

Online Dynamic Algorithm Portfolios

Matteo Gagliolo
Supervisor: Jürgen Schmidhuber

IDSIA, Lugano, Switzerland

University of Lugano, Faculty of Informatics, Switzerland

24th March 2010, Lugano, Switzerland

Why algorithm portfolios?

The issue:

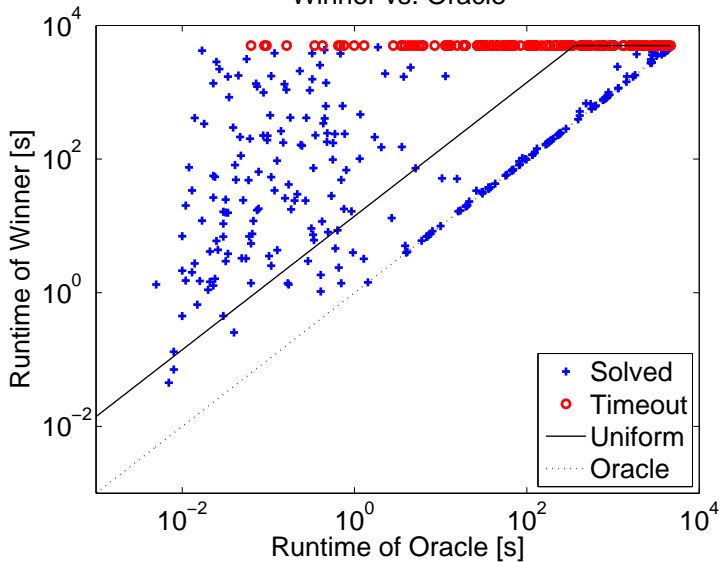
- ▶ Many important problems are **hard**
- ▶ Many alternative algorithms exist
- ▶ Often **no single best**

The idea: exploit performance diversity combining the execution of several algorithms.

The aim: **speeding up** problem solving.

Example: SAT competition

Winner vs. Oracle



Roadmap

Introduction

Algorithm Portfolios

Dynamic Allocation

Censored Sampling

Online Learning

Experiments

Conclusions

Hypotheses

A set of **generalized Las Vegas** algorithms $\mathcal{A} = \{a_1, a_2, \dots, a_N\}$

- ▶ Runtime is a random variable $T \in [0, \infty]$
- ▶ Runtime $T < \infty$ for at least one algorithm

A set of problem instances $\mathcal{B} = \{b_1, b_2, \dots, b_M\}$

- ▶ **search/decision** problems (SAT, CSP, ...)
- ▶ search version of **optimization** problems (find a solution of a given quality)
- ▶ combinatorial optimization (find global optimum, prove optimality)

In all these cases: **performance = runtime**

We do **not** consider: optimization problems in general (quality vs. time trade-off)

Algorithm Selection

- ▶ Pick “training” set \mathcal{B}_{train} of problem instances
- ▶ solve **each** $b \in \mathcal{B}_{train}$ with **each** $a \in \mathcal{A}$, storing $t(b, a)$
- ▶ learn a model of performance $(b, a) \rightarrow t(b, a)$

- ▶ solve each subsequent b with predicted fastest a

(Rice '76, Leyton-Brown et al. '02, ...)

Algorithm Selection

Offline learning:

- ▶ Pick “training” set \mathcal{B}_{train} of problem instances
- ▶ solve **each** $b \in \mathcal{B}_{train}$ with **each** $a \in \mathcal{A}$, storing $t(b, a)$
- ▶ learn a model of performance $(b, a) \rightarrow t(b, a)$

- ▶ solve each subsequent b with predicted fastest a

(Rice '76, Leyton-Brown et al. '02, ...)

Algorithm Selection

Offline learning:

- ▶ Pick “training” set \mathcal{B}_{train} of problem instances
- ▶ solve **each** $b \in \mathcal{B}_{train}$ with **each** $a \in \mathcal{A}$, storing $t(b, a)$
- ▶ learn a model of performance $(b, a) \rightarrow t(b, a)$

Static selection:

- ▶ solve each subsequent b with predicted fastest a

(Rice '76, Leyton-Brown et al. '02, ...)

Algorithm Selection

Offline learning:

- ▶ Pick “training” set \mathcal{B}_{train} of problem instances

How do I pick the size?

- ▶ solve **each** $b \in \mathcal{B}_{train}$ with **each** $a \in \mathcal{A}$, storing $t(b, a)$
- ▶ learn a model of performance $(b, a) \rightarrow t(b, a)$

Static selection:

- ▶ solve each subsequent b with predicted fastest a

(Rice '76, Leyton-Brown et al. '02, ...)

Algorithm Selection

Offline learning:

- ▶ Pick “training” set \mathcal{B}_{train} of problem instances

How do I pick the size?

- ▶ solve **each** $b \in \mathcal{B}_{train}$ with **each** $a \in \mathcal{A}$, storing $t(b, a)$

Do I have to solve each training problem multiple times?

- ▶ learn a model of performance $(b, a) \rightarrow t(b, a)$

Static selection:

- ▶ solve each subsequent b with predicted fastest a

(Rice '76, Leyton-Brown et al. '02, ...)

Algorithm Selection

Offline learning:

- ▶ Pick “training” set \mathcal{B}_{train} of problem instances

How do I pick the size?

- ▶ solve **each** $b \in \mathcal{B}_{train}$ with **each** $a \in \mathcal{A}$, storing $t(b, a)$

Do I have to solve each training problem multiple times?

- ▶ learn a model of performance $(b, a) \rightarrow t(b, a)$

Static selection:

- ▶ solve each subsequent b with predicted fastest a

Can I predict performance before start?

(Rice '76, Leyton-Brown et al. '02, ...)

Offline Static AS

- ▶ Pick training set \mathcal{B}_{train}

How do I pick the size?

- ▶ solve **each** $b \in \mathcal{B}_{train}$ with **each** $a \in A$, storing $t(b, a)$

Do I have to solve each training problem multiple times?

- ▶ learn a model $(b, a) \rightarrow t(b, a)$
- ▶ solve each subsequent b with corresponding “best” a

Can I predict performance before start?

Offline Static AS

- ▶ Pick training set \mathcal{B}_{train}

How do I pick the size?

Online learning

- ▶ solve **each** $b \in \mathcal{B}_{train}$ with **each** $a \in A$, storing $t(b, a)$

Do I have to solve each training problem multiple times?

- ▶ learn a model $(b, a) \rightarrow t(b, a)$
- ▶ solve each subsequent b with corresponding “best” a

Can I predict performance before start?

Offline Static AS

- ▶ Pick training set \mathcal{B}_{train}

How do I pick the size?

Online learning

- ▶ solve **each** $b \in \mathcal{B}_{train}$ with **each** $a \in A$, storing $t(b, a)$

Do I have to solve each training problem multiple times?

Censored sampling

- ▶ learn a model $(b, a) \rightarrow t(b, a)$
- ▶ solve each subsequent b with corresponding “best” a

Can I predict performance before start?

Offline Static AS

- ▶ Pick training set \mathcal{B}_{train}

How do I pick the size?

Online learning

- ▶ solve **each** $b \in \mathcal{B}_{train}$ with **each** $a \in A$, storing $t(b, a)$

Do I have to solve each training problem multiple times?

Censored sampling

- ▶ learn a model $(b, a) \rightarrow t(b, a)$
- ▶ solve each subsequent b with corresponding “best” a

Can I predict performance before start?

Dynamic selection

Algorithm Survival Analysis

- ▶ Randomized algorithm a solving a *decision* problem.
- ▶ *Runtime* is a *random variable*, characterized by a *distribution* (runtime distribution, RTD).
- ▶ RTD can be described by its **cumulative distribution function** (CDF)

$$F(t) = \Pr\{\text{runtime} \leq t\}, \quad F : [0, \infty) \rightarrow [0, 1].$$

- ▶ Or by its **survival function**:

$$S(t) = 1 - F(t)$$

RTD is the natural model of performance for generalized Las Vegas algorithms.

Roadmap

Introduction

Algorithm Portfolios

Dynamic Allocation

Censored Sampling

Online Learning

Experiments

Conclusions

Algorithm Portfolios

$\mathcal{A} = \{a_1, a_2, \dots, a_N\}$ solve the **same** problem instance b , running on the **same** processor, without interacting. As soon as one algorithm solves the instance, all the others are halted.

- ▶ *resource sharing* schedule $\mathbf{s} \in [0, 1]^N, \sum s_n = 1$
- ▶ $\forall t, t_n = s_n t$ is allocated to a_n
- ▶ runtime $t_{\mathcal{A}}(\mathbf{s}) = \min_n \{t_n / s_n\}$

Alternative schedules:

- ▶ Algorithm selection $\mathbf{s} \in \{0, 1\}^N$
- ▶ Task switching schedules $\mathbf{s}(t) \in \{0, 1\}^N$
- ▶ Dynamic schedules $\mathbf{s}(t)$

Per instance vs. per set

Algorithm (portfolio) selection can be

- ▶ **per instance:** repeated independently for each problem instance
- ▶ **per set:** performed once for a set of instances

(Hutter et al. '05)

Related work

1. Per instance selection, based on regression models of expected runtime as a function of instance features:
 $\hat{E}\{t_n|\mathbf{x}_m\}$ (Leyton-Brown et al. '02, . . .)
2. Per instance portfolios based on runtime distributions (RTDs), assumed to be available: $F(t_n|m)$ (Huberman et al. '97, Gomes et al. '98, Finkelstein et al '02)
3. Per set portfolios based on runtime values, assumed to be available. PAC learning. (Petrik '05, Sayag et al. '06, Streeter et al. '07)
4. More general time allocation, interaction among algorithms (Littman et al. '00)

Related work

1. Per instance selection, based on regression models of expected runtime as a function of instance features:
 $\hat{E}\{t_n|\mathbf{x}_m\}$ (Leyton-Brown et al. '02, . . .)
2. Per instance portfolios based on runtime distributions (RTDs), assumed to be available: $F(t_n|m)$ (Huberman et al. '97, Gomes et al. '98, Finkelstein et al '02)
3. Per set portfolios based on runtime values, assumed to be available. PAC learning. (Petrik '05, Sayag et al. '06, Streeter et al. '07)
4. More general time allocation, interaction among algorithms (Littman et al. '00)

Our main contribution (1 + 2):

Per instance portfolios based on regression models of the RTD conditioned on instance features: $\hat{F}(t_n|\mathbf{x}_m)$.

Model based time allocation

Runtime of a portfolio: $\min\{t_k/s_k\}$. Also a random variable.
Its distribution will depend on each F_k and \mathbf{s}

Model based time allocation

Runtime of a portfolio: $\min\{t_k/s_k\}$. Also a random variable.
Its distribution will depend on each F_k and \mathbf{s}

- ▶ Evaluate the RTD of the portfolio for a given \mathbf{s}

Model based time allocation

Runtime of a portfolio: $\min\{t_k/s_k\}$. Also a random variable.
Its distribution will depend on each F_k and \mathbf{s}

- ▶ Evaluate the RTD of the portfolio for a given \mathbf{s}
- ▶ Use the RTD to set \mathbf{s} according to some criterion.

RTD of a portfolio

Can we evaluate RTD of \mathcal{A} based on $\{F_k\}$ and \mathbf{s} ?

- ▶ Wallclock time t , algorithm a_k time $t_k = s_k t$.
- ▶ Algorithm RTD $F_k(t_k)$.
- ▶ Event “ a_k solved the problem at wallclock time t , using share s_k ” has RTD

$$F_{k,s_k}(t) = F_k(s_k t).$$

- ▶ The RTD of portfolio \mathcal{A} with share \mathbf{s} is:

$$S_{\mathcal{A},\mathbf{s}}(t) = \prod_{k=1}^N S_k(s_k t),$$

Time Allocators (TA)

If I know each F_k I can evaluate $F_{\mathcal{A},\mathbf{s}} \forall \mathbf{s}$.
How can I evaluate \mathbf{s} , based on the RTD?

Time Allocators (TA)

If I know each F_k I can evaluate $F_{\mathcal{A},\mathbf{s}} \forall \mathbf{s}$.

How can I evaluate \mathbf{s} , based on the RTD?

- ▶ Minimizing expected solution time (Finkelstein et al. '02)

Time Allocators (TA)

If I know each F_k I can evaluate $F_{\mathcal{A},\mathbf{s}} \forall \mathbf{s}$.

How can I evaluate \mathbf{s} , based on the RTD?

