What Does It Mean to Prefer the Fastest Algorithm?

Formalizing Preferences Over Runtime Distributions

Devon Graham

University of British Columbia

Kevin Leyton-Brown

University of British Columbia
Canada CIFAR AI Chair, Amii

Tim Roughgarden

Columbia University

Talk Overview

Comparing Runtime Distributions

Formalizing Our Preferences

Applying Our Framework

Estimating Expected Utility from Samples

Which algorithm do you prefer?

Consider a family of algorithms that always return the right answer but vary in their runtimes, like a set of SAT solvers. Much existing work assumes that the best algorithm is the one with the **best average runtime**. But is it really?

Which algorithm do you prefer?

Consider a family of algorithms that always return the right answer but vary in their runtimes, like a set of SAT solvers. Much existing work assumes that the best algorithm is the one with the **best average runtime**. But is it really?

Two Algorithms

Algorithm A_1

- Solves 99 instances in 1 second.
- Runs the 100th instance for 10 days but fails to solve it.

Algorithm A_2

- Runs all 100 instances for 10 days each but fails to solve any.

Problem 1

Average runtimes are completely unconstrained by this information. Does that mean that we don't know enough to have any preference between the algorithms?

Which algorithm do you prefer?

Consider a family of algorithms that always return the right answer but vary in their runtimes, like a set of SAT solvers. Much existing work assumes that the best algorithm is the one with the **best average runtime**. But is it really?

Two Algorithms

Algorithm A_1

- Solves 99 instances in 1 second.
- Runs the 100th instance for 10 days but fails to solve it.

Algorithm A_2

- Runs all 100 instances for 10 days each but fails to solve any.

Problem 2

Imagine that both algorithms contain a bug, and the long runs never terminate. Then both averages are infinite. In this case, are we indifferent between A_1 and A_2?

Handling Capped Runs

- We often appear able to **prefer one algorithm over another** even when some runs are stopped early ("capped")
- But how exactly should we account for such runs?
  - consider only the **fraction of problems solved**
  - consider capped runs to have **completed at the captime** (PAR1)
  - consider capped runs to have **completed at $k > 1$ times the captime** (PARk)
- Relative rankings between algorithms depend critically on choice of captime; k

Growing trend: think about algorithm designs as a hypothesis space

Machine learning

Classical approach

- Features based on **expert insight**
- Model family selected by **hand**
- **Manual** tuning of hyperparameters

Growing trend: think about algorithm designs as a hypothesis space

Machine learning

Classical approach

- Features based on expert insight
- Model family selected by hand
- Manual tuning of hyperparameters

Deep learning

- Very **highly parameterized** models, using expert knowledge to identify appropriate invariances and model biases (e.g., convolutional structure)
- “deep”: **many layers** of nodes, each depending on the last
- Use **lots of data** (plus e.g. dropout regularization) to avoid overfitting
- **Computationally intensive search** replaces human design

Growing trend: think about algorithm designs as a hypothesis space

Machine learning

Classical approach

- Features based on expert insight
- Model family selected by hand
- Manual tuning of hyperparameters

Deep learning

- Very highly parameterized models, using expert knowledge to identify appropriate invariances and model biases (e.g., convolutional structure)
- “deep”: many layers of nodes, each depending on the last
- Use lots of data (plus e.g. dropout regularization) to avoid overfitting
- Computationally intensive search replaces human design

Discrete Optimization

Classical approach

- **Expert designs** a heuristic algorithm
- Iteratively conducts **small experiments** to improve the design

Growing trend: think about algorithm designs as a hypothesis space

Machine learning

Classical approach

- Features based on expert insight
- Model family selected by hand
- Manual tuning of hyperparameters

Deep learning

- Very highly parameterized models, using expert knowledge to identify appropriate invariances and model biases (e.g., convolutional structure)
- "deep": many layers of nodes, each depending on the last
- Use lots of data (plus e.g. dropout regularization) to avoid overfitting
- Computationally intensive search replaces human design

