

Algorithm Selection as a Bandit Problem with Unbounded Losses

Matteo Gagliolo Jürgen Schmidhuber

IDSIA, Lugano, Switzerland

University of Lugano, Faculty of Informatics, Switzerland

Learning and Intelligent OptimizationN 4
21st January 2010, Venice, Italy

Why Algorithm Selection?

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

Core idea: automatically select the “best” of a finite set of algorithms $\mathcal{A} = \{a_1, a_2, \dots, a_N\}$, to solve a set of problem instances $\mathcal{B} = \{b_1, b_2, \dots, b_M\}$ (Rice '76)

More general: *allocate computation time* to the elements of \mathcal{A} .

Las Vegas Algorithms

We consider only generalized LVAs (Hoos et al. '04)

- ▶ Runtime is a random variable $T \in [0, \infty]$
- ▶ If $T < \infty$, the solution is correct

Example problems

- ▶ **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: **best = fastest**

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

Algorithm Portfolios

$\mathcal{A} = \{a_1, a_2, \dots, a_N\}$ solve the **same** problem instance b , running on the **same** processor. As soon as one algorithm solves the instance, stop all the others.

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

Alternative schedules:

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

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)

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)

Our main contribution (1 + 2):

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

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.

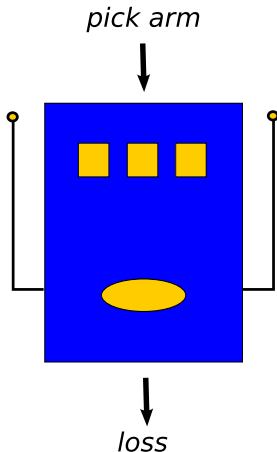
Idea: solve this trade-off in an **online** setting.

Bandit problems

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

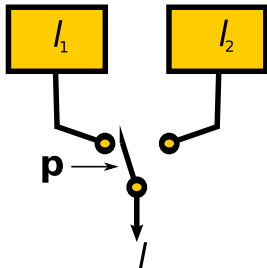
The aim of the game is to minimize the cumulative loss L , or minimize the **regret** against the best arm

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



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.
- ▶ $\mathbf{p} \in [0, 1]^2$, $\sum_n p_n = 1$
- ▶ 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”

Non-stochastic Bandits

Worst case hypothesis for a bandit problem: based on the game so far, an **adversary** sets the losses for the next trial.

Even so, a **bound** on expected regret (on M trials) can be guaranteed.

Example: EXP3LIGHT (Cesa-Bianchi et al. '05) has bound

$$E\{L(M) - L^*\} \leq O(K\sqrt{L^* \log M})$$

Issue: requires $l \in [0, 1]$.

Runtime as loss: $l \in [0, \mathcal{L}]$, \mathcal{L} **unknown**

Unbounded losses

- ▶ **Idea**: normalize losses dividing by a very large \mathcal{L} .
- ▶ **Issue**: runtime values may vary across orders of magnitudes. Difficult to choose the right \mathcal{L} in advance.
 - ▶ \mathcal{L} too small: the bound does not hold.
 - ▶ \mathcal{L} too large: BPS convergence too slow.
- ▶ **Solution**: estimate \mathcal{L} with a doubling trick. Analogous to (Cesa-Bianchi et al. '05) (full information game).

Exp3Light-A

A solver for bandit problems with partial information and an *unknown* (but finite) bound on losses.

K arms, M trials,

losses $l_j(i) \in [0, \mathcal{L}] \forall i = 1, \dots, M, j = 1, \dots, K$

unknown $\mathcal{L} < \infty$

initialize epoch $u = 0$, EXP3LIGHT (K, M)

for each trial $i = 1, \dots, M$ **do**

pick arm $I(i) = j$ with probability $p_j(i)$ from EXP3LIGHT.

incur loss $l_E(i) = l_{I(i)}(i)$.

if $l_{I(i)}(i) > 2^u$ **then**

start next epoch $u = \lceil \log_2 l_{I(i)}(i) \rceil$

restart EXP3LIGHT ($K, M - i$)

else

update EXP3LIGHT with loss $(l_{I(i)}(i)/2^u) \in [0, 1]$ for arm $I(i)$

end if

end for

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)

In the following: WINNER is the algorithm which solved most instances (in the least time)

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 $\mathbf{s}_U = (1/N, \dots, 1/N)$

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

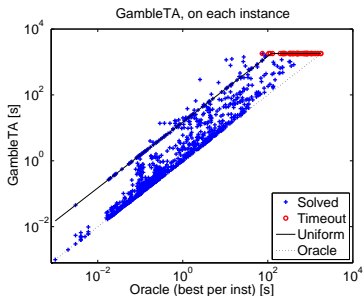
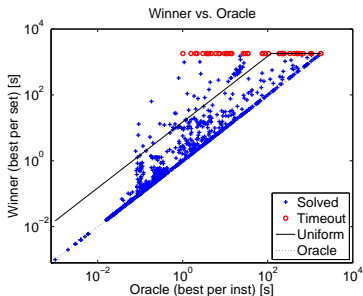
Comparison terms

- ▶ 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

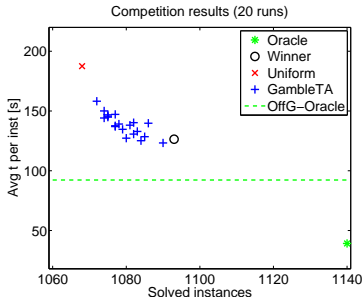
- ▶ EXP3LIGHT-A as BPS
- ▶ Per set RTD models $F(t_n)$
- ▶ Three dynamic TAs from (Gagliolo et al '06)
- ▶ Greedy task-switching schedule from (Streeter et al '07)