- ▶ Minimizing expected solution time (Finkelstein et al. '02)
- ▶ Maximizing solution probability within a given **contract** t_u :

$$\mathbf{s} = \arg \max_{\mathbf{s}} F_{\mathcal{A},\mathbf{s}}(t_u).$$

Time Allocators (TA)

If I know each F_k I can evaluate $F_{\mathcal{A},\mathbf{s}} \forall \mathbf{s}$.

How can I evaluate \mathbf{s} , based on the RTD?

- ▶ Minimizing expected solution time (Finkelstein et al. '02)
- ▶ Maximizing solution probability within a given **contract** t_u :

$$\mathbf{s} = \arg \max_{\mathbf{s}} F_{\mathcal{A},\mathbf{s}}(t_u).$$

- ▶ Minimizing a **quantile** $\alpha \in [0, 1]$ of runtime:

$$\mathbf{s} = \arg \min_{\mathbf{s}} F_{\mathcal{A},\mathbf{s}}^{-1}(\alpha).$$

Time Allocators (TA)

If I know each F_k I can evaluate $F_{\mathcal{A},\mathbf{s}} \forall \mathbf{s}$.

How can I evaluate \mathbf{s} , based on the RTD?

- ▶ Minimizing expected solution time (Finkelstein et al. '02)
- ▶ Maximizing solution probability within a given **contract** t_u :

$$\mathbf{s} = \arg \max_{\mathbf{s}} F_{\mathcal{A},\mathbf{s}}(t_u).$$

- ▶ Minimizing a **quantile** $\alpha \in [0, 1]$ of runtime:

$$\mathbf{s} = \arg \min_{\mathbf{s}} F_{\mathcal{A},\mathbf{s}}^{-1}(\alpha).$$

Constrained function optimization in $[0, 1]^K$ ($K - 1$ dimensions).

An example: SAT-UNSAT

Problem set:

Algorithm set:

An example: SAT-UNSAT

Problem set:

- ▶ SAT instances at phase transition, 50% unsat.
(uf-*, uu-* from SATLIB)

Algorithm set:

An example: SAT-UNSAT

Problem set:

- ▶ SAT instances at phase transition, 50% unsat.
(uf-*, uu-* from SATLIB)

Algorithm set:

- ▶ a_1 : complete solver (Satz-Rand, Gomes et al. '00).
 - ▶ Backtracking search with randomized branching.
 - ▶ Guaranteed to solve, or prove unsat.
 - ▶ Huge runtime variations.

An example: SAT-UNSAT

Problem set:

- ▶ SAT instances at phase transition, 50% unsat.
(uf-*, uu-* from SATLIB)

Algorithm set:

- ▶ a_1 : complete solver (Satz-Rand, Gomes et al. '00).
 - ▶ Backtracking search with randomized branching.
 - ▶ Guaranteed to solve, or prove unsat.
 - ▶ Huge runtime variations.
- ▶ a_2 : incomplete solver (G2-WSAT, Li et al '05).
 - ▶ Local search.
 - ▶ Much faster on sat instances.
 - ▶ Cannot solve unsat (infinite runtime).

An example: SAT-UNSAT

Problem set:

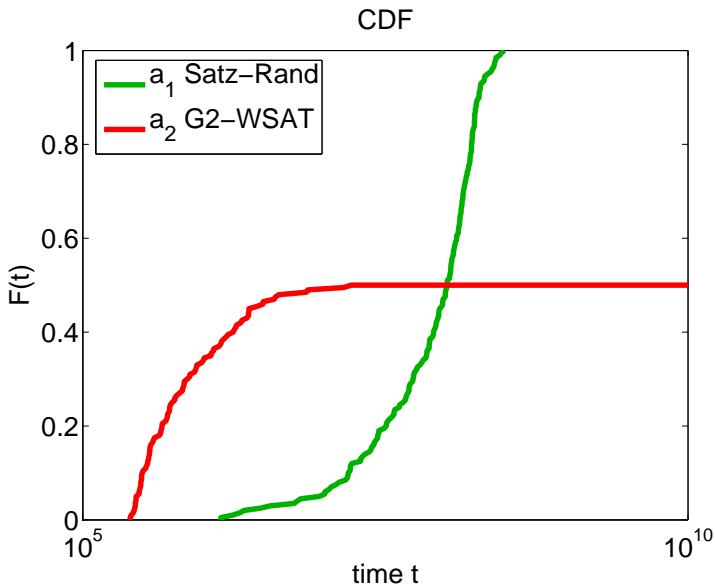
- ▶ SAT instances at phase transition, 50% unsat.
(uf-*, uu-* from SATLIB)

Algorithm set:

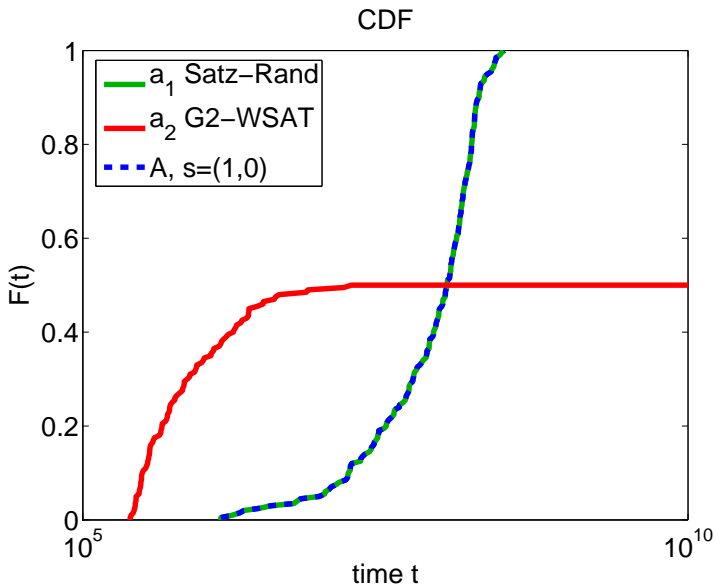
- ▶ a_1 : complete solver (Satz-Rand, Gomes et al. '00).
 - ▶ Backtracking search with randomized branching.
 - ▶ Guaranteed to solve, or prove unsat.
 - ▶ Huge runtime variations.
- ▶ a_2 : incomplete solver (G2-WSAT, Li et al '05).
 - ▶ Local search.
 - ▶ Much faster on sat instances.
 - ▶ Cannot solve unsat (infinite runtime).

Easy, if we could tell satisfiability in advance...

