Karlsruhe Institute of Technology

Online-Autotuning in the Presence of Algorithmic Choice

Philip Pfaffe, Martin Tillmann, Sigmar Walter, and Walter F. Tichy

KIT - Institute of Program Structures and Data Organization

Motivation

For a given task, there may be **multiple algorithms** available, each with its own set of tunable parameters.

Choice of optimal algorithm may depend on runtime context:

- Input
- Hardware
- System load

Autotune algorithmic choice **at runtime**

Motivation

For a given task, there may be **multiple algorithms** available, each with its own set of tunable parameters.

Choice of optimal algorithm may depend on runtime context:

- Input
- Hardware
- System load

Autotune algorithmic choice **at runtime**

Motivation

For a given task, there may be **multiple algorithms** available, each with its own set of tunable parameters.

Choice of optimal algorithm may depend on runtime context:

- Input
- Hardware
- System load

Autotune algorithmic choice **at runtime**

Autotuning – The Basics

KIT Karlsruhe Institute of Technology

Search space T_a for an algorithm a with **tuning parameters** $\tau_{a,j}$:

$$T_a = \tau_{a,0} \times \cdots \times \tau_{a,J}$$

A **configuration** $C_a \in T_a$ is measured by the timing function m_a. The context K describes external influences (hardware, input data).

$$C_{optimal,a} = \arg\min_{C_a} m_a(C_a, K)$$

Autotuning – The Basics

Search space T_a for an algorithm a with **tuning parameters** $\tau_{a,j}$:

$$T_a = \tau_{a,0} \times \cdots \times \tau_{a,J}$$

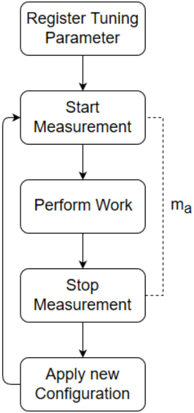
A **configuration** $C_a \in T_a$ is measured by the timing function m_a. The context K describes external influences (hardware, input data).

$$C_{optimal,a} = \underset{C_a}{\arg\min}\ m_a(C_a, K)$$

The Online-Autotuning Scenario

Online-Autotuning performs tuning at application runtime.

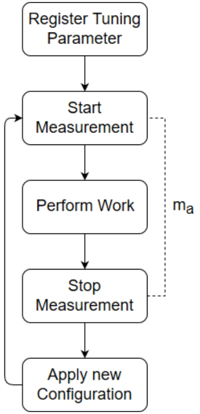
- Minimize overall application runtime.
- Minimize sum of tuning iterations $\sum_i m_a(C_i)$.
- Each evaluated configuration C_i has to be amortized.



The Online-Autotuning Scenario

Online-Autotuning performs tuning at application runtime.

- Minimize overall application runtime.
- Minimize sum of tuning iterations $\sum_i m_a(C_i)$.
- Each evaluated configuration C_i has to be amortized.

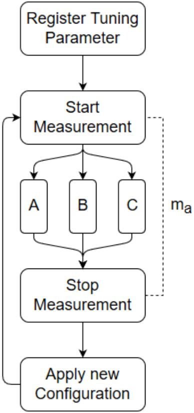


Algorithmic Choice

Choose algorithm **A**, **B** or **C** in the current context.

Algorithms have their own search spaces T_A, T_B and T_C.

Finding $C_{optimal,A}$, $C_{optimal,B}$ and $C_{optimal,C}$ before choosing the optimal algorithm is not feasible in an online scenario.

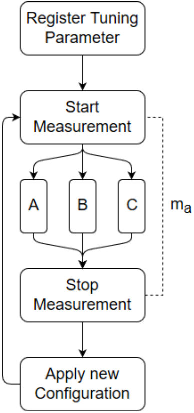


Algorithmic Choice

Choose algorithm **A**, **B** or **C** in the current context.

Algorithms have their own search spaces T_A, T_B and T_C.

Finding $C_{optimal,A}$, $C_{optimal,B}$ and $C_{optimal,C}$ before choosing the optimal algorithm is not feasible in an online scenario.