Solver Competitions



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

Solver Competitions

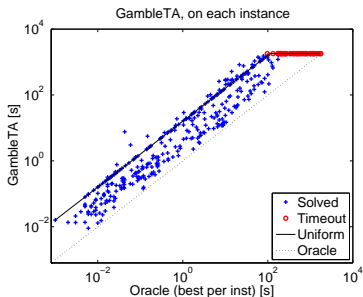
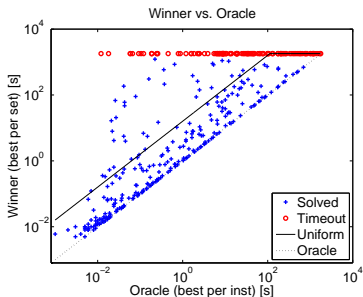


	solved	avg. t	SU	OVH
ORACLE	1140	39.2	3.23	0
WINNER	1093	126.4	1	2.23
UNIFORM	1068	187.5	0.67	3.78
GAMBLETA	1077.6	141.9	0.89	2.62
OFFG-ORACLE	—	92	1.37	1.35

On 20 runs: won 0, WTU 0.

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

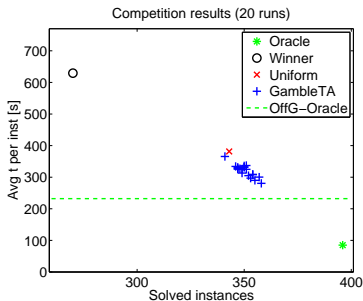
Solver Competitions



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

WINNER: `bsol03`.

Solver Competitions

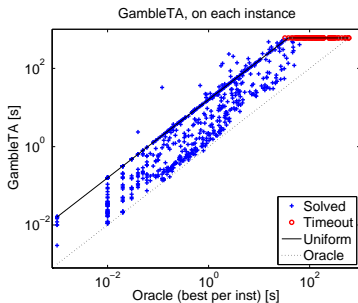
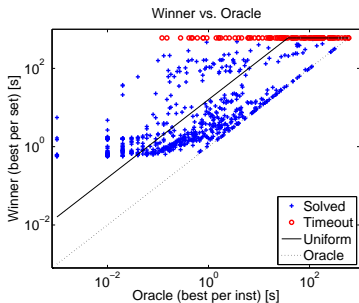


	solved	avg. t	SU	OVH
ORACLE	396	85.0	7.40	0
WINNER	270	629.3	1	6.40
UNIFORM	343	381.5	1.65	3.49
GAMBLETA	349.0	327.3	1.93	2.85
OFFG-ORACLE	—	232	2.71	1.73

On 20 runs: won 20, WTU 1.

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

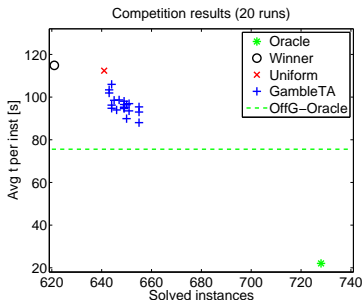
Solver Competitions



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

WINNER: AQME-C4.5.

Solver Competitions



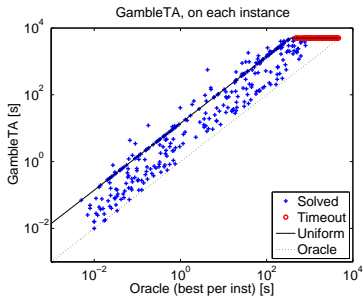
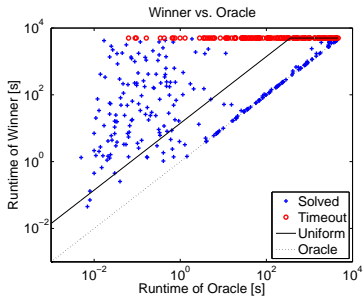
	solved	avg. t	SU	OVH
ORACLE	728	22.1	5.20	0
WINNER	621	114.9	1	4.20
UNIFORM	641	112.4	1.02	4.09
GAMBLETA	647.0	97.8	1.18	3.43
OFFG-ORACLE	—	76	1.52	2.42

On 20 runs: won 20, WTU 0.

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

WINNER: AQME-C4.5.

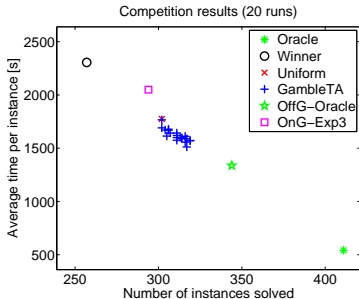
Solver Competitions



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

WINNER: March KS.

Solver Competitions



	solved	avg. t	SU	OVH
ORACLE	411	540.8	4.26	0
WINNER	257	2305.4	1	3.26
UNIFORM	302	1774.9	1.30	2.28
GAMBLETA	309.0	1639.6	1.41	2.03
OFFG-ORACLE	344	1337	1.72	1.47
ONG-EXP3	294	2050	1.12	2.79

On 20 runs: won 20, WTU 0.

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

WINNER: March KS.

Conclusions

- ▶ A BPS for non-stochastic games with unbounded losses
- ▶ A general framework for online time allocation
- ▶ Bounds on worst-case performance
- ▶ Per set experiments: Comparable to or better than UNIFORM, WINNER, ONG-EXP3

Conclusions

- ▶ A BPS for non-stochastic games with unbounded losses
- ▶ A general framework for online time allocation
- ▶ Bounds on worst-case performance
- ▶ Per set experiments: Comparable to or better than UNIFORM, WINNER, ONG-EXP3

Future work

- ▶ evaluate correct scores
- ▶ use instance features
- ▶ prune \mathcal{A}