Discrete Optimization

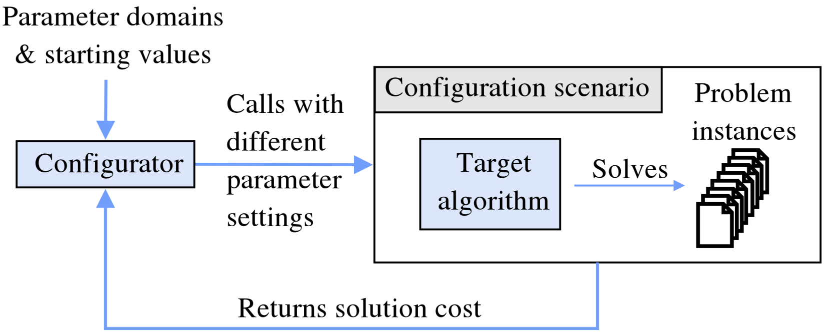
Classical approach

- Expert designs a heuristic algorithm
- Iteratively conducts small experiments to improve the design

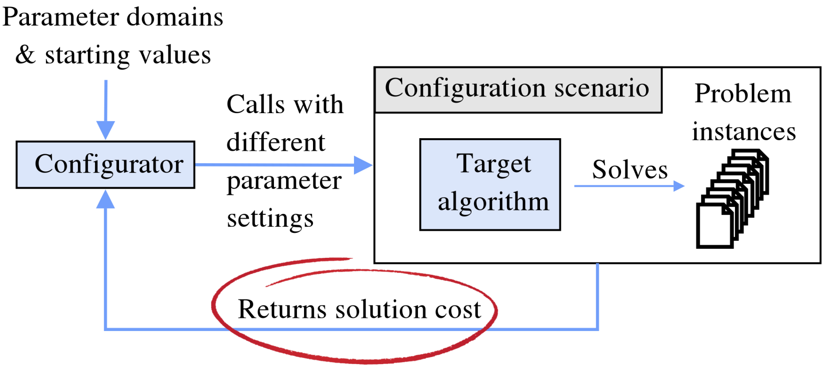
Learning in the space of algorithm designs

- Very **highly parameterized** algorithms express a combinatorial space of heuristic design choices that make sense to an expert
- "deep": **many layers** of parameters, each depending on the last
- Use **lots of data** to characterize the distribution of interest
- **Computationally intensive search** replaces human design

Algorithm Configuration Visualized



Algorithm Configuration Visualized



Talk Overview

Comparing Runtime Distributions

Formalizing Our Preferences

Applying Our Framework

Estimating Expected Utility from Samples

Setup

Definitions:

- $\mathcal{A}$: a set of **algorithms**
- $A \in \mathcal{A}$: the **distribution over possible runtimes** of the corresponding algorithm
  - accounts for both instance distribution and random seeds
- t_A : a **runtime** sampled from A
  - $t_A \in [0, \infty]$: runtimes are non-negative and may be infinite
  - $A = \delta_t$: runs always take t
- K : a **distribution over possible captimes**
  - sometimes we're not sure how long we'll have before we need to stop a run
- κ : a **captime** sampled from K
  - $K = \delta_\kappa$: captime is fixed and known

Axiomatization

The Axiomatic Method

Reason from **observations about our preferences** in easily understood scenarios to **derive a general rule** that describes our preferences in all scenarios.

Our Axiomatization

- Draws on von Neumann & Morgenstern's **expected utility** derivation
  - One axiom needs nontrivial adaptation to capture our domain, though we preserve the spirit
- Additional, **runtime-specific axioms**
  - "Faster is better"
  - "We care about solving our problem"

Classical Axioms: Transitivity, Monotonicity, and Continuity

These axioms **very closely mirror the classic VNM setup** *(e.g., K does not change anything here)*

Axiom: Transitivity ***"preferences are acyclic"***

If $A_1 \succeq_K A_2$ and $A_2 \succeq_K A_3$, then $A_1 \succeq_K A_3$.