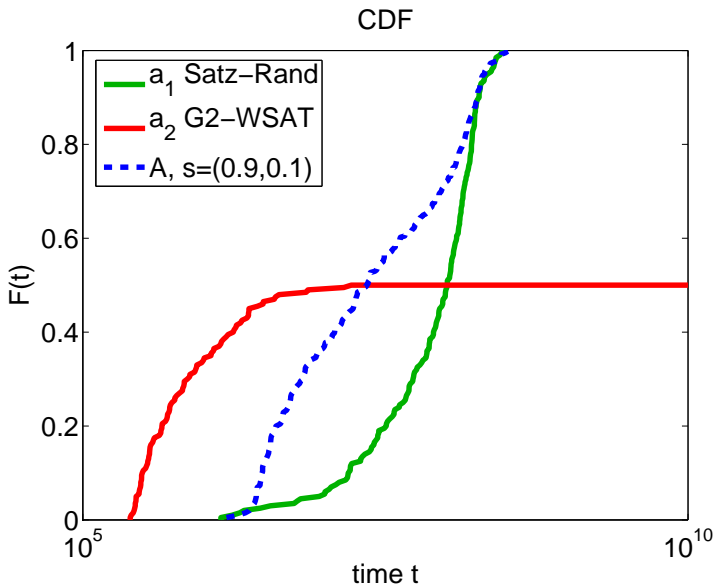
SAT-UNSAT: Portfolio RTD



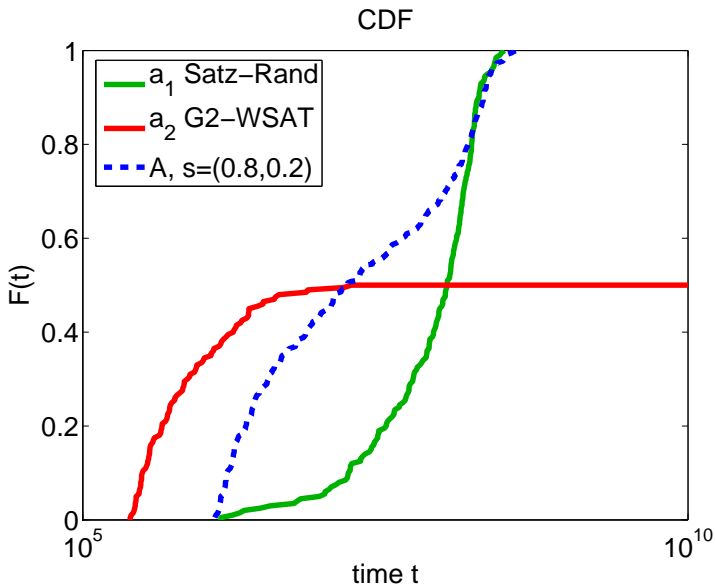
SAT-UNSAT: Portfolio RTD



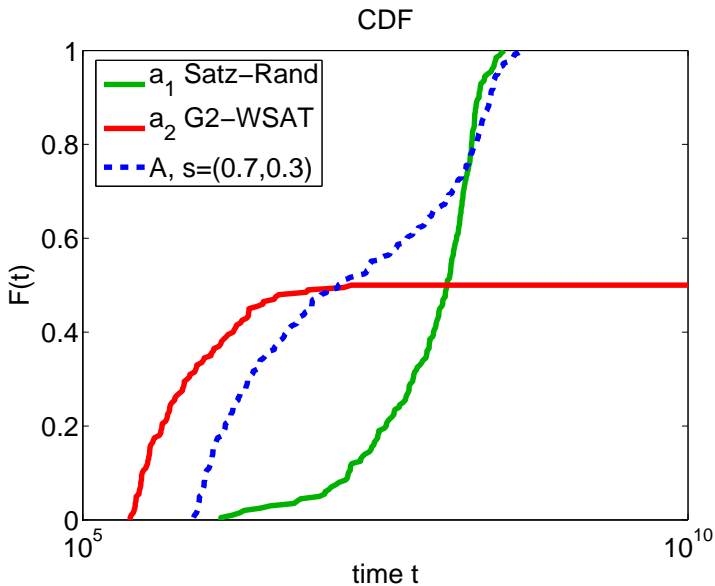
SAT-UNSAT: Portfolio RTD



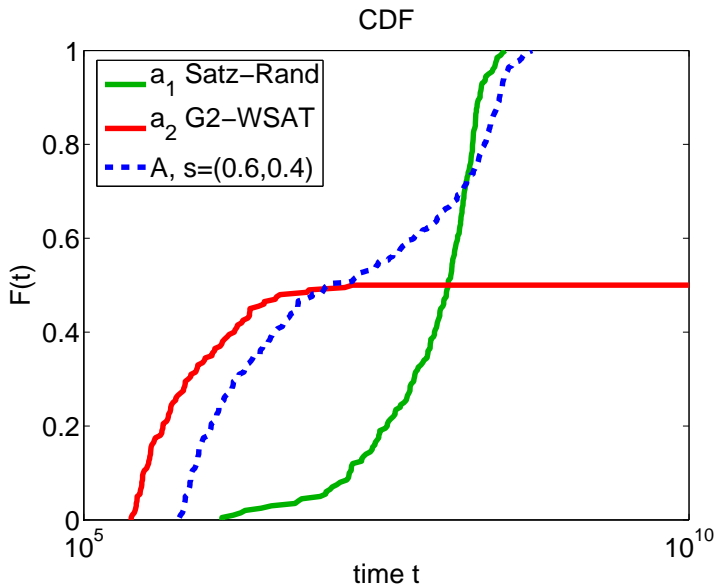
SAT-UNSAT: Portfolio RTD



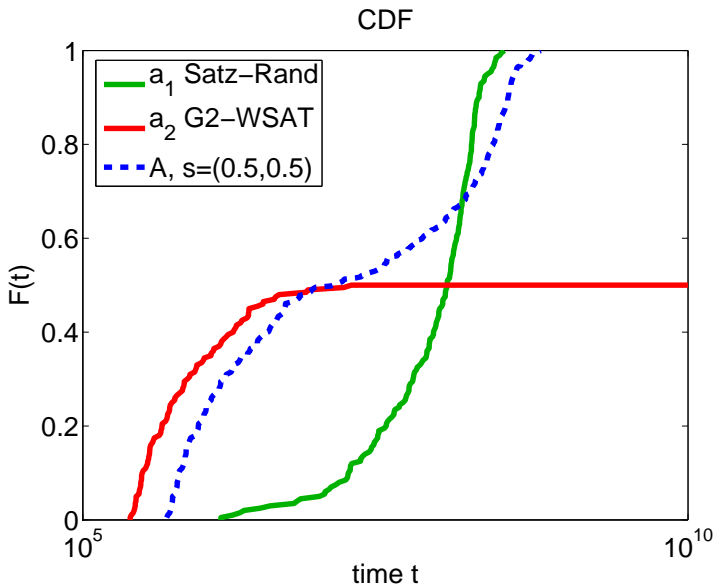
SAT-UNSAT: Portfolio RTD



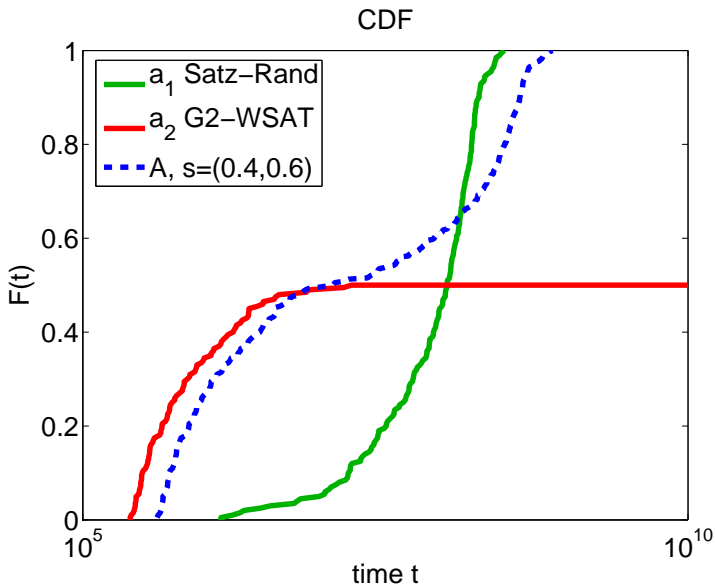
SAT-UNSAT: Portfolio RTD



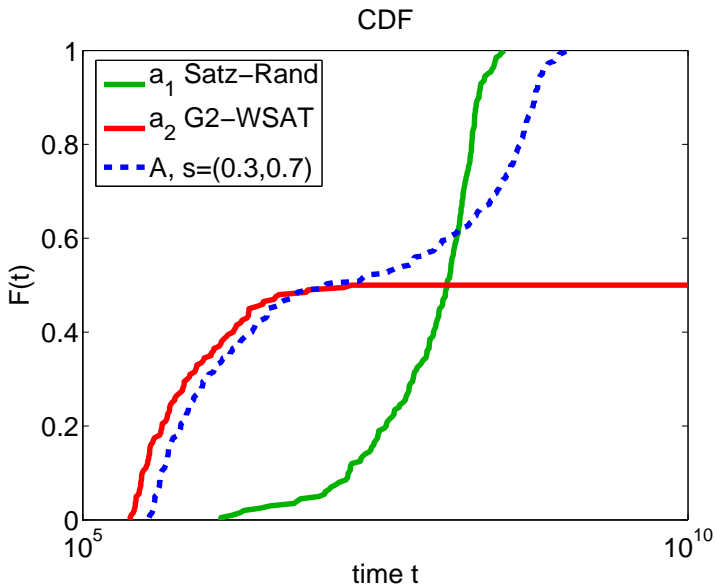
SAT-UNSAT: Portfolio RTD



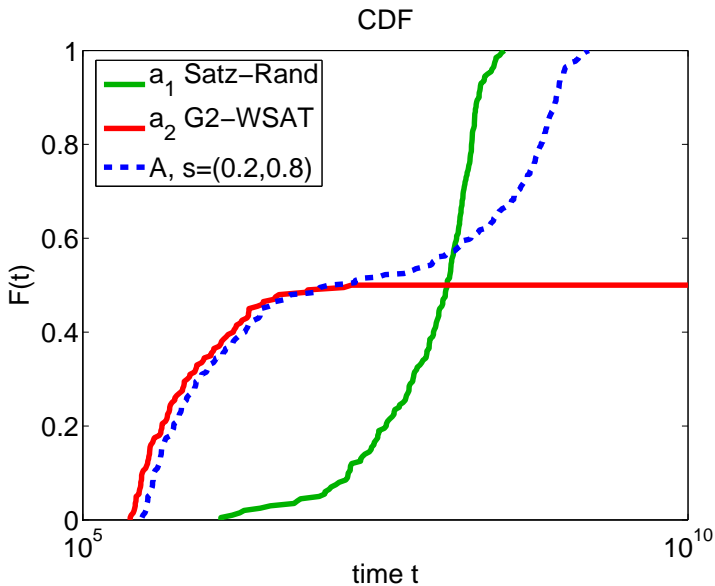
SAT-UNSAT: Portfolio RTD



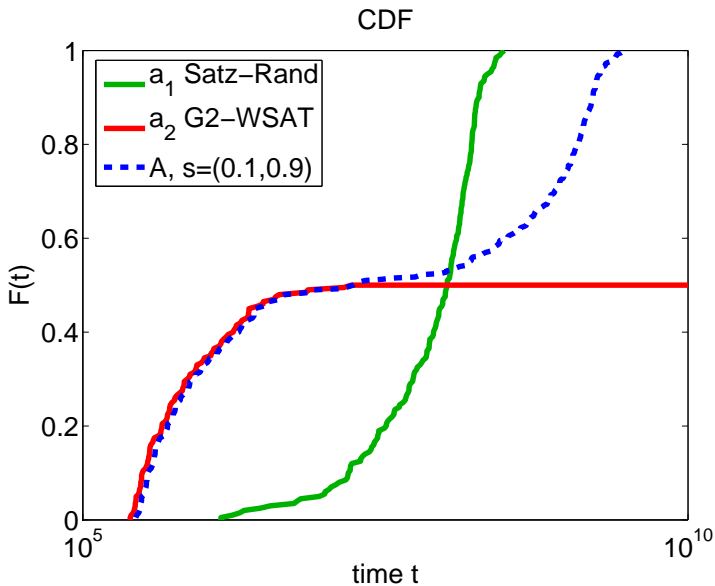
SAT-UNSAT: Portfolio RTD



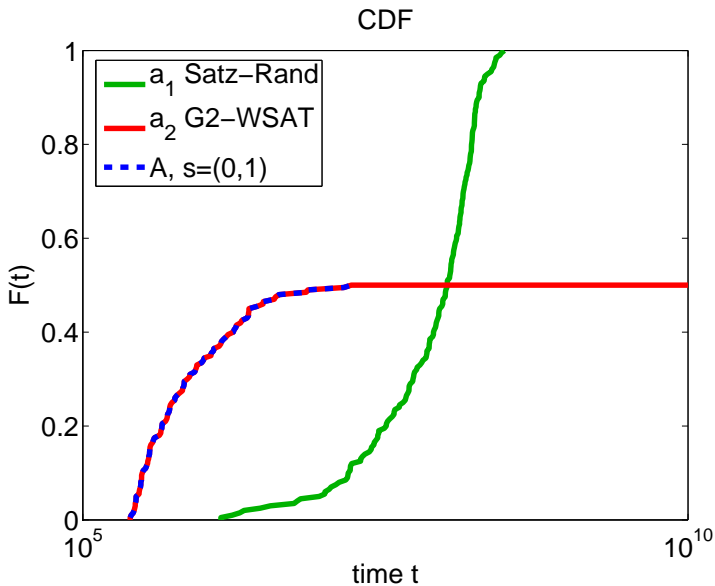
SAT-UNSAT: Portfolio RTD



SAT-UNSAT: Portfolio RTD

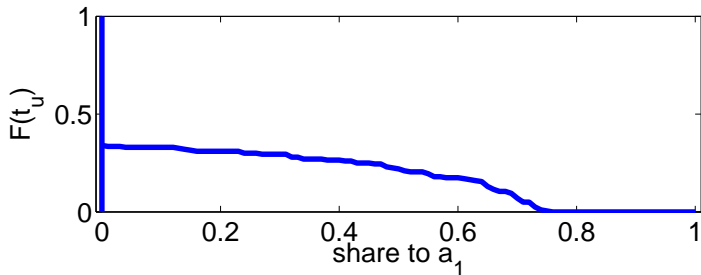
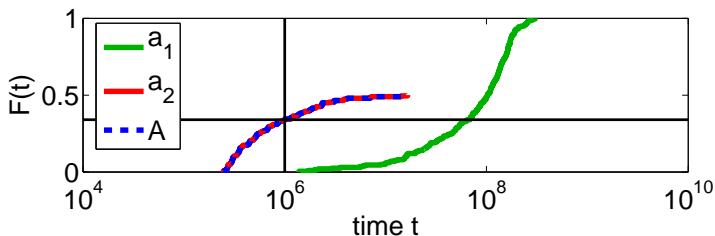


SAT-UNSAT: Portfolio RTD



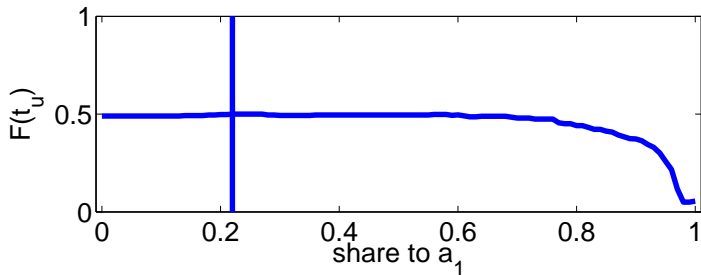
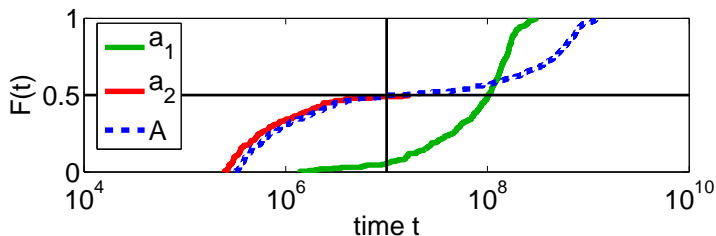
SAT-UNSAT: Contract allocation rule

contract $t_u = 10^6$



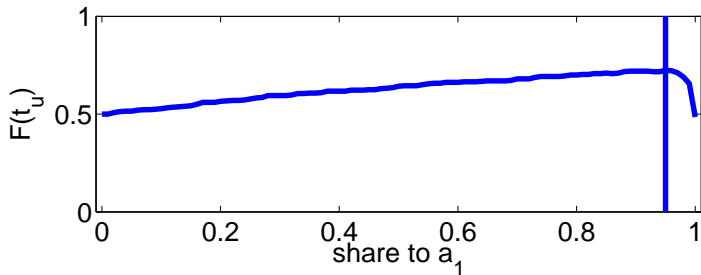
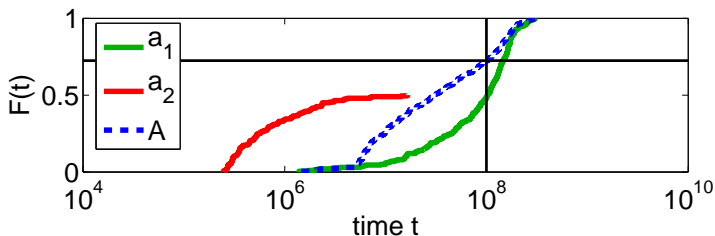
SAT-UNSAT: Contract allocation rule

contract $t_u = 10^7$



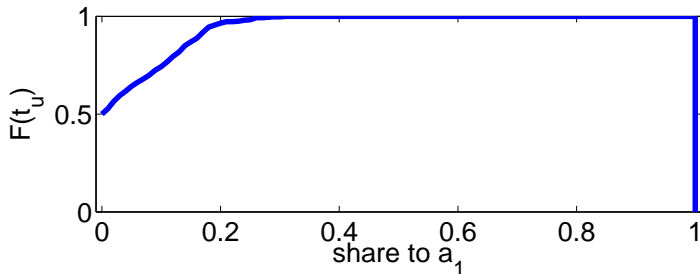
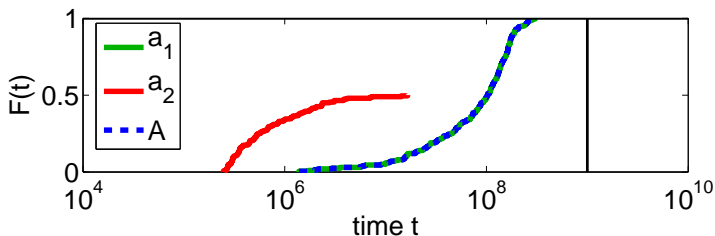
SAT-UNSAT: Contract allocation rule

