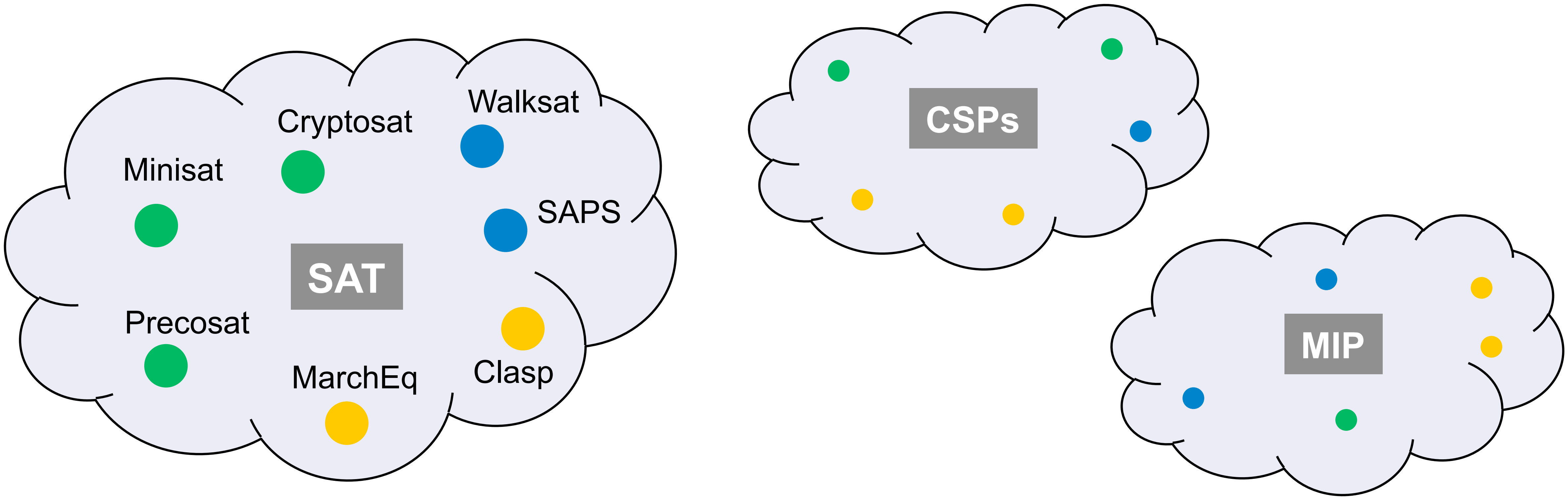
Parallel SAT Solver Selection and Scheduling

Yuri Malitsky	4C*
Ashish Sabharwal	IBM Watson
Horst Samulowitz	IBM Watson
Meinolf Sellmann	IBM Watson

*This research has been partially supported by EU FET grant ICON (project number 284715)

Algorithm Portfolio: Motivation



- Combinatorial problems such as SAT, CSPs, and MIP have several competing solvers with complementary strengths
  - Different solvers excel on different kinds of instances