Classical Axioms: Transitivity, Monotonicity, and Continuity

These axioms **very closely mirror the classic VNM setup** *(e.g., K does not change anything here)*

Axiom: Transitivity *"preferences are acyclic"*

If $A_1 \succeq_K A_2$ and $A_2 \succeq_K A_3$, then $A_1 \succeq_K A_3$.

An operation

Mixing: an operation on distributions A_1 and A_2

The *mixture distribution* $\left[p : A_1, (1-p) : A_2\right]$ returns a runtime sampled from A_1 with probability p and from from A_2 with probability $1-p$.

Classical Axioms: Transitivity, Monotonicity, and Continuity

These axioms **very closely mirror the classic VNM setup** *(e.g., K does not change anything here)*

Axiom: Transitivity *"preferences are acyclic"*

If $A_1 \succeq_K A_2$ and $A_2 \succeq_K A_3$, then $A_1 \succeq_K A_3$.

An operation

Mixing: an operation on distributions A_1 and A_2

The *mixture distribution* $\left[p : A_1, (1-p) : A_2\right]$ returns a runtime sampled from A_1 with probability p and from from A_2 with probability $1-p$.

Axiom: Monotonicity *"we prefer mixtures featuring more of a good thing"*

If $A_1 \succeq_K A_2$ then for any $p, q \in [0, 1]$ we have

$[p : A_1, (1-p) : A_2] \succeq_K [q : A_1, (1-q) : A_2]$ if and only if $p \geq q$.

Classical Axioms: Transitivity, Monotonicity, and Continuity

These axioms **very closely mirror the classic VNM setup** *(e.g., K does not change anything here)*

Axiom: Transitivity *"preferences are acyclic"*

If $A_1 \succeq_K A_2$ and $A_2 \succeq_K A_3$, then $A_1 \succeq_K A_3$.

An operation

Mixing: an operation on distributions A_1 and A_2

The *mixture distribution* $\left[p : A_1, (1-p) : A_2\right]$ returns a runtime sampled from A_1 with probability p and from from A_2 with probability $1-p$.

Axiom: Monotonicity *"we prefer mixtures featuring more of a good thing"*

If $A_1 \succeq_K A_2$ then for any $p, q \in [0, 1]$ we have
$[p : A_1, (1-p) : A_2] \succeq_K [q : A_1, (1-q) : A_2]$ if and only if $p \geq q$.

Axiom: Continuity *"there always exists an indifference point"*

If $A_1 \succeq_K A_2 \succeq_K A_3$, then there exists $p \in [0, 1]$ such that $A_2 \simeq_K [p : A_1, (1-p) : A_3]$.

Independence Axiom

If we're indifferent between each of a set of outcome pairs, we're also indifferent between mixtures that equally weight respective elements of each pair.

Another operation

Compounding: an operation on collection of distributions $M(t, \kappa)$

The **compound distribution**: $\left[M(t,\kappa) \mid t \sim A, \kappa \sim K\right]$ first samples t from A and κ from K, then returns a runtime sampled from some given $M(t, \kappa)$.

Axiom: independence ***"pointwise indifference extends to distributions"***

If $\delta_t \simeq_{\delta_\kappa} M(t, \kappa)$ for all t, κ, then $A \simeq_K [M(t,\kappa) \mid t \sim A, \kappa \sim K]$.

If, for every t and κ, we are indifferent *(given that we face captime κ)* between obtaining runtime t with certainty and sampling a runtime from some distribution $M(t, \kappa)$...

then we are also indifferent *(given that we face a captime sampled from K)* between the runtime distribution of algorithm A and the runtime distribution corresponding to sampling t from A and κ from K and then sampling a runtime from $M(t, \kappa)$.

Eagerness and Relevance Axioms

Our first four[1] axioms **imply the existence of a utility function.**

Two more runtime-specific axioms **further constrain this utility function**.

