

Towards Distributed Algorithm Portfolios

Matteo Gagliolo^{1,2} Jürgen Schmidhuber¹

¹IDSIA/University of Lugano, Switzerland

²IRIDIA, ULB, Brussels, Belgium

DCAI08, October 24, 2008

Algorithm selection

- ▶ Research in AI produced many different algorithms for each problem class.
- ▶ Often different algorithms are better/faster on different problem instances.
- ▶ Automated algorithm selection has long become a field of research (Rice '76).

Core idea: select the “best” of a finite set of algorithms $\{a_1, a_2, \dots, a_K\}$, for a given problem *instance*. Typically, selection is based on a model of algorithm performance.

Algorithm selection

- ▶ Research in AI produced many different algorithms for each problem class.
- ▶ Often different algorithms are better/faster on different problem instances.
- ▶ Automated algorithm selection has long become a field of research (Rice '76).

Core idea: select the “best” of a finite set of algorithms $\{a_1, a_2, \dots, a_K\}$, for a given problem *instance*. Typically, selection is based on a model of algorithm performance.

In the following: “best” = *fastest*

Algorithm Portfolios

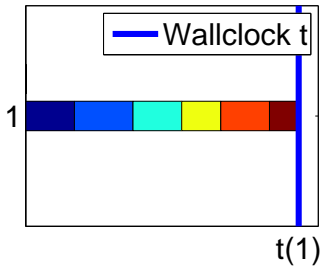
In general terms, algorithm selection can be defined as the process of allocating computational resources to a set of alternative algorithms, in order to improve some measure of performance on a set of problem instances.

- ▶ finite set \mathcal{A} of algorithms $\{a_1, a_2, \dots, a_K\}$
- ▶ solving the **same** problem instance b in parallel
- ▶ sharing a single CPU with share $\mathbf{s} \in [0, 1]^K, \sum \mathbf{s}_k = 1$
- ▶ $\forall t, t_k = s_k t$ is allocated to a_k

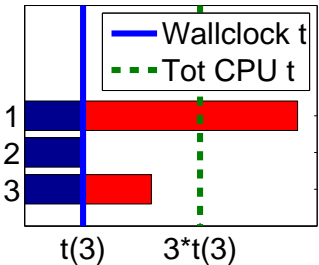
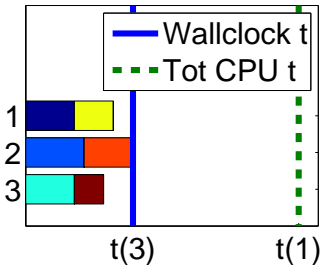
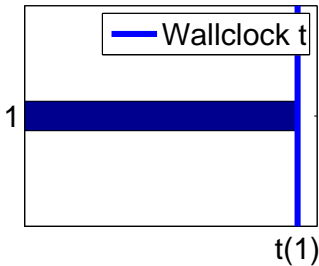
Instead of using the model of $t(a, b)$ to select a **single** algorithm, we will use it to select \mathbf{s} , i.e., **allocate time** to elements of \mathcal{A} (Huberman et al. '97, Gomes et al. '97).

Parallel Computing vs. Algorithm Portfolios

Parallel computing



Algorithm portfolio



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**:

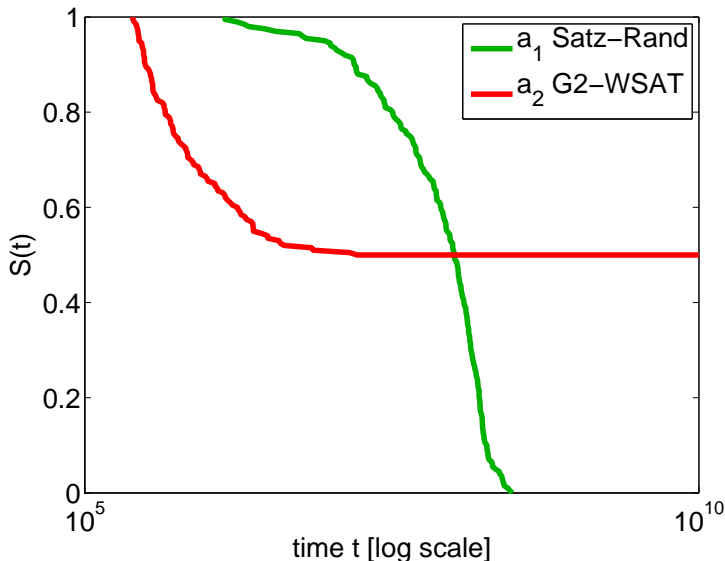
$$S(t) = 1 - F(t)$$

RTD is the natural model of performance for decision problem solvers.