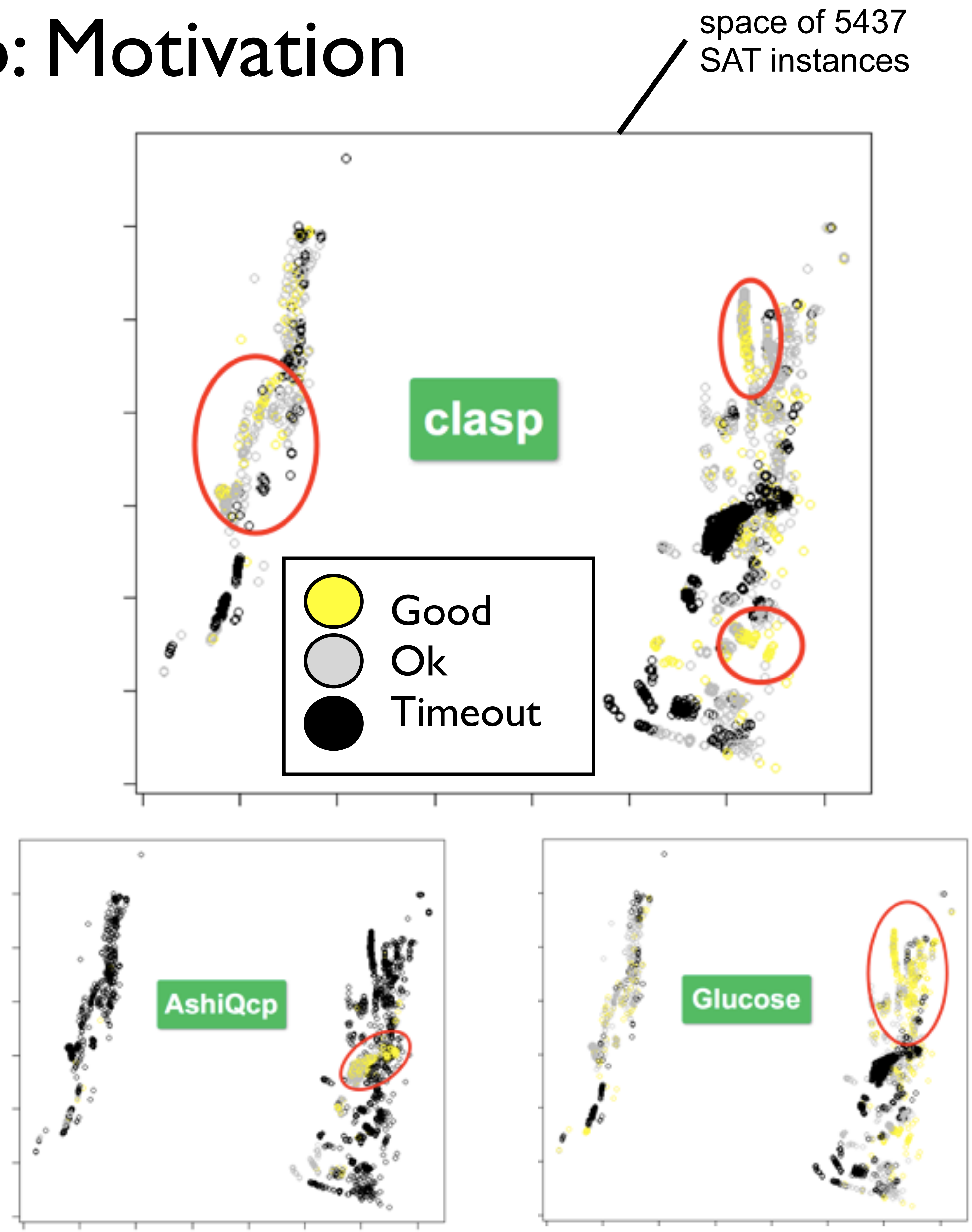
- **Ideal strategy**: given an instance, dynamically decide which solver(s) to use from a portfolio of solvers

Algorithm Portfolio: Motivation

algorithm is **good** on the instance
(≤ 25% slower than VBS)