Axiom: Eagerness ***"faster is better"***

For any $t \leq t'$, if A's entire support is contained in $[t, t']$, then $\delta_t \succeq_K A \succeq_K \delta_{t'}$ for all K.

Axiom: Relevance ***"we strictly prefer to solve our problem"***

$\delta_t \succ_{\delta_\kappa} \delta_{t'}$ for all κ and $t < \kappa \leq t'$.

[1] We don't need a "decomposability" axiom because our base outcome space consists of probability distributions rather than discrete events.

Main Result

Theorem.

If our preferences follow the six axioms, then for any runtime distributions A_1 and A_2 and any captime distribution K, **there exists[2] a function $u : \mathbb{R}^2 \to [0,1]$** such that

$$A_1 \succeq_K A_2 \iff \mathbb{E}_{t\sim A_1}\Big[\mathbb{E}_{\kappa\sim K}[u(t,\kappa)]\Big] \geq \mathbb{E}_{t\sim A_2}\Big[\mathbb{E}_{\kappa\sim K}[u(t,\kappa)]\Big].$$

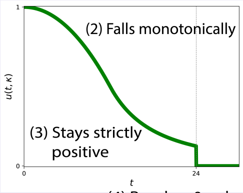
Furthermore, u has the following properties:

(1) $u(0,\kappa) = 1$;

(2) $u(\cdot,\kappa)$ is (weakly) monotone decreasing;

(3) $u(t,\kappa) > 0$, for all $t < \kappa$;

(4) $u(\kappa, t') = 0$, for all $t' \geq \kappa$.



[2] We can also show that the only such functions are positive affine transforms of the u mentioned: $u' = au + b$, $a > 0$.

Proof Sketch Part 1: Existence of u

Continuity: for every non-negative t, κ, there exists a value $p(t, \kappa) \in [0, 1]$ as defined below.

A key definition

Function $p(t, \kappa)$

$p(t, \kappa)$ is a value that **makes us indifferent between** solving at time t and a gamble between solving at time 0 (**the best outcome**) or timing out at κ (**the worst outcome**):

If $t < \kappa$, then $p(t, \kappa)$ is the value that satisfies: $\delta_t \simeq_{\delta_\kappa} \left[p(t, \kappa) : \delta_0, (1 - p(t, \kappa)) : \delta_\kappa\right]$

If $t \geq \kappa$, then $p(t, \kappa) = 0$.

We can use the expectation of p as the score for any algorithm A:

- Construct distribution $A' = \left[[p(t, \kappa) : \delta_0 \,,\, (1 - p(t, \kappa)) : \delta_\kappa] \mid t \sim A, \kappa \sim K\right]$
- A' returns 0 with probability $\mathbb{E}_{t \sim A, \kappa \sim K}[p(t, \kappa)]$, otherwise returns $\kappa \sim K$.
- independence: $A_1 \simeq_K A_1'$; similarly, $A_2 \simeq_K A_2'$.

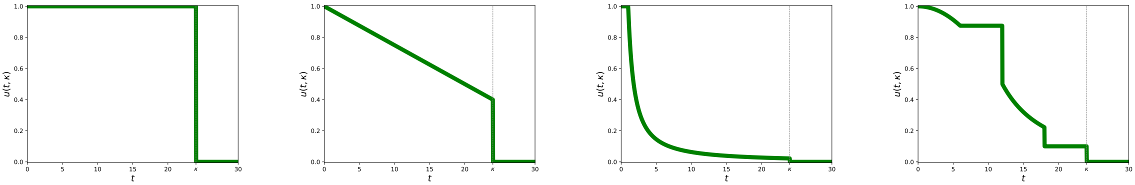
  (Consider $M(t, \kappa) = [p(t, \kappa) : \delta_0 \,,\, (1 - p(t, \kappa)) : \delta_\kappa]$. Note p was defined so that $\delta_t \simeq_{\delta_\kappa} M(t, \kappa)$.)
- Monotonicity: $A_1' \succeq_K A_2' \iff \mathbb{E}_{t \sim A_1, \kappa \sim K}[p(t, \kappa)] \geq \mathbb{E}_{t \sim A_2, \kappa \sim K}[p(t, \kappa)]$
- Transitivity: $A_1 \succeq_K A_2 \iff A_1' \succeq_K A_2'$