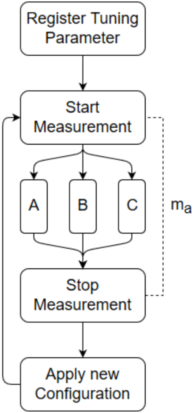


Algorithmic Choice

Choose algorithm **A**, **B** or **C** in the current context.

Algorithms have their own search spaces T_A, T_B and T_C.

Finding $C_{optimal,A}$, $C_{optimal,B}$ and $C_{optimal,C}$ before choosing the optimal algorithm is not feasible in an online scenario.



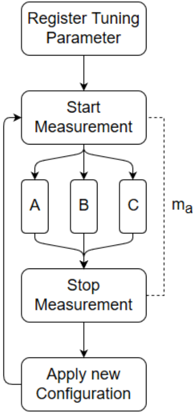
Algorithmic Choice

Autotune search spaces concurrently.

Exploit the only degree of freedom:

- Order of evaluation

Amortize each sampled configuration.
Choose near-optimal configurations, ignore bad configurations.



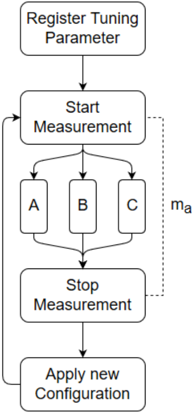
Algorithmic Choice

Autotune search spaces concurrently.

Exploit the only degree of freedom:

- Order of evaluation

Amortize each sampled configuration.
Choose near-optimal configurations, ignore bad configurations.



Algorithmic Choice

Tuning problem with algorithmic choice:

$$C_{opt} = \underset{A \in \mathcal{A}, C \in T_A}{\arg\min} \; m_A(C)$$

Evaluation in two phases:

1. Choose algorithm A
2. Perform tuning iteration on T_A

Have to manage state of all search spaces T_A.

Nominal Parameters

Algorithmic Choice introduces **nominal tuning parameters** into our scenario.

Class	*Distinguishing Property*	*Example*
Nominal	Labels	Choice of algorithm
Ordinal	Order	Choice of buffer sizes from a set `small`, `medium`, `large`
Interval	Distance	Percentage of a maximum buffer size
Ratio	Natural Zero, Equality of Ratios	Number of threads

Known tuning strategies that rely on a measure of **direction** or **distance** are not applicable for nominal parameters.

Algorithmic Choice – Strategies

Strategies for algorithmic choice:

- ϵ-Greedy
- Gradient Weighted
- Optimum Weighted
- Sliding Window Area-Under-The-Curve

ϵ-Greedy Strategy

The ϵ-Greedy strategy is a parameterized probabilistic method.

Probability	Action
$1 - \epsilon$	currently best performing algorithm
ϵ	random algorithm with uniform probability

Parameter ϵ controls the explorative behavior. We used 0.05, 0.1 and 0.2 as values.

Weighted Probabilistic Methods

Choose algorithm A with probability proportional to weight w_A. Weights are selected by the concrete strategy.

- Gradient Weighted
- Optimum Weighted
- Sliding Window Area-Under-The-Curve

The selection probability of algorithm A is then $P_A = \frac{w_A}{\sum_{A' \in \mathcal{A}} w_{A'}} > 0$.

Gradient Weighted Strategy

Choose algorithm A with probability proportional to weight w_A, based on the **gradient** G_A observed in the performance of the latest iteration window $[i_0, i_1]$.

$$G_A = \frac{\frac{1}{m_{A,i_1}} - \frac{1}{m_{A,i_0}}}{i_1 - i_0}$$

$$w_A = \begin{cases} G_A + 2 & \text{if } G_A \geq -1 \\ -\frac{1}{G_A} & \end{cases}$$

We use an iteration window of 16.