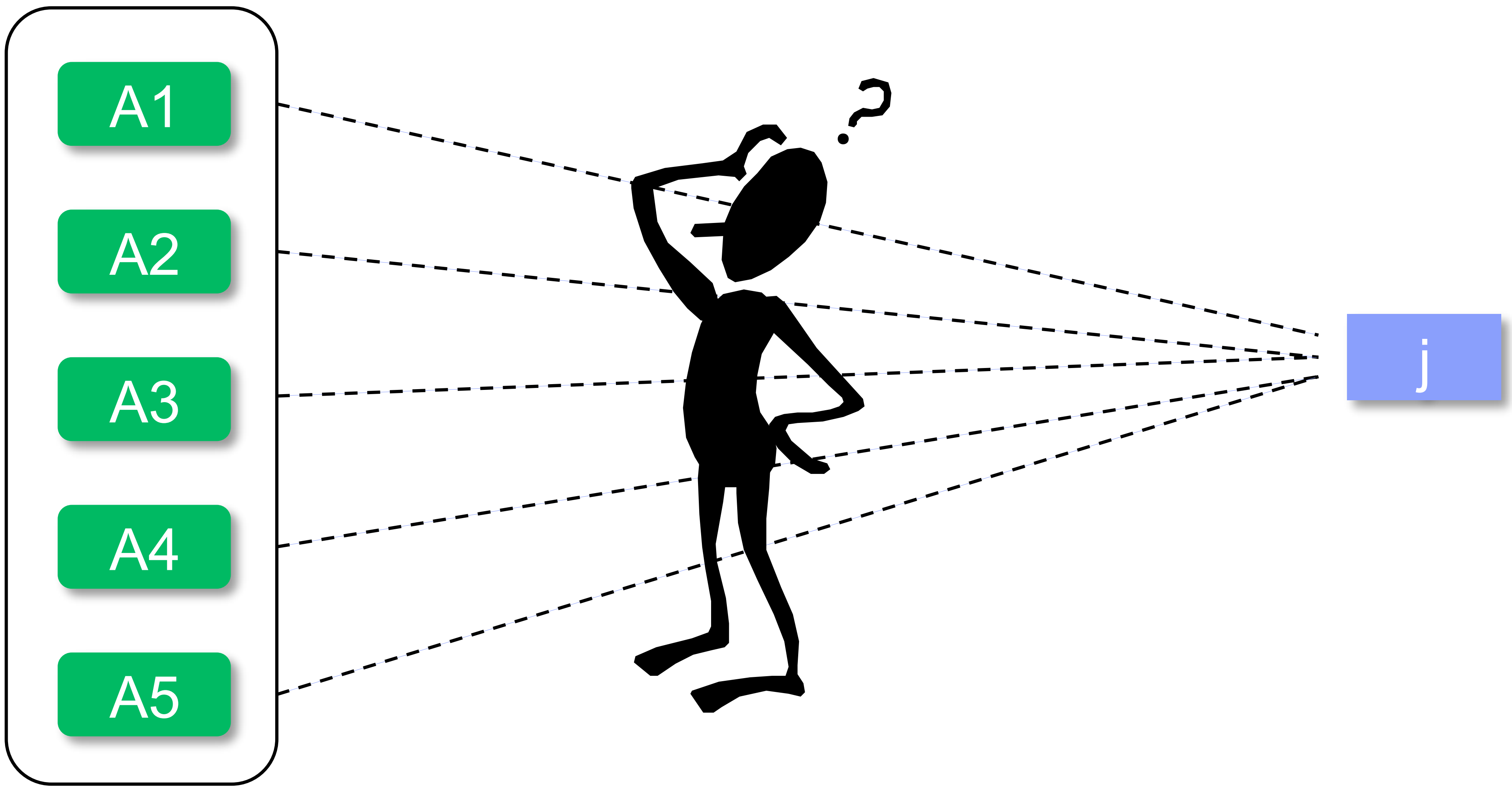
algorithm is **ok** on the instance
(> 25% slower than VBS)

algorithm is **bad** on the instance
(times out after 5000 sec)



