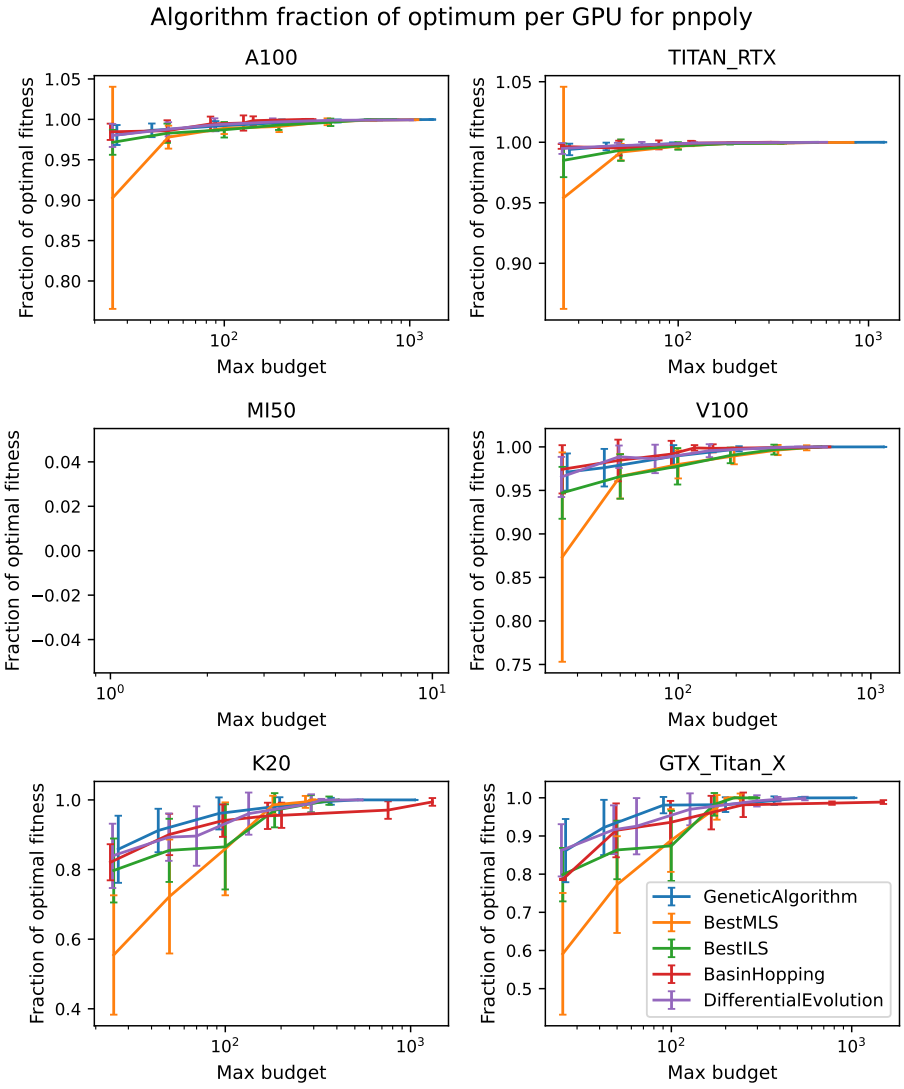


Figure A14: **Point-in-polygon:** Fraction of optimal runtime per GPU for GA, BestMLS, BestILS, basin hopping, and differential evolution over 50 runs. Each point is the mean fraction of optimal runtime found (y-axis) for mean budget used (logarithmic x-axis), with error bars indicating the standard deviation in fraction of optimum. The point-in-polygon kernel was not implemented for the MI50 GPU.

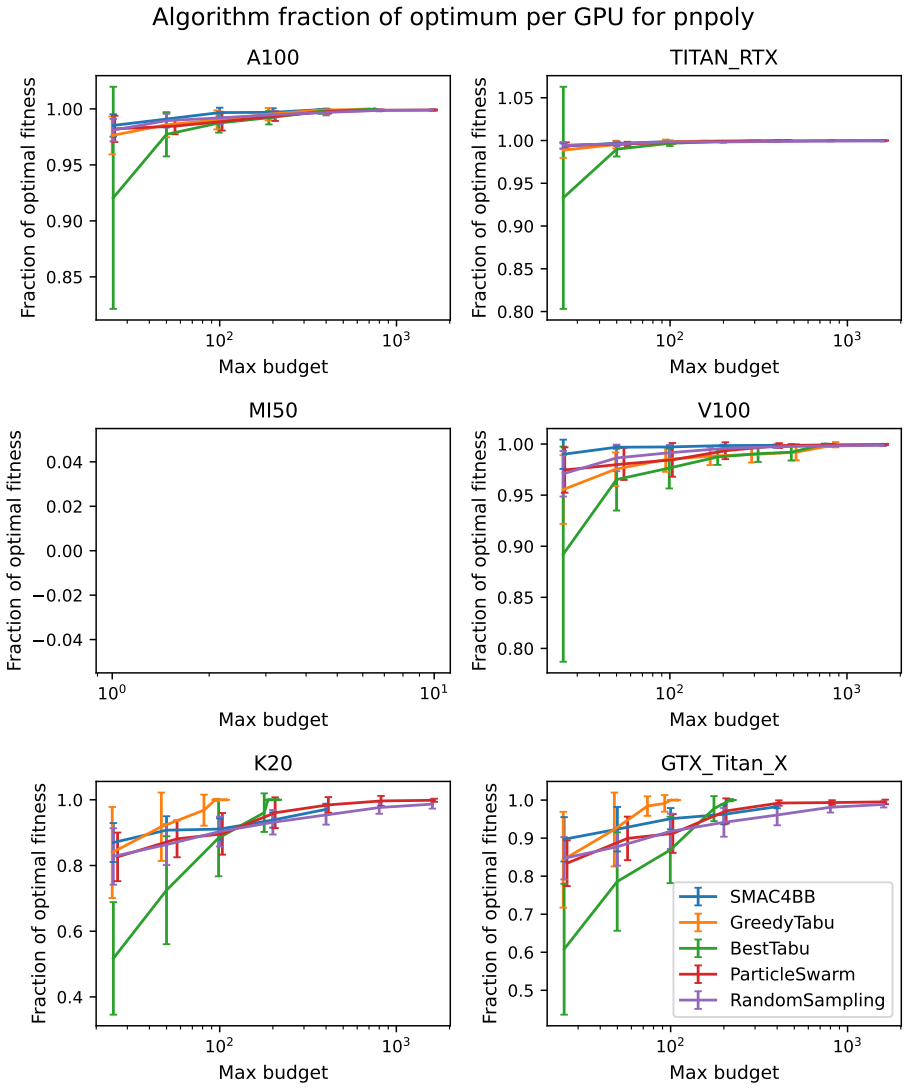


Figure A15: **Point-in-polygon:** Fraction of optimal runtime per GPU for SMAC, FirstTabu, BestTabu, PSO, and random sampling over 50 runs. Each point is the mean fraction of optimal runtime found (y-axis) for mean budget used (logarithmic x-axis), with error bars indicating the standard deviation in fraction of optimum. The point-in-polygon kernel was not implemented for the MI50 GPU.