A' in words:
Sample t from A, sample κ from K, return a runtime sampled from $[p(t, \kappa) : \delta_0,$ $1 - p(t, \kappa) : \delta_\kappa]$

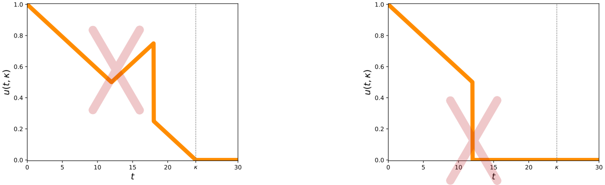
Proof Sketch Part 2: Structure of u

(1) $u(0, \kappa) = 1$ (by Eagerness, Monotonicity)
(2) $u(t, \kappa) \geq u(t', \kappa)$ for any $t \leq t'$ (by Eagerness)
(3) $u(t, \kappa) > 0$, for all $t < \kappa$ (by Relevance)
(4) $u(\kappa, t') = 0$, for all $t' \geq \kappa$ (by definition of p)

u can look like this ...



... but not like this.



Talk Overview

Comparing Runtime Distributions

Formalizing Our Preferences

Applying Our Framework

Estimating Expected Utility from Samples

Concrete Example 1

Example (step-function utility, known captime)

We must find a solution to our integer program by midnight or it will be useless.

- Step-function utility *(i.e., $u(t,\kappa) = 1$ for all $t < \kappa$)*
- Fixed and known captime κ *(i.e., $K = \delta_\kappa$)*

$\Longrightarrow$ maximize $\Pr_{t \sim A}\left(t \leq \kappa\right)$

"The best algorithm is the one most likely to finish before the captime."

"Which algorithm is better?"

Algorithm A_1

- Solves 99 instances in 1 second.
- Runs the 100th instance for 10 days but fails to solve it.

Algorithm A_2

- Runs all 100 instances for 10 days each but fails to solve any.

A_1 preferred to A_2 for any $\kappa < 10$ days.

(unknown for $\kappa \geq 10$ days: maybe A_2 will solve all problems and A_1 won't)

Concrete Example 2

Example (linear value for money; pay for compute)

We are risk neutral and have a linear value for money.

We face no explicit runtime cap but we have to buy our compute on Amazon EC2.

- Solving the problem is worth v
- Each hour of compute costs α
- We can avoid negative payoffs by setting captime $\kappa^* = v/\alpha$

$\Longrightarrow$ maximize $\mathbb{E}_t \max(v - \alpha t, 0)$

"The best algorithm costs least on average."

"Which algorithm is better?"

Algorithm A_1

- Solves 99 instances in 1 second.
- Runs the 100th instance for 10 days but fails to solve it.

Algorithm A_2

- Runs all 100 instances for 10 days each but fails to solve any.

A_1 preferred to A_2 if $v/\alpha < 10 \cdot 24$ hours.

Step-Function Utility, Uncertain Captime

Now let's consider the case of **uncertain captime**

- One intuitive setting where **expected utility decreases with time**:
  - utility is a step function (I just want to solve the problem)
  - I'm uncertain about exactly **when I'll need to stop the run**

Step-function utilities

- $\mathbb{E}_{\kappa \sim K}[u(t, \kappa)] = \Pr_{\kappa \sim K}(t < \kappa)$: i.e., 1 minus the CDF of K
- We'll denote this as $u(t)$ when K is implicit from context

Concrete Example 3

Example (step-function utility, unknown captime)

Our client will demand an answer at some point in the future, described by K

- Step-function utility *(i.e., $u(t, \kappa) = 1$ for all $t < \kappa$)*
- **Unknown captime** ($\kappa \sim K$)

$\Longrightarrow$

maximize $\mathbb{E}_{t \sim A}\left[\Pr_{\kappa \sim K}(t \leq \kappa)\right]$

"The best algorithm is the one most likely to finish before the captime, **in expectation** over captimes."