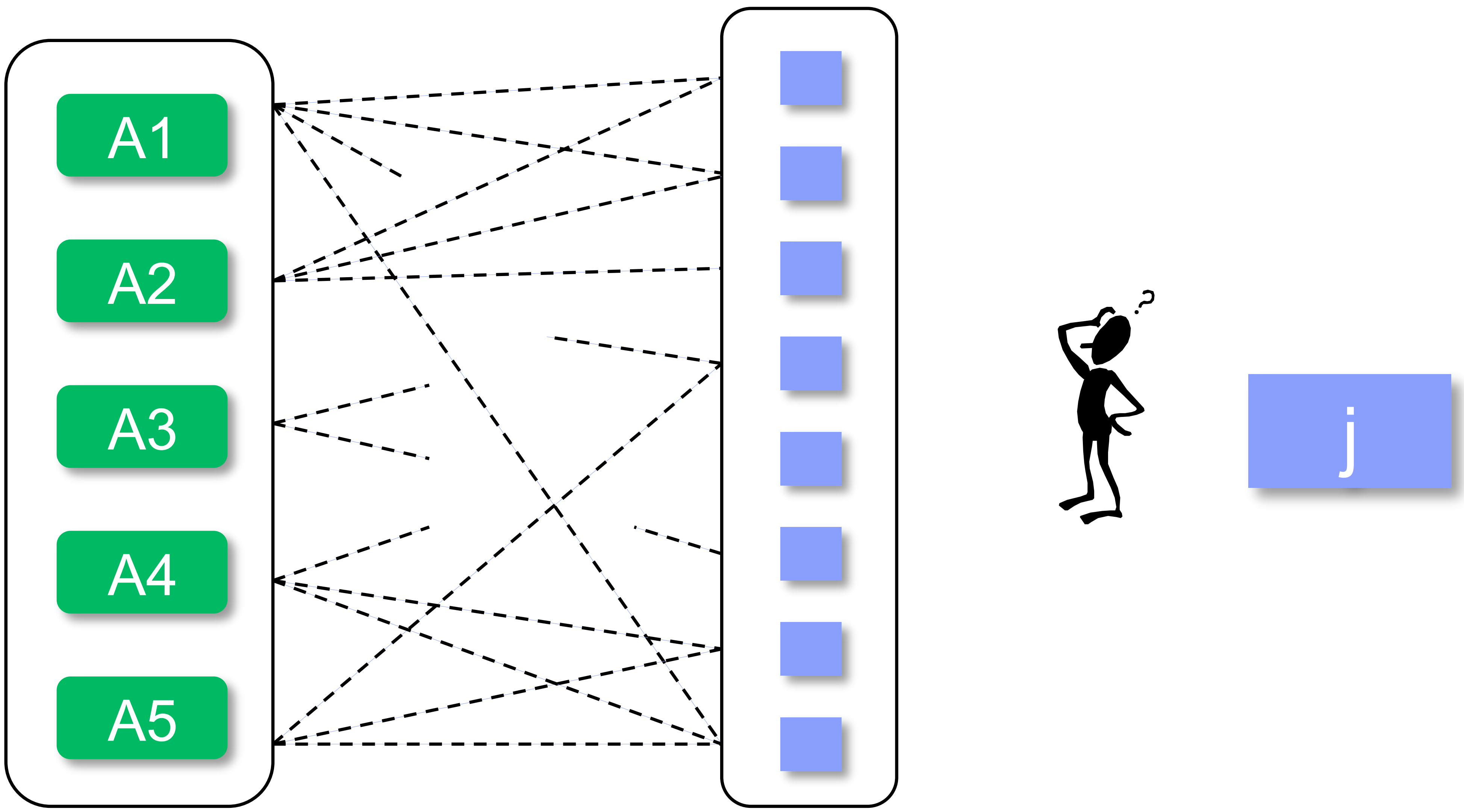
* **VBS**: Virtual Best Solver

Algorithm Portfolios: How?