contract $t_u = 10^8$



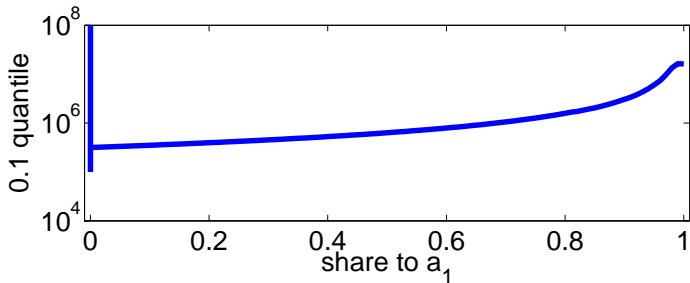
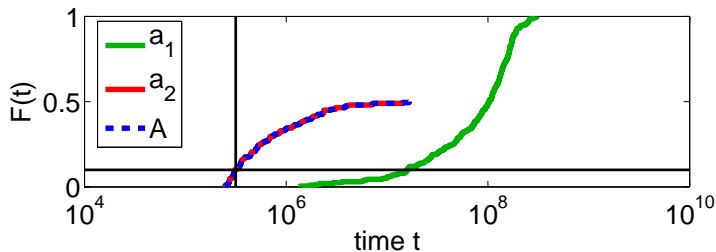
SAT-UNSAT: Contract allocation rule

contract $t_u = 10^9$



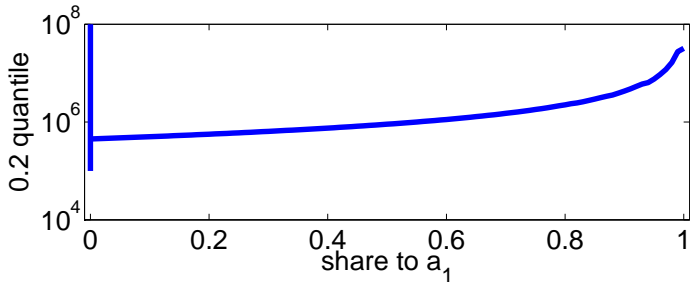
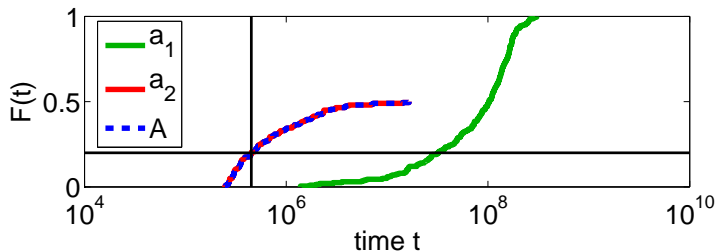
SAT-UNSAT: Quantile allocation rule

quantile $\alpha=0.1$



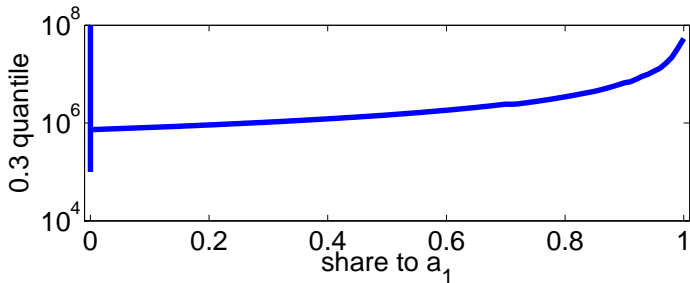
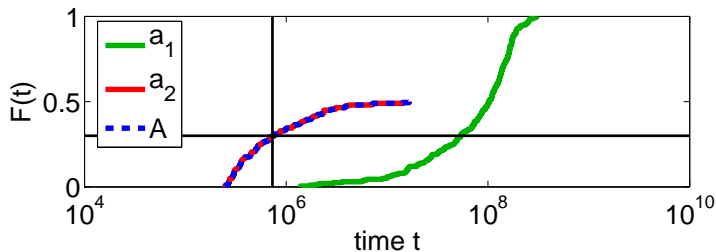
SAT-UNSAT: Quantile allocation rule

quantile $\alpha=0.2$



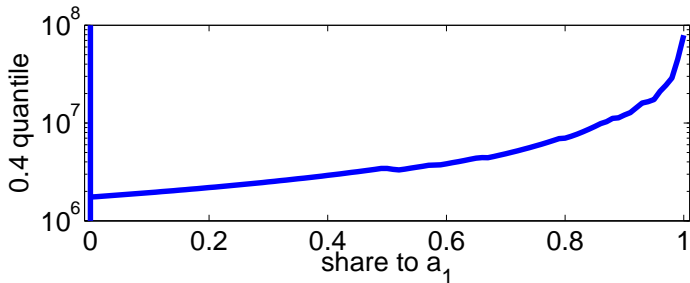
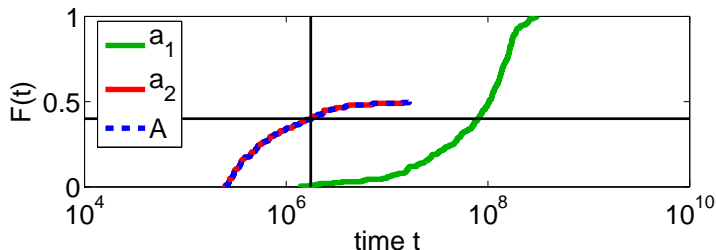
SAT-UNSAT: Quantile allocation rule

quantile $\alpha=0.3$



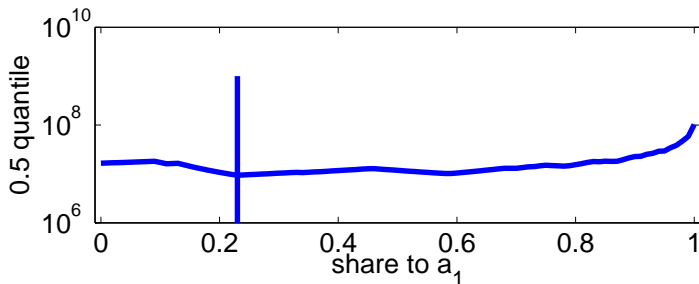
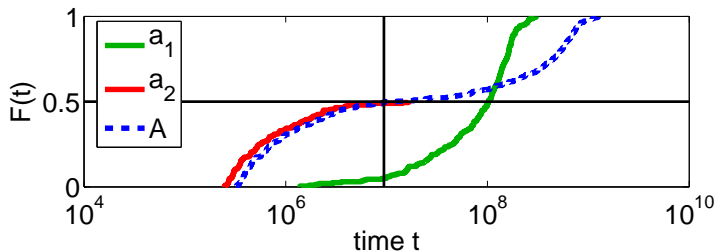
SAT-UNSAT: Quantile allocation rule

quantile $\alpha=0.4$



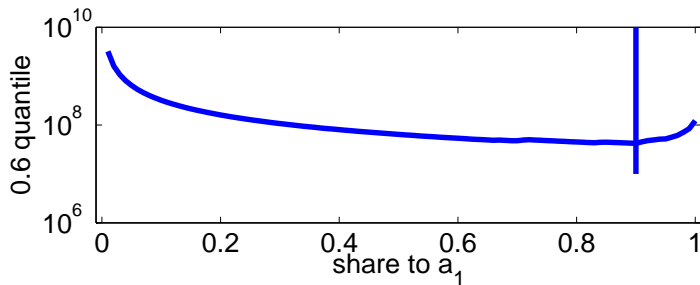
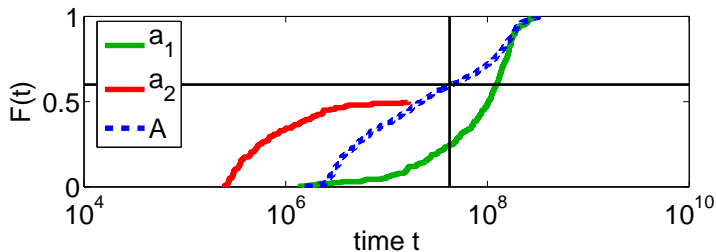
SAT-UNSAT: Quantile allocation rule

quantile $\alpha=0.5$



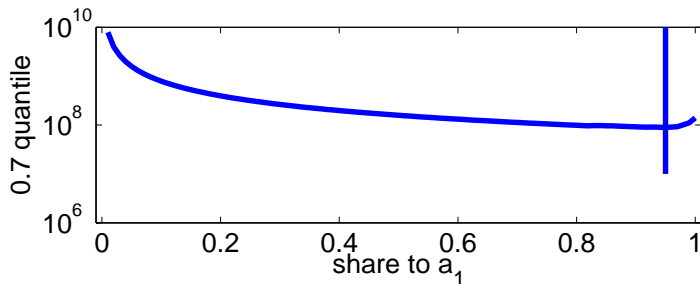
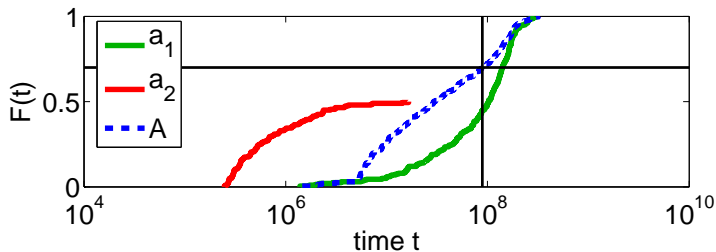
SAT-UNSAT: Quantile allocation rule

quantile $\alpha=0.6$



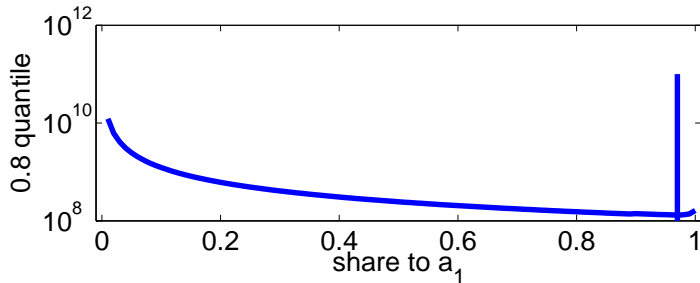
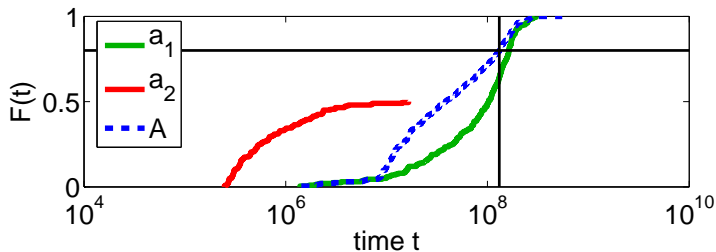
SAT-UNSAT: Quantile allocation rule

quantile $\alpha=0.7$



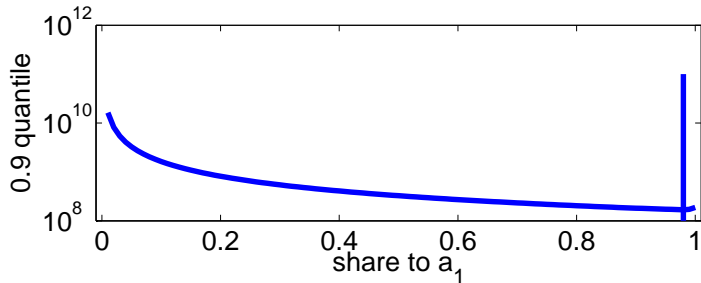
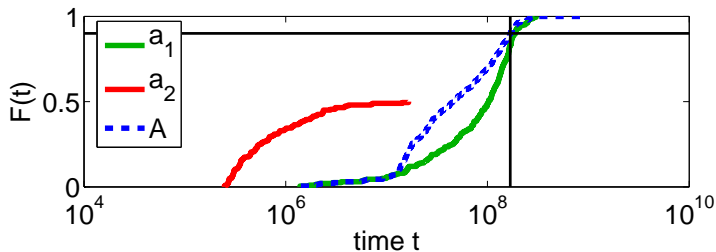
SAT-UNSAT: Quantile allocation rule

quantile $\alpha=0.8$



SAT-UNSAT: Quantile allocation rule

quantile $\alpha=0.9$



Roadmap

Introduction

Algorithm Portfolios

Dynamic Allocation

Censored Sampling

Online Learning

Experiments

Conclusions

Static vs. dynamic selection

Can I predict performance before start?