"Which algorithm is better?"

Algorithm A_1

- Solves 99 instances in 1 second.
- Runs the 100th instance for 10 days but fails to solve it.

Algorithm A_2

- Runs all 100 instances for 10 days each but fails to solve any.

A_1 preferred to A_2 if κ is always at least 1 second and there is even just a 1% chance that κ will be less than 10 days.

Constrained K: The Method of Maximum Entropy

What if we know only **constraints on the distribution K**?

Principle of Maximum Entropy

If we do not know which distribution to use among some set of alternatives, we should **use the one having greatest entropy**, since it is the least informative and thus incorporates no extraneous assumptions.

Maximizing entropy “spreads out” the distribution’s probability mass as much as the conditions allow

Example (Bounded interval)

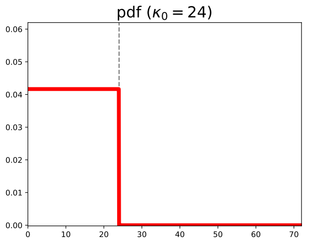
If all we know is that the distribution has support on some bounded interval, the uniform distribution has maximum entropy because it is “flattest.”

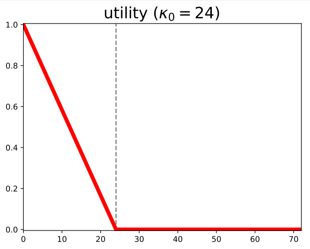
Choosing K: Some examples

Example (Bounded time interval)

Our client will need a solution to their SAT problem sometime in the next 24 hours.

The maximum entropy distribution is uniform:





$$\text{maximize } \mathbb{E}_{t\sim A}\left[u(t)\right] \quad \text{where} \quad u(t) = \begin{cases} 1 - \frac{t}{24h} & \text{if } t < 24h \\ 0 & \text{otherwise} \end{cases}$$

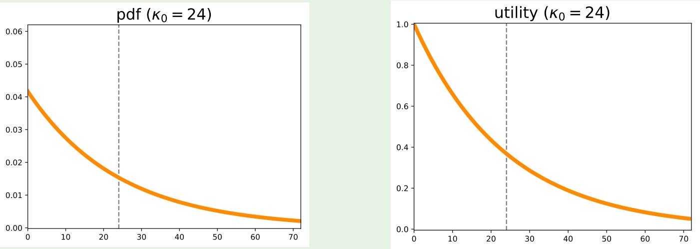
(linear utility function)

Choosing K: Some examples

Example (Known mean)

We do not know how long the client will give us to solve their SAT problem but the expected value of the captime distribution is 24 hours.

The maximum entropy distribution is exponential:



$$\text{maximize} \underset{t \sim A}{\mathbb{E}} \left[e^{-\frac{t}{24h}} \right]$$

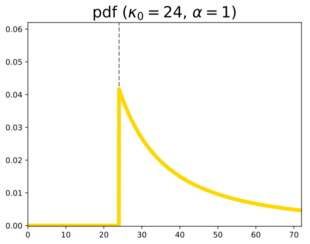
(exponential utility function)

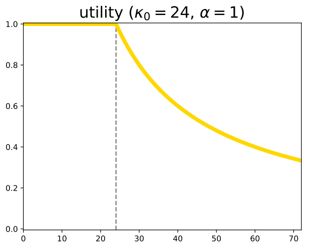
Choosing K: Some examples

Example (Known expected order of magnitude)

The client will certainly give us at least one day and on the order of d days (i.e., the expected value of the log captime will be $\log d$).

The maximum entropy distribution is Pareto with shape parameter $\alpha = (\ln d - \ln 24)^{-1}$:





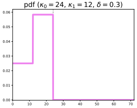
$$\text{maximize } \mathbb{E}_{t\sim A}\left[u(t)\right] \quad \text{where} \quad u(t) = \begin{cases} 1 & \text{if } t < 24h \\ \left(\frac{24h}{t}\right)^{\alpha} & \text{otherwise} \end{cases}$$

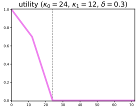
(geometric utility function)

And More...