SAT-UNSAT: RTD

CDF on subset (uf-250, uu-250), $F(t|n = 250)$

Survival



Heavy-tailed RTD

a_1, a_2, \dots, a_K can also be different runs of a same algorithm, with different random seeds.

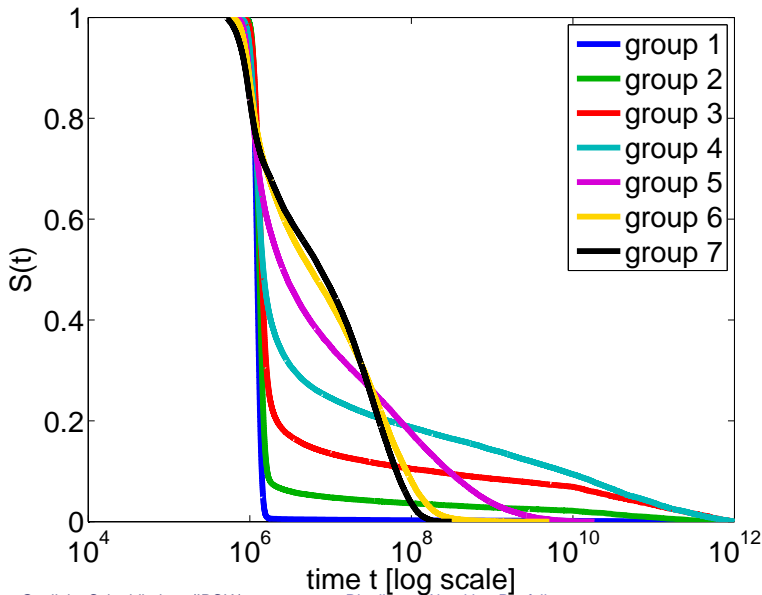
A situation in which this is particularly useful

$$S(t) \rightarrow_{t \rightarrow \infty} Ct^{-\alpha}$$

This means: most runs are short, some take a very long time... E.g., Satz-Rand on structured underconstrained problems. (Gomes et al. 2000).