Static vs. dynamic selection

Can I predict performance before start?

- ▶ $F_a(t|b)$ conditioned on problem features cannot discriminate between SAT and UNSAT.

Static vs. dynamic selection

Can I predict performance before start?

- ▶ $F_a(t|b)$ conditioned on problem features cannot discriminate between SAT and UNSAT.
- ▶ How would a human do? Try to run the two a for a while, then decide.

Static vs. dynamic selection

Can I predict performance before start?

- ▶ $F_a(t|b)$ conditioned on problem features cannot discriminate between SAT and UNSAT.
- ▶ How would a human do? Try to run the two a for a while, then decide.
- ▶ **Idea:** maybe I should condition on **state** information, obtained **at runtime**.

Static vs. dynamic selection

Can I predict performance before start?

- ▶ $F_a(t|b)$ conditioned on problem features cannot discriminate between SAT and UNSAT.
- ▶ How would a human do? Try to run the two a for a while, then decide.
- ▶ **Idea:** maybe I should condition on **state** information, obtained **at runtime**.
- ▶ Simplest: **time spent**.

$$F(t|T > y) = \frac{F(t) - F(y)}{1 - F(y)}$$

Static vs. dynamic selection

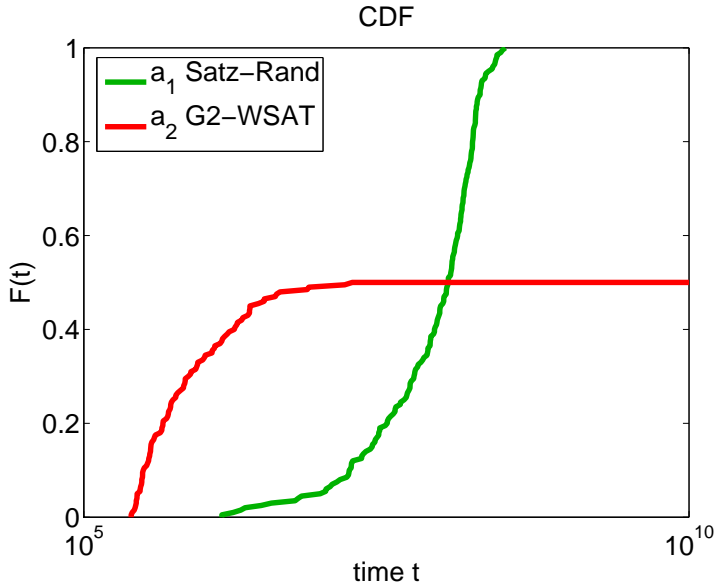
Can I predict performance before start?

- ▶ $F_a(t|b)$ conditioned on problem features cannot discriminate between SAT and UNSAT.
- ▶ How would a human do? Try to run the two a for a while, then decide.
- ▶ **Idea:** maybe I should condition on **state** information, obtained **at runtime**.
- ▶ Simplest: **time spent**.

$$F(t|T > y) = \frac{F(t) - F(y)}{1 - F(y)}$$

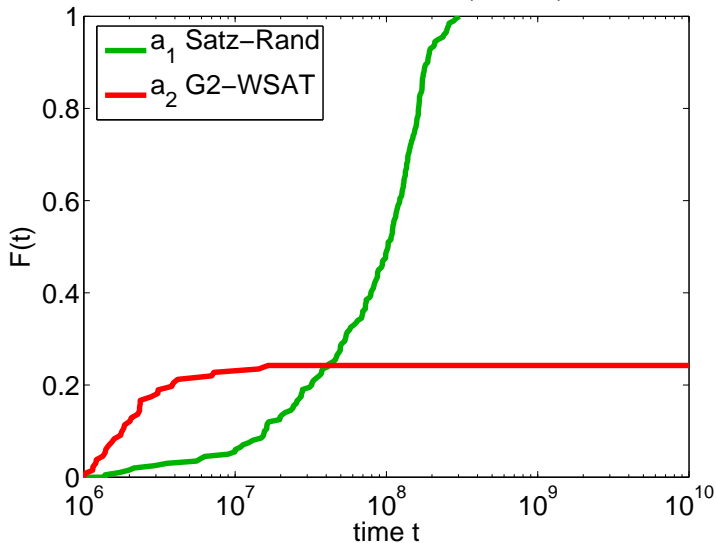
- ▶ More complex: **time-varying** covariates.

SAT-UNSAT: Dynamic RTD



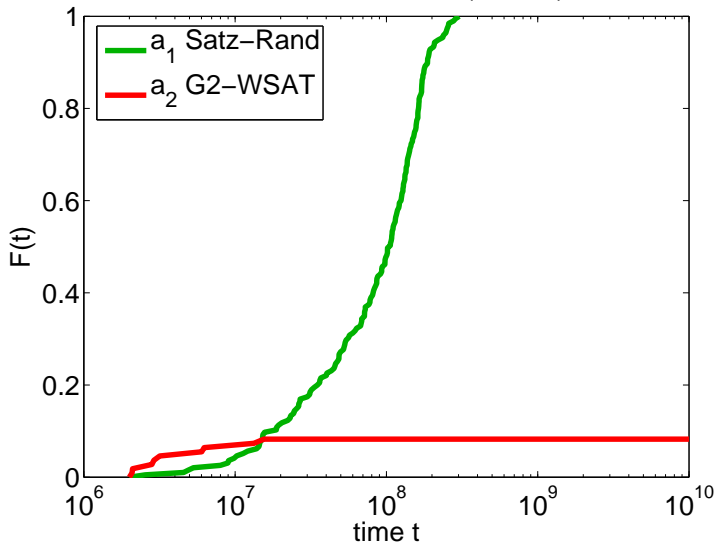
SAT-UNSAT: Dynamic RTD

CDF after $t=2 \times 10^6$ $s=(0.5,0.5)$



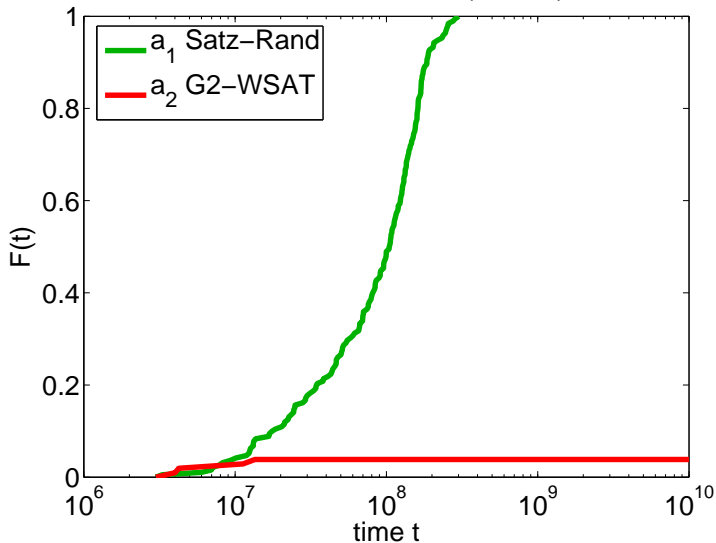
SAT-UNSAT: Dynamic RTD

CDF after $t=4 \times 10^6$ $s=(0.5,0.5)$



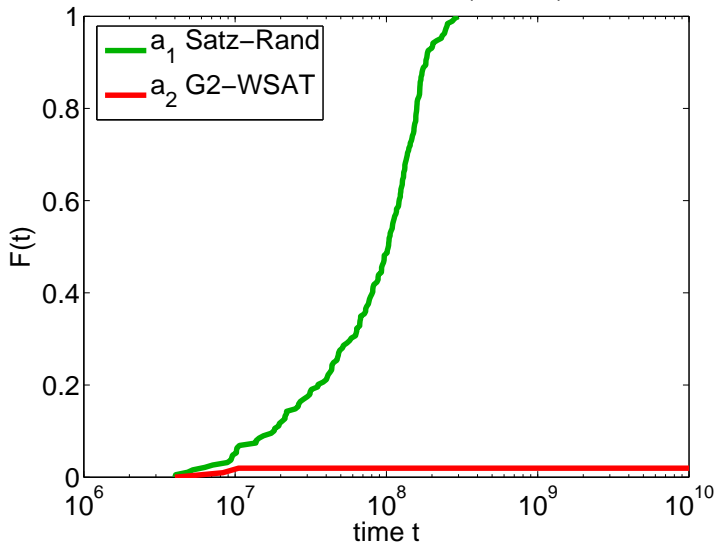
SAT-UNSAT: Dynamic RTD

CDF after $t=6 \times 10^6$ $s=(0.5,0.5)$



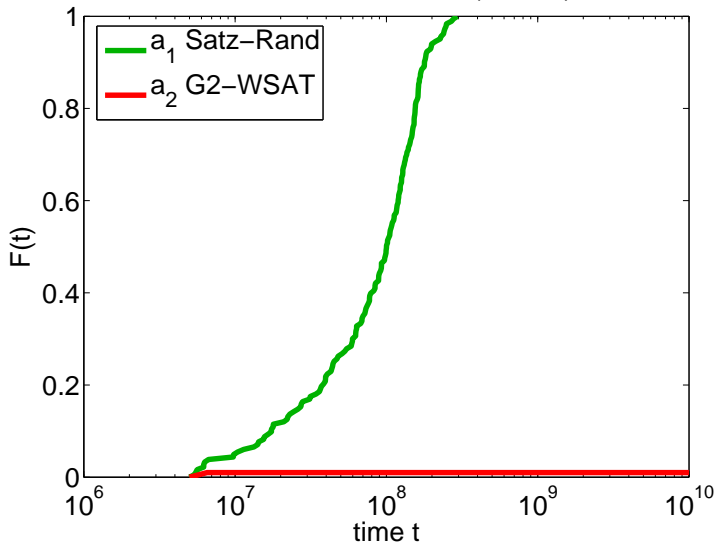
SAT-UNSAT: Dynamic RTD

CDF after $t=8 \times 10^6$ $s=(0.5,0.5)$



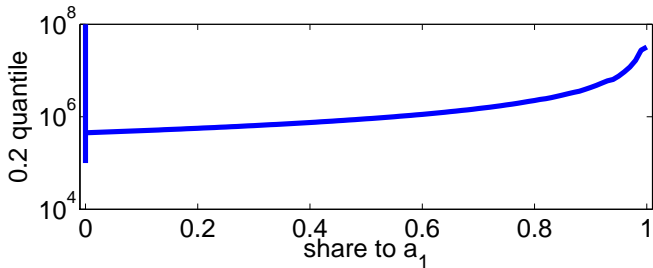
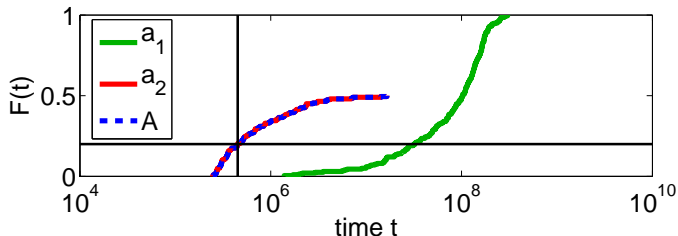
SAT-UNSAT: Dynamic RTD

CDF after $t=10 \times 10^6$ s=(0.5,0.5)



SAT-UNSAT: Dynamic allocation

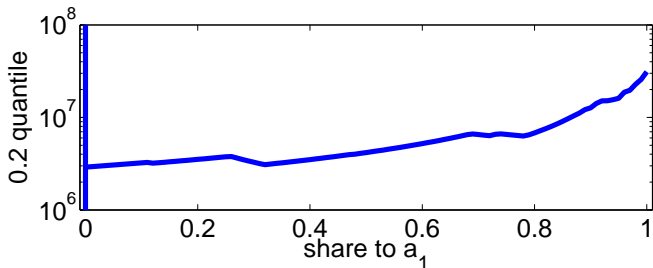
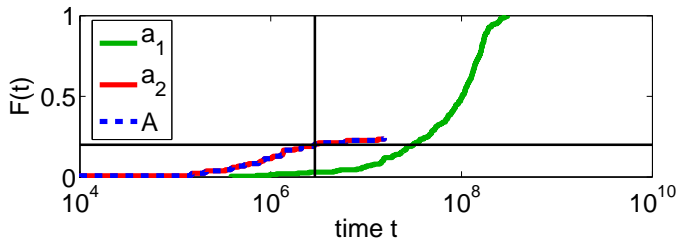
At $t = 0$
quantile $\alpha=0.2$