Optimum Weighted Strategy

Choose algorithm A with probability proportional to weight w_A, based on the **current optimal performance**.

$$w_A = \max_i \frac{1}{m_{A,i}}$$

Sliding Window Area-Under-The-Curve Strategy

The Sliding Window AUC strategy is again a probabilistic method, which assigns a weight w_A based on the **area under the algorithm's performance curve** within a sliding iteration window $[i_0, i_1]$.

$$w_A = \frac{\sum_{i=i_0}^{i_1} \frac{1}{m_{A,i}}}{i_1 - i_0}$$

We use a window size of 16.

Evaluation

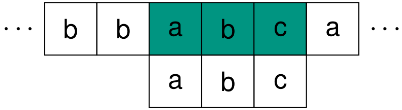
Two case studies:

- Parallel String Matching
  - Seven algorithms and one heuristic.
  - No tuning parameters besides algorithmic choice.
- Raytracing
  - Four data structures.
  - Tuning parameters for tree bounds and construction heuristics.

Parallel String Matching

Parallel versions of:

- Boyer-Moore
- Knuth-Morris-Pratt
- ShiftOr
- Hash3
- SSEF
- EBOM, FSBNDM
- Hybrid



Text corpora: bible and the human genome.

The query pattern and text are supplied at program invocation. Any precomputation is part of the algorithms runtime.

Parallel String Matching

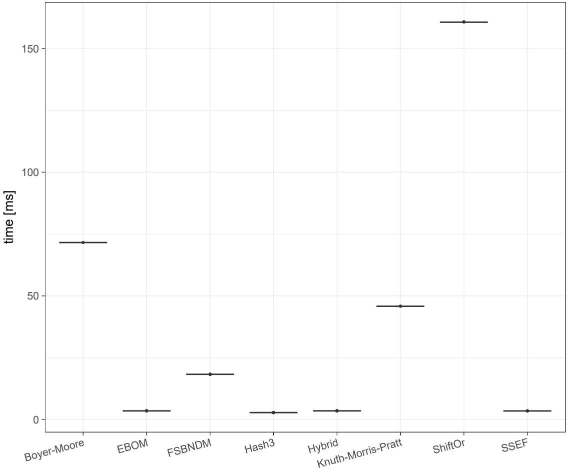


Figure: Performance of the parallel string matching algorithms

Parallel String Matching

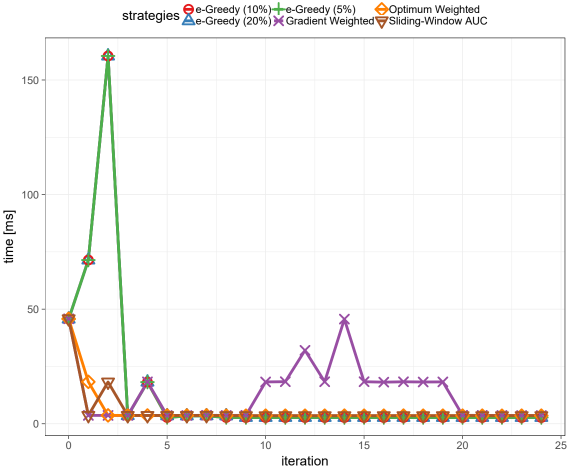


Figure: Median performance in individual iterations of all strategies

Parallel String Matching

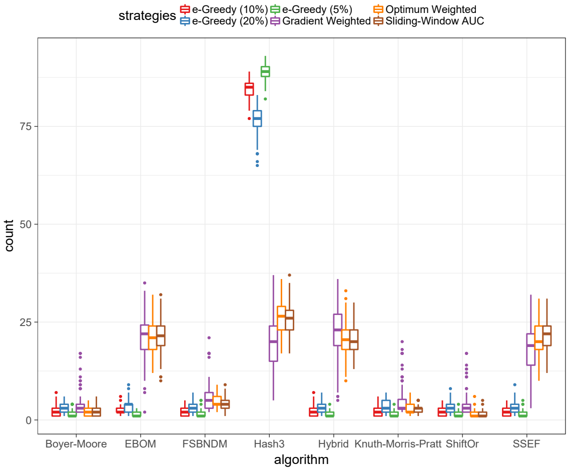
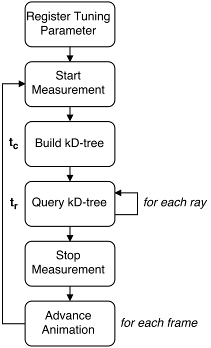


Figure: Frequency of all algorithms being chosen by the strategies

Raytracing

Two phase raytracing application.
Iterate over 100 frames:

1. Construct SAH kD-tree.
2. Cast rays, query kD-tree.



Four different datastructes:

- Inplace
- Wald-Havran
- Nested
- Lazy

Each datastructure has their own tuning space with three or four parameters.