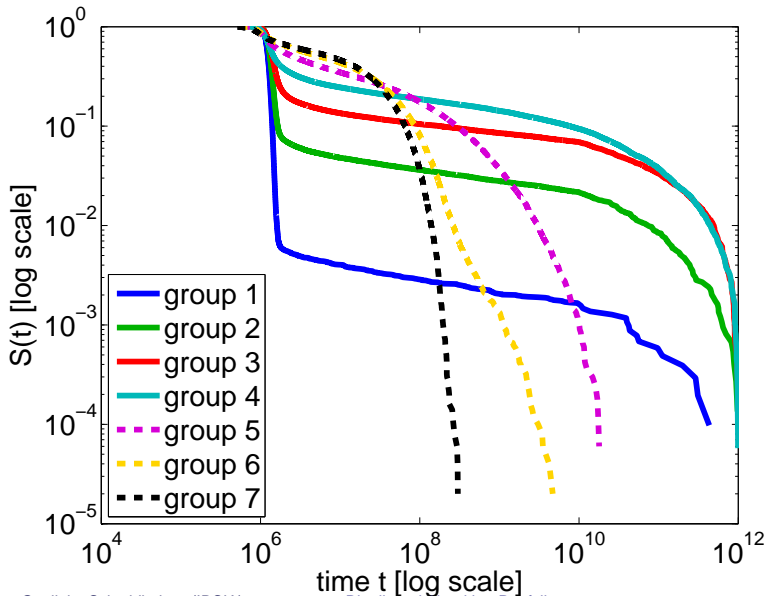
Heavy-tailed RTD

Example: Satz-Rand on structured graph coloring



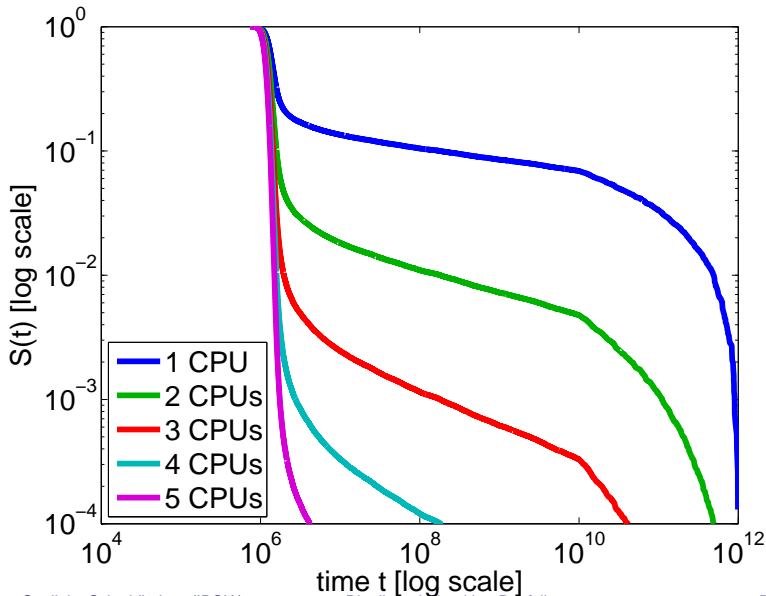
Heavy-tailed RTD

Example: Satz-Rand on structured graph coloring



Heavy-tailed RTD

Instance group 3: multiple instances of Satz-Rand in parallel



Model based time allocation

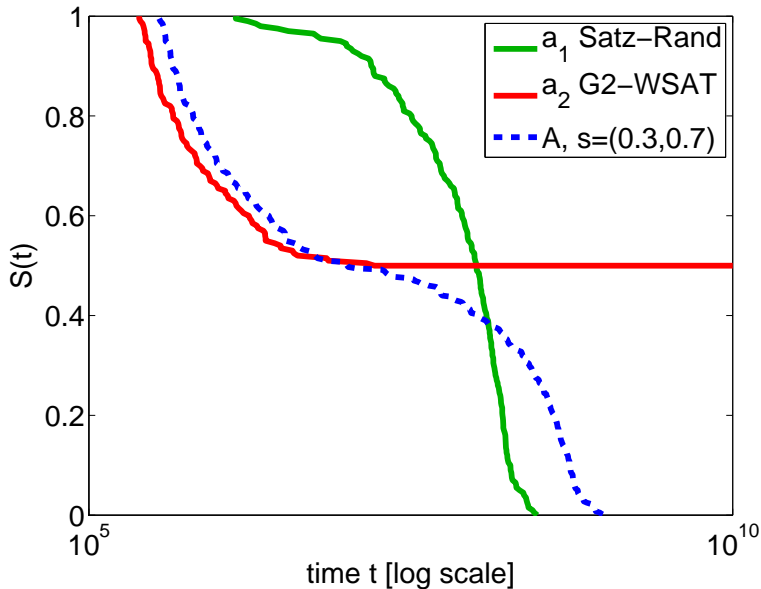
Runtime of a portfolio: $\min\{t_k/s_k\}$. Also a random variable.
Its distribution will depend on each F_k and \mathbf{s}

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

(Gagliolo et al. 06')

SAT-UNSAT: Portfolio RTD

Survival



Single CPU

K algorithms on 1 CPU: K dimensional share, summing to one:

$$\mathbf{s} = \{s_k\}, s_k \in [0, 1], \sum_{k=1}^K s_k = 1$$

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

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

(Gagliolo et al. 06')

Multiple CPUs

K algorithms on J CPUs: $K \times J$ dimensional share, with columns summing to one:

$$\mathbf{S} = \{s_{kj}\}, s_{kj} \in [0, 1], \sum_{k=1}^K s_{kj} = 1 \quad \forall j \in \{0, 1, \dots, J\}$$

The RTD of portfolio \mathcal{A} with share \mathbf{S} is:

$$S_{\mathcal{A}, \mathbf{s}}(t) = \prod_{j=1}^J \prod_{k=1}^K S_k(s_{kj}t)$$

Distributed Time Allocators

For 1 CPU:

- ▶ Minimizing expected solution time:

$$\mathbf{s}^* = \arg \min_{\mathbf{s}} \int_0^{+\infty} S_{\mathcal{A},\mathbf{s}}(t) dt.$$

- ▶ Maximizing solution probability within a given **contract** t_u :

$$\mathbf{s}^* = \arg \min_{\mathbf{s}} S_{\mathcal{A},\mathbf{s}}(t_u).$$

- ▶ 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 $(K - 1)$ dimensions.
(Gagliolo et al. 06')

Distributed Time Allocators

For J CPUs:

- ▶ Minimizing expected solution time:

$$\mathbf{s}^* = \arg \min_{\mathbf{s}} \int_0^{+\infty} S_{\mathcal{A},\mathbf{s}}(t) dt.$$

- ▶ Maximizing solution probability within a given **contract** t_U :

$$\mathbf{s}^* = \arg \min_{\mathbf{s}} S_{\mathcal{A},\mathbf{s}}(t_U).$$

- ▶ 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 $J(K - 1)$ dimensions.