SAT-UNSAT: Dynamic allocation

After $t = 2 \times 10^6$ $s=(0.5,0.5)$

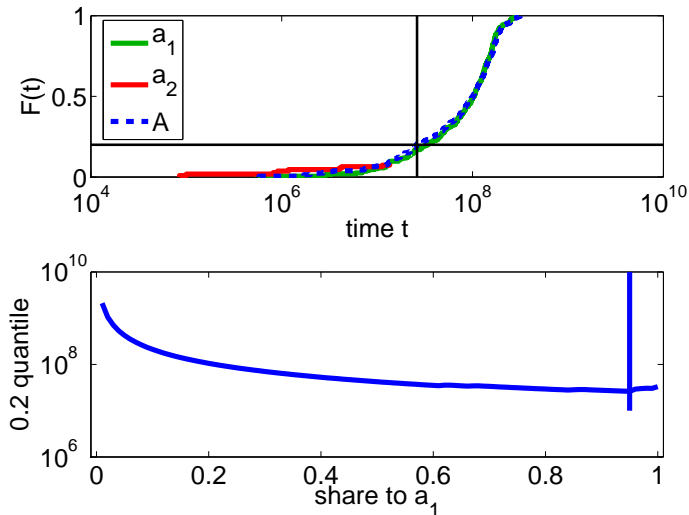
quantile $\alpha=0.2$



SAT-UNSAT: Dynamic allocation

After $t = 4 \times 10^6$ $s=(0.5,0.5)$

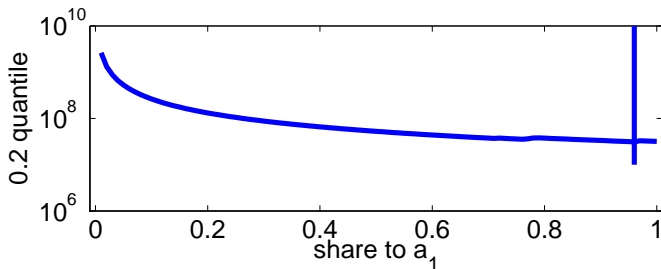
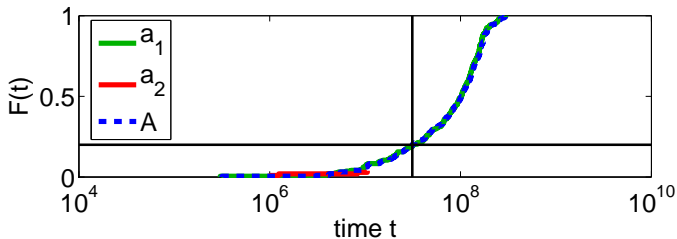
quantile $\alpha=0.2$



SAT-UNSAT: Dynamic allocation

After $t = 6 \times 10^6$ $s=(0.5,0.5)$

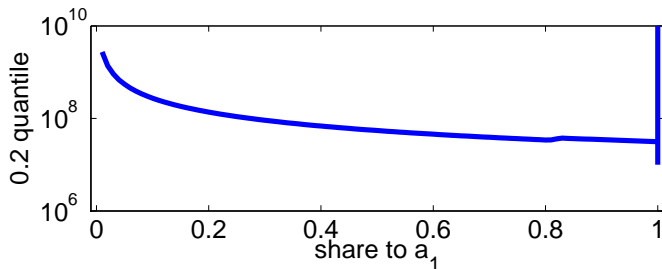
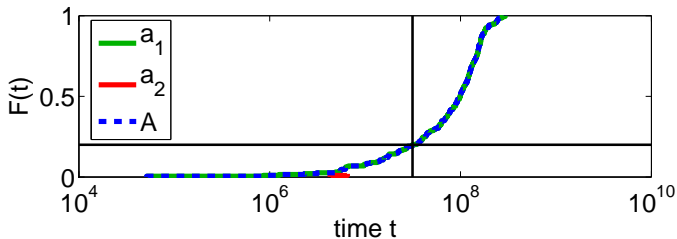
quantile $\alpha=0.2$



SAT-UNSAT: Dynamic allocation

After $t = 8 \times 10^6$ $s=(0.5,0.5)$

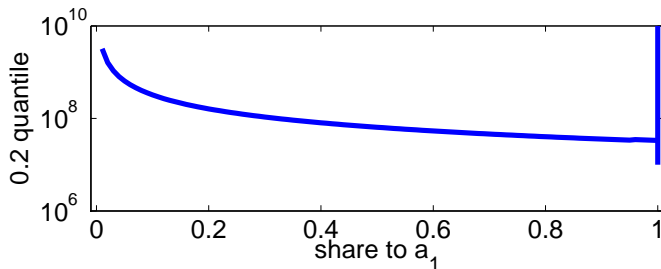
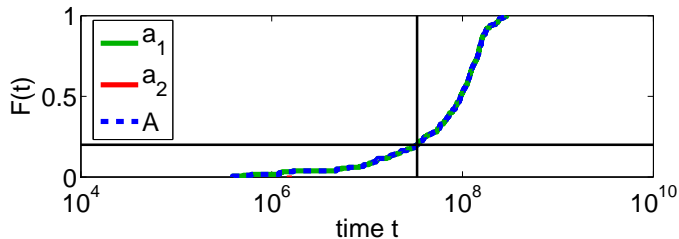
quantile $\alpha=0.2$



SAT-UNSAT: Dynamic allocation

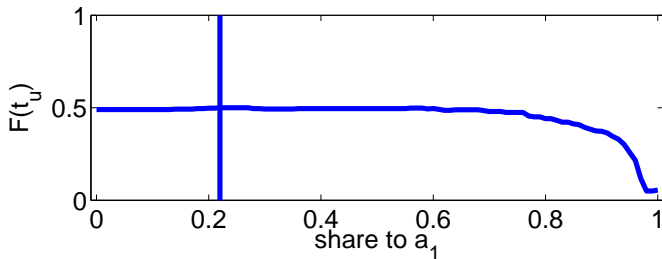
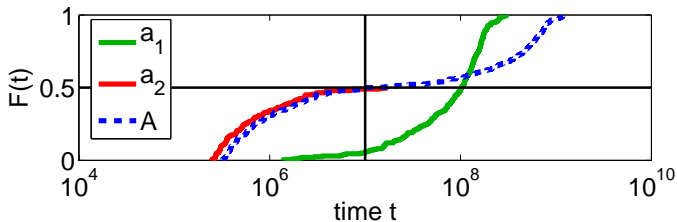
After $t = 10^7$ $s=(0.5,0.5)$

quantile $\alpha=0.2$



SAT-UNSAT: Dynamic allocation

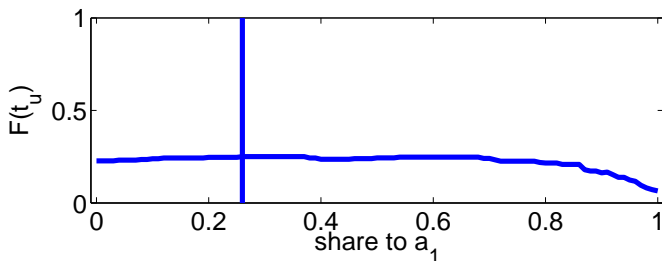
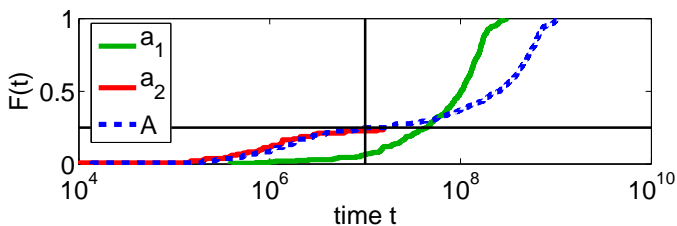
At $t = 0$
contract $t_u = 10^7$



SAT-UNSAT: Dynamic allocation

After $t = 2 \times 10^6$ $s=(0.5,0.5)$

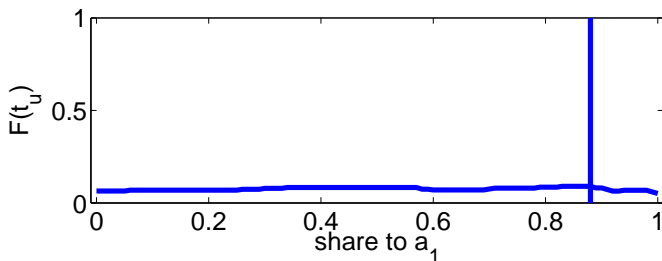
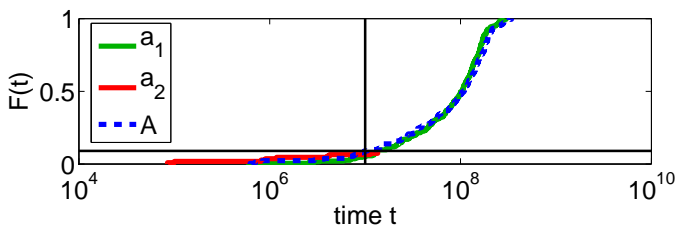
contract $t_u = 10^7$



SAT-UNSAT: Dynamic allocation

After $t = 4 \times 10^6$ $s=(0.5,0.5)$

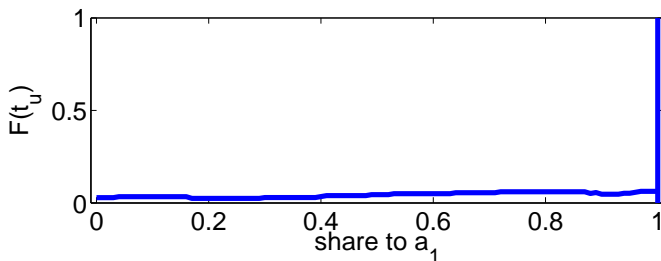
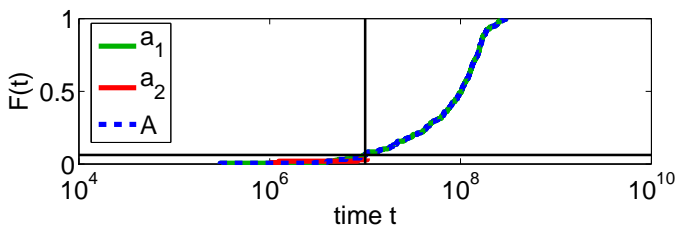
contract $t_u=10^7$



SAT-UNSAT: Dynamic allocation

After $t = 6 \times 10^6$ $s=(0.5,0.5)$

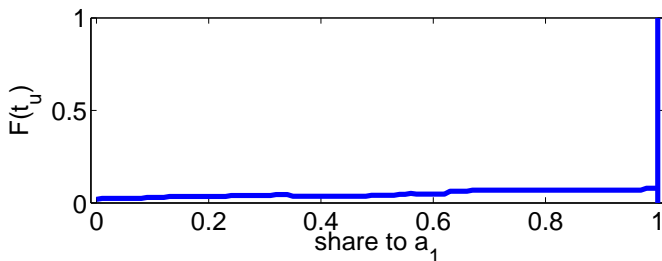
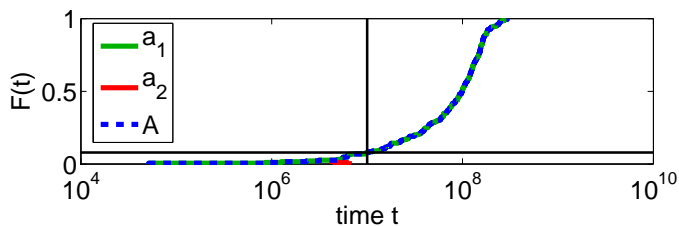
contract $t_u = 10^7$



SAT-UNSAT: Dynamic allocation

After $t = 8 \times 10^6$ $s=(0.5,0.5)$

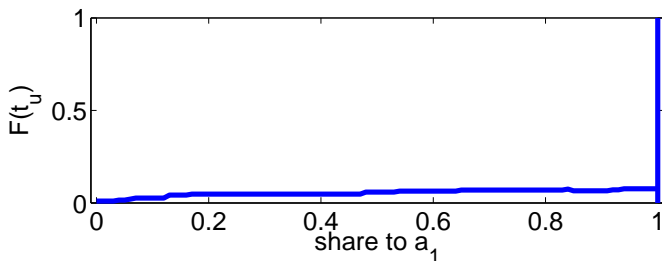
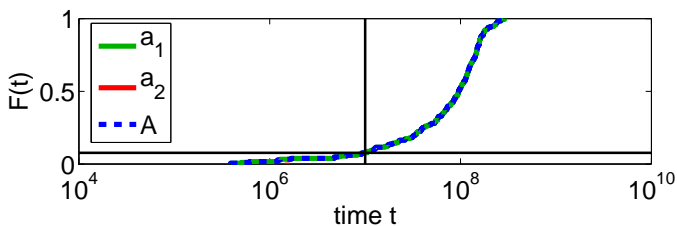
contract $t_u = 10^7$



SAT-UNSAT: Dynamic allocation

After $t = 10^7$ $s=(0.5,0.5)$

contract $t_u=10^7$



Offline Dynamic Algorithm Portfolio

- ▶ Pick “training” set \mathcal{B}_{train} of problem instances
- ▶ solve **each** $b \in \mathcal{B}_{train}$ with **each** $a \in \mathcal{A}$, storing $t(b, a)$
- ▶ learn a model of conditional RTD $F_a(t|b)$
- ▶ solve each subsequent b with portfolio \mathcal{A} with share $\mathbf{s} = \text{TA}(F_a(t|b))$

Can I predict performance before start?