κ is less than κ_0 with certainty, and less than κ_1 with probability δ (piecewise uniform distribution)

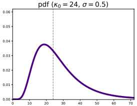
$$u(t) = \begin{cases} 1 - \frac{\delta t}{\kappa_1} & \text{if } t \le \kappa_1 \\ \frac{(1-\delta)(\kappa_0 - t)}{\kappa_0 - \kappa_1} & \text{if } \kappa_1 < t < \kappa_0 \\ 0 & \text{otherwise} \end{cases}$$

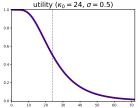




$\log \kappa$ has mean $\log \kappa_0$ and variance σ^2 (log-normal distribution)

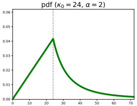
$$u(t) = \tfrac{1}{2} - \tfrac{1}{2}\,\mathrm{erf}\left(\frac{\log(t/\kappa_0)}{\sqrt{2}\sigma}\right)$$

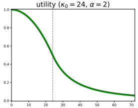




$\log \kappa$ has mean $\log \kappa_0$, and mean absolute deviation $1/\alpha$ (log-Laplace distribution)

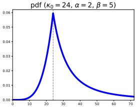
$$u(t) = \begin{cases} 1 - \frac{1}{2}\left(\frac{t}{\kappa_0}\right)^{\alpha} & \text{if } t < \kappa_0 \\ \frac{1}{2}\left(\frac{\kappa_0}{t}\right)^{\alpha} & \text{otherwise} \end{cases}$$

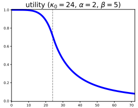




$\log \kappa$ has mean $\log \kappa_0$, and asymmetric absolute deviation (generalized log-Laplace distribution)

$$u(t) = \begin{cases} 1 - \frac{\alpha}{\alpha+\beta}\left(\frac{t}{\kappa_0}\right)^{\beta} & \text{if } t < \kappa_0 \\ \frac{\beta}{\alpha+\beta}\left(\frac{\kappa_0}{t}\right)^{\alpha} & \text{otherwise} \end{cases}$$





Concrete Example 4

Example (step-function utility, known order of magnitude)

Our client will give us at least 1 day and on the order of days $(\kappa \geq 1d,\ \mathbb{E}_{\kappa\sim K}[\log(\frac{\kappa}{1d})] = 1)$

- Step-function utility *(i.e., $u(t,\kappa) = 1$ for all $t < \kappa$)*
- Captime measured in days

$\Longrightarrow$

maximize $\mathbb{E}_{t\sim A}\left[u(t)\right]$ where

$$u(t) = \begin{cases} 1 & \text{if } t < 1d \\ \frac{1}{t} & \text{otherwise} \end{cases}$$

(Pareto)

"Which algorithm is better?"

Algorithm A_1

- Solves 99 instances in 1 second.
- Runs the 100th instance for 10 days but fails to solve it.

Algorithm A_2

- Runs all 100 instances for 10 days each but fails to solve any.

A_1 is at least 10 times better than A_2 (expected utility of $\geq \frac{99}{100}$ vs. $\leq \frac{1}{10}$)

Concrete Example 5

Example (step-function utility, symmetric order of magnitude)

Our unreliable client says they will give us on the order of days; we consider them equally likely to impose a captime above and below one day $\left(\mathbb{E}_{\kappa\sim K}\left[\left|\log\left(\frac{\kappa}{1d}\right)\right|\right] = 1\right)$

- Step-function utility *(i.e., $u(t,\kappa) = 1$ for all $t < \kappa$)*
- Captime measured in days
- Require smoothness at $t = 1d$

$\Longrightarrow$ maximize $\mathbb{E}_{t\sim A}\left[u(t)\right]$ where

$$u(t) = \begin{cases} 1 - \frac{1}{2}t & \text{if } t < 1d \\ \frac{1}{2}\left(\frac{1}{t}\right) & \text{otherwise} \end{cases}$$

(Log Laplace)

"Which algorithm is better?"