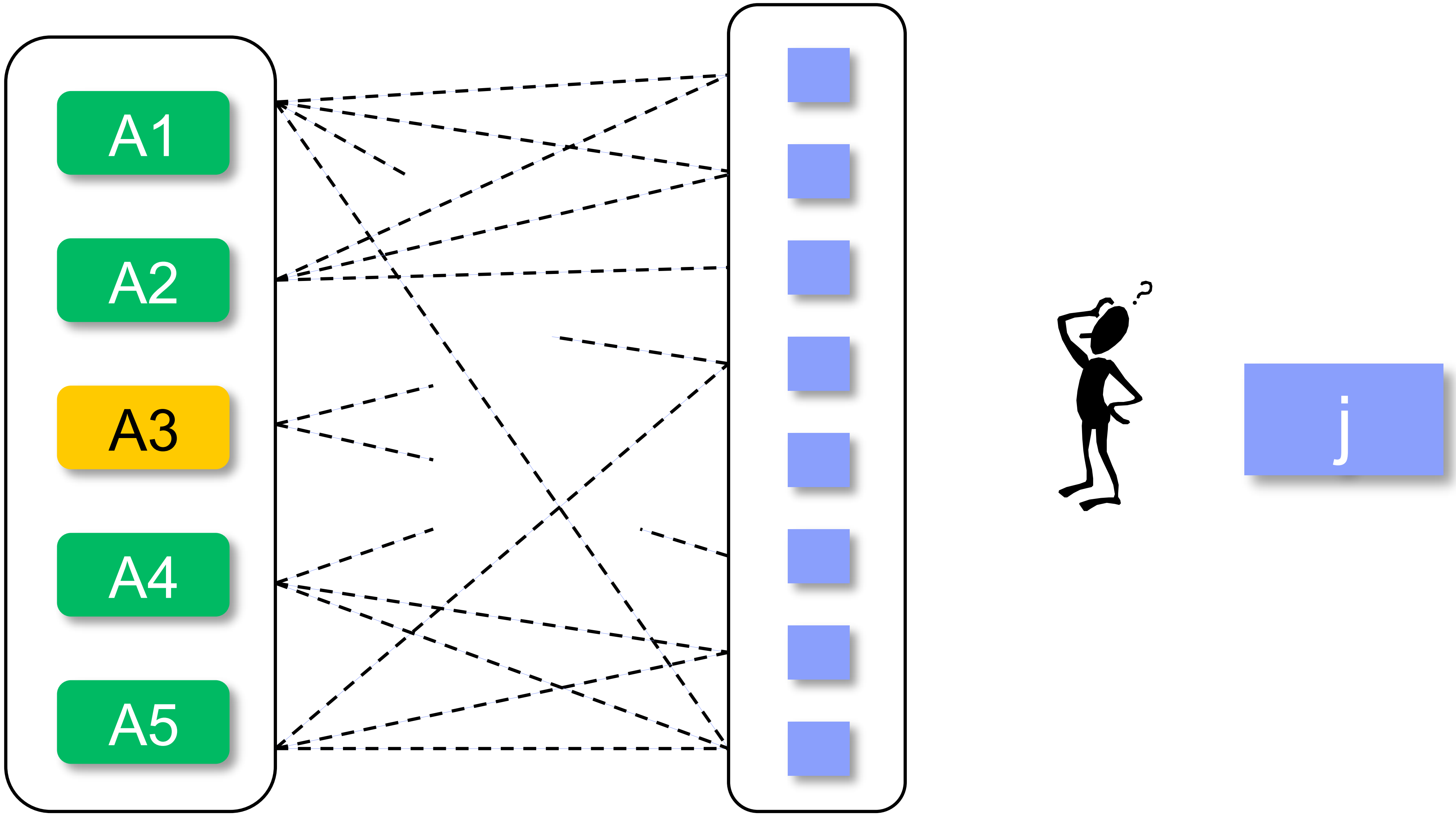
- Given a portfolios of algorithms A1, A2, …, A5, when an instance j comes along, how should we decide which solver(s) to use without actually running the solvers on j?