Offline Dynamic Algorithm Portfolio

- ▶ Pick “training” set \mathcal{B}_{train} of problem instances
- ▶ solve **each** $b \in \mathcal{B}_{train}$ with **each** $a \in \mathcal{A}$, storing $t(b, a)$
- ▶ learn a model of conditional RTD $F_a(t|b)$
- ▶ *For each b , while not solved:*

Offline **Dynamic** Algorithm Portfolio

- ▶ Pick “training” set \mathcal{B}_{train} of problem instances
- ▶ solve **each** $b \in \mathcal{B}_{train}$ with **each** $a \in \mathcal{A}$, storing $t(b, a)$
- ▶ learn a model of conditional RTD $F_a(t|b)$
- ▶ *For each b , while not solved:*
 - ▶ *update RTDs $F_k(t|b, y_k) \forall a_k$*

Offline **Dynamic** Algorithm Portfolio

- ▶ Pick “training” set \mathcal{B}_{train} of problem instances
- ▶ solve **each** $b \in \mathcal{B}_{train}$ with **each** $a \in \mathcal{A}$, storing $t(b, a)$
- ▶ learn a model of conditional RTD $F_a(t|b)$
- ▶ *For each b , while not solved:*
 - ▶ *update RTDs $F_k(t|b, y_k) \forall a_k$*
 - ▶ *update Δt*

Offline **Dynamic** Algorithm Portfolio

- ▶ Pick “training” set \mathcal{B}_{train} of problem instances
- ▶ solve **each** $b \in \mathcal{B}_{train}$ with **each** $a \in \mathcal{A}$, storing $t(b, a)$
- ▶ learn a model of conditional RTD $F_a(t|b)$
- ▶ *For each b , while not solved:*
 - ▶ *update RTDs $F_k(t|b, y_k) \forall a_k$*
 - ▶ *update Δt*
 - ▶ *update $\mathbf{s} = TA(\{F_k\})$*

Offline **Dynamic** Algorithm Portfolio

- ▶ Pick “training” set \mathcal{B}_{train} of problem instances
- ▶ solve **each** $b \in \mathcal{B}_{train}$ with **each** $a \in \mathcal{A}$, storing $t(b, a)$
- ▶ learn a model of conditional RTD $F_a(t|b)$
- ▶ *For each b , while not solved:*
 - ▶ *update RTDs $F_k(t|b, y_k) \forall a_k$*
 - ▶ *update Δt*
 - ▶ *update $\mathbf{s} = TA(\{F_k\})$*
 - ▶ *run portfolio \mathcal{A} with share \mathbf{s} up to time Δt*

Offline **Dynamic** Algorithm Portfolio

- ▶ Pick “training” set \mathcal{B}_{train} of problem instances

How do I pick \mathcal{B}_{train} ? TA?

- ▶ solve **each** $b \in \mathcal{B}_{train}$ with **each** $a \in \mathcal{A}$, storing $t(b, a)$

Do I have to solve each training problem multiple times?

- ▶ learn a model of conditional RTD $F_a(t|b)$

- ▶ *For each b , while not solved:*

- ▶ *update RTDs $F_k(t|b, y_k) \forall a_k$*
- ▶ *update Δt*
- ▶ *update $\mathbf{s} = TA(\{F_k\})$*
- ▶ *run portfolio \mathcal{A} with share \mathbf{s} up to time Δt*

Roadmap

Introduction

Algorithm Portfolios

Dynamic Allocation

Censored Sampling

Online Learning

Experiments

Conclusions

Censored Sampling

Do I have to solve each training problem multiple times?

Censored Sampling

Do I have to solve each training problem multiple times?

- ▶ When the portfolio solves a problem instance, I get to know the runtime of **one** of the algorithms. The others are still running.

Censored Sampling

Do I have to solve each training problem multiple times?

- ▶ When the portfolio solves a problem instance, I get to know the runtime of **one** of the algorithms. The others are still running.
- ▶ Ideally I would like to **stop** the remaining algorithms.

Censored Sampling

Do I have to solve each training problem multiple times?

- ▶ When the portfolio solves a problem instance, I get to know the runtime of **one** of the algorithms. The others are still running.
- ▶ Ideally I would like to **stop** the remaining algorithms.
- ▶ If I do stop them, I can only gather some **incomplete** information on the runtime (a lower bound).

Censored sampling

Do I have to solve each training problem multiple times?

Example:

An engineer wants to estimate the lifetime distribution of a new kind of lightbulb. To collect a reliable lifetime sample, he would like to leave a large number N of bulbs on, until they all fail.

Censored sampling

Do I have to solve each training problem multiple times?

Example:

An engineer wants to estimate the lifetime distribution of a new kind of lightbulb. To collect a reliable lifetime sample, he would like to leave a large number N of bulbs on, until they all fail.

This takes too long

Censored sampling

Do I have to solve each training problem multiple times?

Example:

An engineer wants to estimate the lifetime distribution of a new kind of lightbulb. To collect a reliable lifetime sample, he would like to leave a large number N of bulbs on, until they all fail.

This takes too long

He will have to stop the experiments after $n \ll N$ bulbs fail, and record failure times, and lifetime of remaining bulbs, forming a **censored** sample.

Censored sampling

Do I have to solve each training problem multiple times?

Example:

An engineer wants to estimate the lifetime distribution of a new kind of lightbulb. To collect a reliable lifetime sample, he would like to leave a large number N of bulbs on, until they all fail.

This takes too long

He will have to stop the experiments after $n \ll N$ bulbs fail, and record failure times, and lifetime of remaining bulbs, forming a **censored** sample.

Incomplete data is quite common in survival analysis, and cannot be discarded. Most estimators can deal with it.

Censored sampling

Do I have to solve each training problem multiple times?

Example:

An engineer wants to estimate the lifetime distribution of a new kind of lightbulb. To collect a reliable lifetime sample, he would like to leave a large number N of bulbs on, until they all fail.

This takes too long

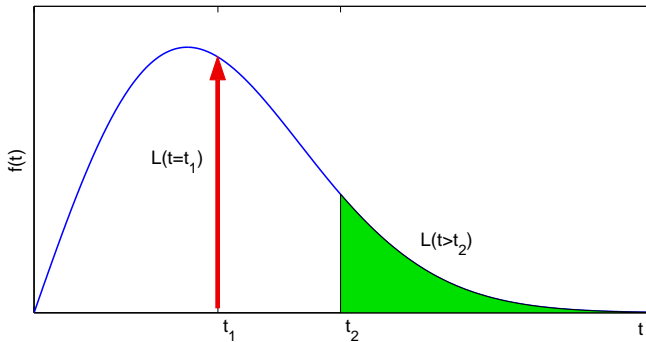
He will have to stop the experiments after $n \ll N$ bulbs fail, and record failure times, and lifetime of remaining bulbs, forming a **censored** sample.

Incomplete data is quite common in survival analysis, and cannot be discarded. Most estimators can deal with it.

*Also **runtime sampling can be censored**: there is no need to solve the same problem twice!*

Censored sampling

Parametric pdf model: one component **fails** at time t_1 , one survives until time t_2



Roadmap

Introduction

Algorithm Portfolios

Dynamic Allocation

Censored Sampling

Online Learning

Experiments

Conclusions

Time Allocation

Model-based or not, all practical time allocators are based on a *runtime sample*.

How to collect it efficiently?

Given \mathcal{A}, \mathcal{B} , and a time allocator $\text{TA}_{\mathcal{M}}$

- ▶ Pick M_0 “training” instances
- ▶ solve **each** $b_m, m \leq M_0$ with **each** $a_n \in \mathcal{A}$, storing $t_n(m)$
- ▶ (learn a model of performance \mathcal{M})
- ▶ allocate time on the remaining $M - M_0$ instances using $\text{TA}_{\mathcal{M}}$

Time Allocation

Model-based or not, all practical time allocators are based on a *runtime sample*.

How to collect it efficiently?

Given \mathcal{A}, \mathcal{B} , and a time allocator $\text{TA}_{\mathcal{M}}$

Offline learning:

- ▶ Pick M_0 “training” instances
- ▶ solve **each** b_m , $m \leq M_0$ with **each** $a_n \in \mathcal{A}$, storing $t_n(m)$
- ▶ (learn a model of performance \mathcal{M})
- ▶ allocate time on the remaining $M - M_0$ instances using $\text{TA}_{\mathcal{M}}$

Time Allocation

Model-based or not, all practical time allocators are based on a *runtime sample*.

How to collect it efficiently?

Given \mathcal{A}, \mathcal{B} , and a time allocator $\text{TA}_{\mathcal{M}}$

Offline learning:

- ▶ Pick M_0 “training” instances
- ▶ solve **each** $b_m, m \leq M_0$ **with UNIFORM** updating a **censored** runtime sample
- ▶ (learn a model of performance \mathcal{M})
- ▶ allocate time on the remaining $M - M_0$ instances using $\text{TA}_{\mathcal{M}}$

Time Allocation

Model-based or not, all practical time allocators are based on a *runtime sample*.

How to collect it efficiently?

Given \mathcal{A}, \mathcal{B} , and a time allocator $\text{TA}_{\mathcal{M}}$

Offline learning:

- ▶ Pick M_0 “training” instances

How do I pick M_0 ?

- ▶ solve each $b_m, m \leq M_0$ with UNIFORM updating a censored runtime sample
- ▶ (learn a model of performance \mathcal{M})
- ▶ allocate time on the remaining $M - M_0$ instances using $\text{TA}_{\mathcal{M}}$

Exploration-Exploitation Trade-off in TA

How do I pick the size of the training set?

Exploration-Exploitation Trade-off in TA

How do I pick the size of the training set?

Exploration: of the performances of the a_k on different problem instances.

Exploration-Exploitation Trade-off in TA

How do I pick the size of the training set?

Exploration: of the performances of the a_k on different problem instances.

Exploitation: of the best algorithm/problem combinations, based on current model's predictions.

Exploration-Exploitation Trade-off in TA

How do I pick the size of the training set?

Exploration: of the performances of the a_k on different problem instances.

Exploitation: of the best algorithm/problem combinations, based on current model's predictions.

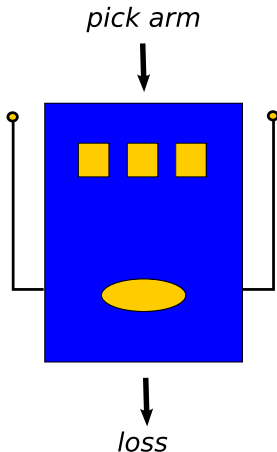
Idea: solve this trade-off in an **online** setting.

Bandit problems

A K -armed bandit (Robbins 1952) has different arms, each generating losses from different distributions.

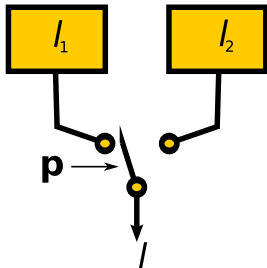
The aim of the game is to minimize the cumulative loss L , or minimize the **regret** against the best arm

$$L(M) - \min_k \sum_{j=1}^M l_k(j)$$



Bandit problems

A bandit problem solver (BPS) maps the history of observed losses to a probability distribution $\mathbf{p} = (p_1, \dots, p_K)$, from which the choice for the successive trial will be picked.



Algorithm Selection as a Bandit Problem

- ▶ K algorithms $\{a_1, \dots, a_K\}$ are the K arms
- ▶ $\mathbf{p} \in [0, 1]^K$, $\sum_k p_k = 1$
- ▶ Algorithm a_k has a finite runtime $t_k(j)$ on problem b_j
- ▶ For each incoming problem b_m
 - ▶ run algorithm k with prob. p_k
 - ▶ observe loss $t_k(j)$ if problem solved
 - ▶ update \mathbf{p} using your favorite bandit problem solver

This game implements **per set AS**: in the limit, the algorithm that performs best on all problems is selected. In practice this can be much less efficient than **per instance AS**.