Algorithm A_1

- Solves 99 instances in 1 second.
- Runs the 100th instance for 10 days but fails to solve it.

Algorithm A_2

- Runs all 100 instances for 10 days each but fails to solve any.

A_1 is at least 20 times better than A_2 (expected utility of $\geq \frac{99}{100}$ vs. $\leq \frac{1}{20}$)

Talk Overview

Comparing Runtime Distributions

Formalizing Our Preferences

Applying Our Framework

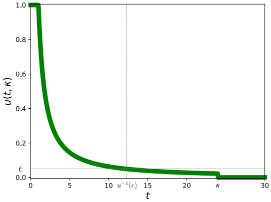
Estimating Expected Utility from Samples

Estimating A's expected utility from samples

- In reality we only get to estimate A from **samples** $\{t_1, \ldots, t_n\}$
  - Law of large numbers: $\frac{1}{n}\sum_{i=1}^{n} u(t_i) \to \mathbb{E}[u(A)]$ as $n \to \infty$.
  - But we only observe **capped runtimes** $\hat{t}_i = \min\{t_i, \kappa\}$

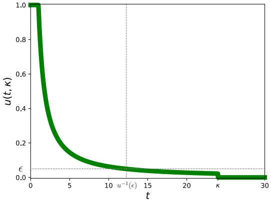
Estimating A's expected utility from samples

- In reality we only get to estimate A from **samples** $\{t_1, \ldots, t_n\}$
  - Law of large numbers: $\frac{1}{n}\sum_{i=1}^{n} u(t_i) \to \mathbb{E}[u(A)]$ as $n \to \infty$.
  - But we only observe **capped runtimes** $\hat{t}_i = \min\{t_i, \kappa\}$
- assume u is bounded between 0 and 1
- recall that u is monotone decreasing in t
- $u^{-1}(\epsilon)$ is that t for which $u(t)$ first falls to a value $\leq \epsilon$



Estimating A's expected utility from samples

- In reality we only get to estimate A from **samples** $\{t_1, \ldots, t_n\}$
  - Law of large numbers: $\frac{1}{n}\sum_{i=1}^{n} u(t_i) \to \mathbb{E}[u(A)]$ as $n \to \infty$.
  - But we only observe **capped runtimes** $\hat{t}_i = \min\{t_i, \kappa\}$
- assume u is bounded between 0 and 1
- recall that u is monotone decreasing in t
- $u^{-1}(\epsilon)$ is that t for which $u(t)$ first falls to a value $\leq \epsilon$



Theorem

We can ***ϵ-estimate $\mathbb{E}_{t\sim A}[u(t)]$ from capped samples*** *if and only if we sample with captime $\kappa \geq u^{-1}(\epsilon)$. Worst case time cost less than $u^{-1}(\epsilon/2) \cdot \frac{\ln(2/\delta)}{2}\left(\frac{2-\epsilon}{\epsilon}\right)^2$.*

- Worst-case runtime cost thus depends only on u, ϵ, and δ, **not on A's runtime distribution**.

Conclusion

- It's nontrivial to formalize why we **prefer one runtime distribution to another**, particularly in the presence of capping
  - Getting this right is important if we're going to **learn special-purpose algorithms for given datasets**, e.g. via algorithm configuration
- We present a **utility-theoretic answer** to this question based on axiomatic assumptions about preferences over runtime distributions
  - The result depends on the way utility decreases with time and on the captime distribution
- We describe a **maximum-entropy approach to modeling captime distributions** under various realistic constraints
- We show that A's expected utility can be **approximately estimated from samples** in time that does not depend on the captime distribution
- Key **ongoing work**: establishing that A_1 is at least ϵ-better than A_2 with probability $1 - \delta$