Distributed Time Allocators

Definition: A distributed share \mathbf{S} is *homogeneous* if it shares machine time in the same way on each CPU (i. e., if all its columns are equal).

Distributed Time Allocators

Definition: A distributed share \mathbf{S} is *homogeneous* if it shares machine time in the same way on each CPU (i. e., if all its columns are equal).

Theorem: \forall quantile-optimal share \mathbf{S}^* there is an homogeneous equivalent optimal \mathbf{S}_h^* with the same quantile.

An analogous theorem can be proved for the contract allocator, but not for the expected time allocator.

Distributed Time Allocators

Definition: A distributed share \mathbf{S} is *homogeneous* if it shares machine time in the same way on each CPU (i. e., if all its columns are equal).

Theorem: \forall quantile-optimal share \mathbf{S}^* there is an homogeneous equivalent optimal \mathbf{S}_h^* with the same quantile.

An analogous theorem can be proved for the contract allocator, but not for the expected time allocator.

Quantile and contract TAs: Constrained function optimization in $(K - 1)$ dimensions, regardless of J .

How many CPUs?

If I have J CPUs available, should I use them all for each problem instance? Or solve J instances in parallel? Or..?

Heuristic solution:

Initialize $J_a = J$. For each problem b , asynchronously:

- ▶ evaluate the optimal shares for $j = 1, \dots, J_a$ CPUs, and the corresponding performances (e.g., quantiles)
- ▶ solve b on j^* CPUs giving the best performance, $J_a = J_a - j^*$
- ▶ $J_a = J_a + j^*$ when b solved

Experiments

Benchmarks:

- ▶ Algorithms Satz-Rand and G2WSAT on a set of sat-unsat instances ($uf-*$, $uu-*$ from SATLIB)
- ▶ Satz-Rand alone on 9 sets of Graph Coloring instances (Gent et al. '99): in this case only j is set.

Results:

- ▶ Speedup ($\text{runtime}(1)/\text{runtime}(J)$)
- ▶ Efficiency ($\text{speedup}(J)/J$)

for $J = 1, 5, 10, 15, 20$.

Experiments

Benchmarks:

- ▶ Algorithms Satz-Rand and G2WSAT on a set of sat-unsat instances ($uf-*$, $uu-*$ from SATLIB)
- ▶ Satz-Rand alone on 9 sets of Graph Coloring instances (Gent et al. '99): in this case only j is set.

Results:

- ▶ Speedup ($\text{runtime}(1)/\text{runtime}(J)$)
- ▶ Efficiency ($\text{speedup}(J)/J$)

for $J = 1, 5, 10, 15, 20$.

Note that **Speedup might be larger than J** , (and Efficiency might be larger than 1), as not all computations are carried out (portfolio effect).

Experiment Results

#CPUs	Speedup				Efficiency			
	5	10	15	20	5	10	15	20
GC 0	4.34	7.19	9.23	11.19	0.87	0.72	0.62	0.56
GC 1	1.00	279.26	83.55	85.87	0.20	27.93	5.57	4.29
GC 2	2.65	18.44	35.98	97.74	0.53	1.84	2.40	4.89
GC 3	2.59	6.55	9.80	21.28	0.52	0.66	0.65	1.06
GC 4	3.97	6.18	6.94	8.81	0.79	0.62	0.46	0.44
GC 5	3.36	3.47	7.69	7.83	0.67	0.35	0.51	0.39
GC 6	3.55	5.13	5.79	8.54	0.71	0.51	0.39	0.43
GC 7	5.93	8.27	12.65	11.95	1.19	0.83	0.84	0.60
GC 8	5.06	9.02	11.62	12.01	1.01	0.90	0.77	0.60
GC all	3.06	4.46	5.50	8.22	0.61	0.45	0.37	0.41
SAT-UNSAT	3.46	4.81	6.02	7.08	0.69	0.48	0.40	0.35

Conclusions

- ▶ A blueprint for an “intelligent” cluster front-end
- ▶ Can be fully distributed
- ▶ Heuristic choice of j
- ▶ Effective when algorithms display heavy-tailed RTD
- ▶ Ongoing: extension to *optimization* problems (quality-time trade-off)