TA selection as a bandit problem

- ▶ What I would like is to be able to decide online whether the model is ready for use or not.

TA selection as a bandit problem

- ▶ What I would like is to be able to decide online whether the model is ready for use or not.
- ▶ I don't need to select among K algorithms, but to select among two **time allocators** (UNIFORM and $TA_{\mathcal{M}}$).

TA selection as a bandit problem

- ▶ What I would like is to be able to decide online whether the model is ready for use or not.
- ▶ I don't need to select among K algorithms, but to select among two **time allocators** (UNIFORM and $TA_{\mathcal{M}}$).
- ▶ If $TA_{\mathcal{M}}$ eventually converges to a good performance, the BPS will start exploit it

TA selection as a bandit problem

- ▶ What I would like is to be able to decide online whether the model is ready for use or not.
- ▶ I don't need to select among K algorithms, but to select among two **time allocators** (UNIFORM and $TA_{\mathcal{M}}$).
- ▶ If $TA_{\mathcal{M}}$ eventually converges to a good performance, the BPS will start exploit it
- ▶ Worst case: If $TA_{\mathcal{M}}$ does not improve, the uniform share will be used instead

GAMBLETA

- ▶ N algorithms $\{a_1, \dots, a_N\}$
- ▶ 2 time allocators: uniform and model-based.
- ▶ $\mathbf{p} \in [0, 1]^N$, $\sum_n p_n = 1$
- ▶ Time allocator TA_k has a finite runtime $t_k(m)$ on problem b_m
- ▶ For each incoming problem b_m
 - ▶ pick TA_k with prob. p_k
 - ▶ observe loss $t_k(m)$
 - ▶ update \mathbf{p} using your favorite BPS

More TAs can be added as additional “arms”

Roadmap

Introduction

Algorithm Portfolios

Dynamic Allocation

Censored Sampling

Online Learning

Experiments

Conclusions

Solver Competitions

- ▶ A set of competing algorithms $\mathcal{A} = \{a_1, a_2, \dots, a_N\}$
- ▶ A set of problem instances $\mathcal{B} = \{b_1, b_2, \dots, b_M\}$
- ▶ A timeout t_{\max} for each instance

The winner is decided based on several criteria, including the number of instances solved, their difficulty, the time spent.

The detailed results are made public, and often offer an **interesting algorithm selection benchmark** (Petrik '06, Streeter '07)

Comparison terms

N algorithms, M instances. a_n solves b_m in a time $t_n(m)$

- ▶ ORACLE: an ideal selector, with foresight of runtimes, running only the fastest algorithm independently for each instance (per instance best)

$$t_O(\mathcal{B}) = \sum_{m=1}^M \min_n \{t_n(m)\}$$

- ▶ WINNER: the algorithm which solves all instances in the least time (per set best).

$$t_W(\mathcal{B}) = \min_n \left\{ \sum_{m=1}^M t_n(m) \right\}$$

- ▶ UNIFORM: a resource sharing portfolio of all N algorithms in parallel, with equal share $\mathbf{s}_U = (1/N, \dots, 1/N)$

$$t_U(m) = N t_O(m) \quad \forall m$$

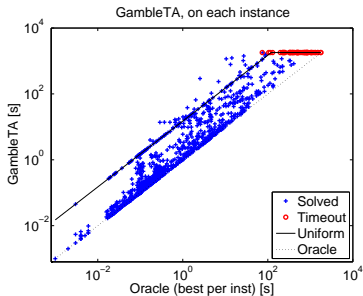
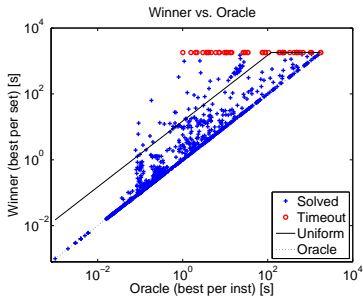
Comparison terms

- ▶ OFFG-ORACLE: Per set greedy task-switching schedule, based on prior knowledge of runtimes. (Streeter et al '07)
- ▶ ONG-EXP3: Per set greedy task-switching schedule, approximated online. (Streeter et al '07)

GAMBLETA

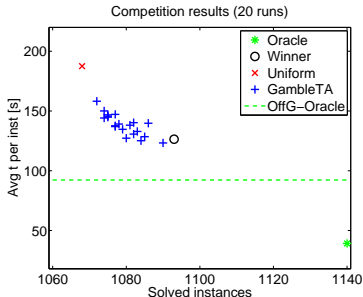
- ▶ EXP3LIGHT-A as BPS
- ▶ Per set RTD models $F(t_n)$
- ▶ Three dynamic TAs from (Gagliolo et al '06)
- ▶ Greedy task-switching schedule from (Streeter et al '07)

Solver Competitions



CPAI'06, Binary ext., 15 algorithms, 1140 instances, timeout 1800 s.
WINNER: VALCSP3.

Solver Competitions

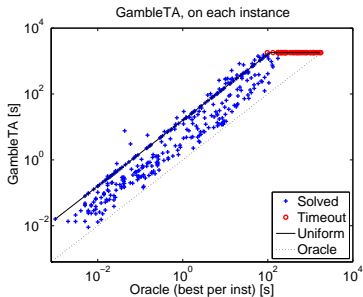
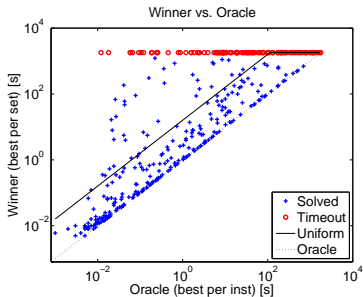


	solved	avg. t
ORACLE	1140	39.2
WINNER	1093	126.4
UNIFORM	1068	187.5
GAMBLETA	1077.6	141.9
OFFG-ORACLE	—	92

On 20 runs: won 0, WTU 0.

CPAI'06, Binary ext., 15 algorithms, 1140 instances, timeout 1800 s.
WINNER: VALCSP3.

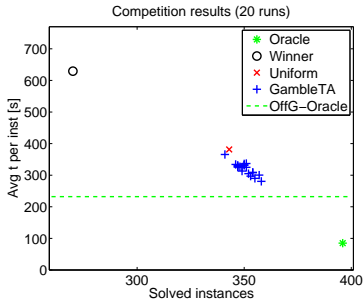
Solver Competitions



PB'07, Opt. small ints., 16 algorithms, 396 instances, timeout 1800 s.

WINNER: `bsol03`.

Solver Competitions

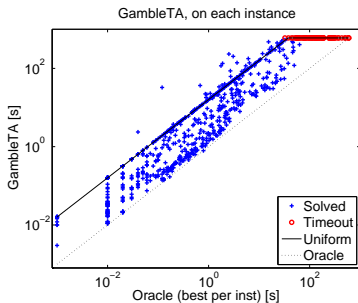
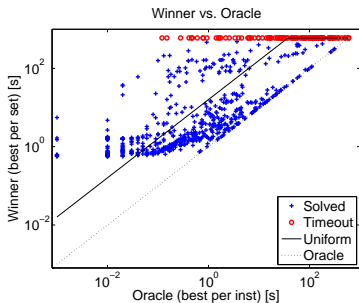


	solved	avg. t
ORACLE	396	85.0
WINNER	270	629.3
UNIFORM	343	381.5
GAMBLETA	349.0	327.3
OFFG-ORACLE	—	232

On 20 runs: won 20, WTU 1.

PB'07, Opt. small ints., 16 algorithms, 396 instances, timeout 1800 s.
WINNER: `bsol03`.

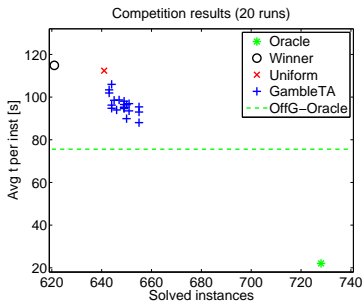
Solver Competitions



QBFEVAL'07, Formal verification, 16 algorithms, 728 instances, timeout 600 s.

WINNER: AQME-C4.5.

Solver Competitions



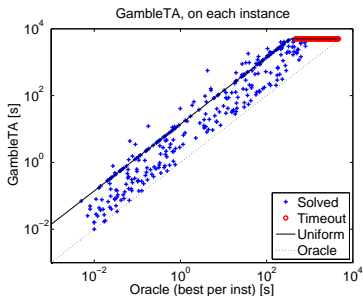
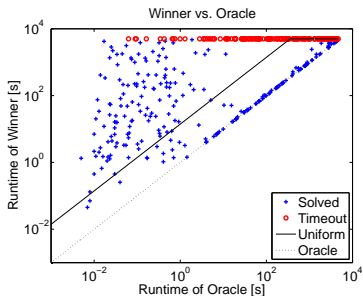
	solved	avg. t
ORACLE	728	22.1
WINNER	621	114.9
UNIFORM	641	112.4
GAMBLETA	647.0	97.8
OFFG-ORACLE	—	76

On 20 runs: won 20, WTU 0.

QBFEVAL'07, Formal verification, 16 algorithms, 728 instances, timeout 600 s.

WINNER: AQME-C4.5.

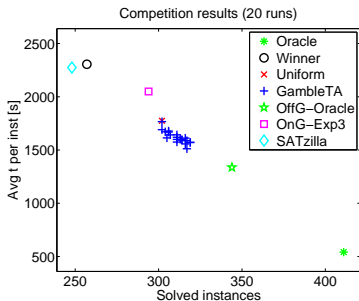
Solver Competitions



SAT'07, Random, 14 algorithms, 411 instances, timeout 5000 s.

WINNER: March KS. (Competition winner: **SATZILLA**)

Solver Competitions



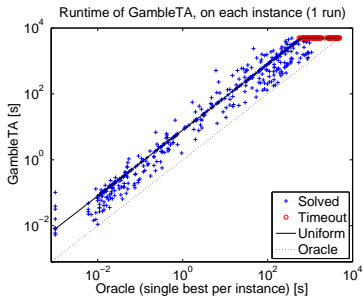
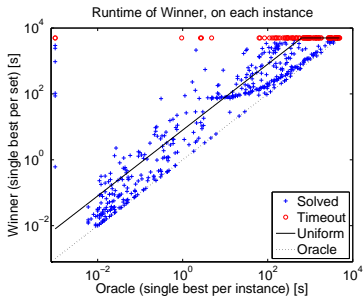
	solved	avg. t
ORACLE	411	540.8
WINNER	257	2305.4
UNIFORM	302	1774.9
GAMBLETA	309.0	1639.6
OFFG-ORACLE	344	1337
ONG-EXP3	294	2050
SATZILLA	248	2274.5

On 20 runs: won 20, WTU 0.

SAT'07, Random, 14 algorithms, 411 instances, timeout 5000 s.

WINNER: March KS. (Competition winner: SATZILLA)

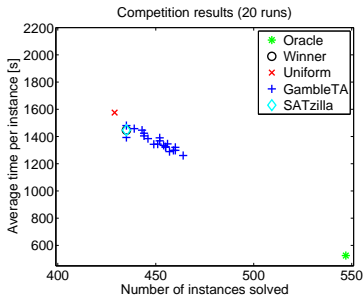
Solver Competitions



SAT'09, Random, 8 algorithms, 547 instances, timeout 5000 s.

WINNER: SATzilla.

Solver Competitions



	solved	avg. t
ORACLE	547	480.1
WINNER	435	1446.2
UNIFORM	435	1511.8
GAMBLETA	447.4	1383.6
SATZILLA	435	1446.2

On 20 runs: won 19, WTU 0.

SAT'09, Random, 8 algorithms, 547 instances, timeout 5000 s.

WINNER: SATzilla.

Roadmap

Introduction

Algorithm Portfolios

Dynamic Allocation

Censored Sampling

Online Learning

Experiments

Conclusions

Conclusions

- ▶ a general framework for online time allocation
- ▶ a set of heuristic model-based time allocators
- ▶ on top of it, a principled exploration-exploitation balance
- ▶ bounds on regret w.r.t. best TA
- ▶ almost no parameters
- ▶ comparable to or better than UNIFORM, WINNER, ONG-EXP3

Conclusions

Other contributions:

- ▶ Extension to multiple processors
- ▶ Online restart strategies (GAMBLER)
- ▶ A BPS for games with unbounded losses (EXP3LIGHT-A)

Future work:

- ▶ Prune \mathcal{A}
- ▶ Feature selection
- ▶ Time-varying covariates
- ▶ Extension to optimization problems