Raytracing

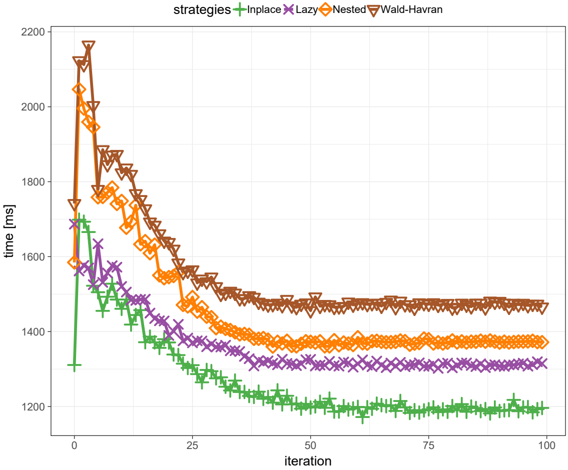


Figure: Tuning timeline of all four algorithms. The plot shows the average time taken in every iteration.

Raytracing

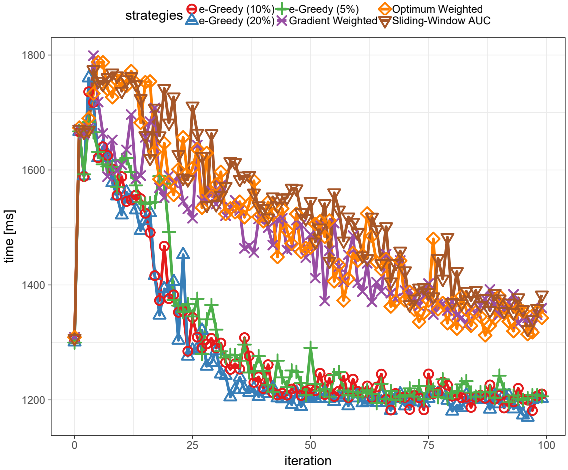


Figure: Median performance in individual iterations of all strategies

Raytracing

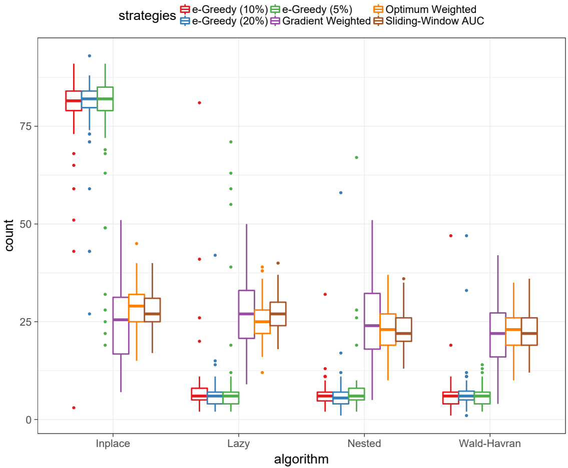


Figure: Frequency of all algorithms being chosen by the strategies

Conclusion

The ϵ-Greedy strategy is able to achieve the fastest convergence. The remaining strategies achieve convergence as well but at a slower rate.

Future work will generalize from the problem of algorithmic choice towards **arbitrary nominal parameters**. This requires combining the techniques presented here to achieve maximum convergence speed while defending against local extrema.

Online-Autotuning in the Presence of Algorithmic Choice

Thank you for your attention.

`https://code.ipd.kit.edu/pfaffe/libtuning`