Algorithm Portfolios: Use Machine Learning



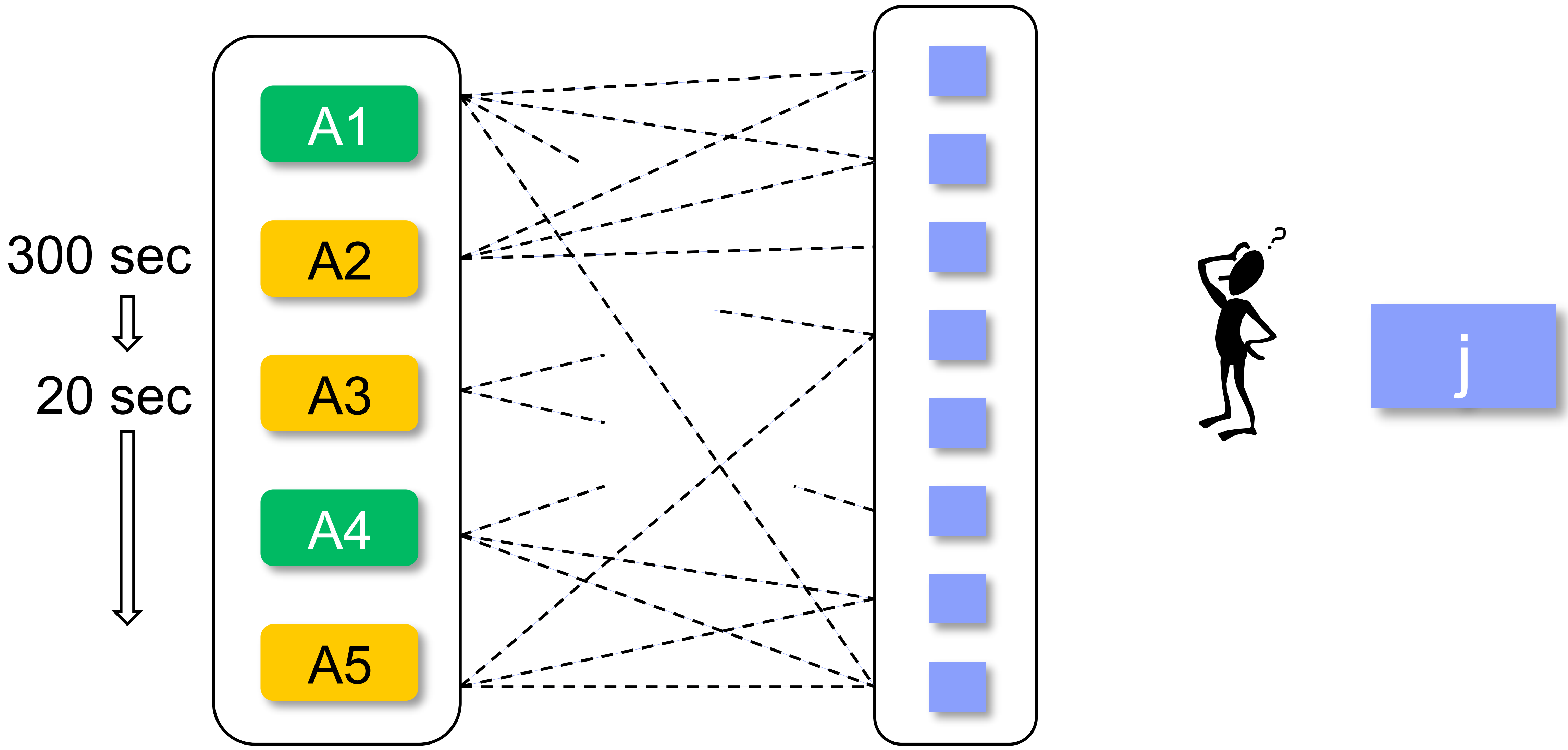
- Pre-compute how long each Ai takes on each training instance
- “Use this information” when selecting solver(s) for instance j

Flavor 1: Algorithm Selection



- Output: **one single solver** that is expected to perform the best on instance j in the given time limit T

Flavor 2: Algorithm Scheduling



- Output: **a sequence of (solver,runtime) pairs** that is expected to perform the best on instance j in total time T

3S: Semi-Static Scheduling

- **Question**: *given a set of training instances, what is the best solver schedule for these?*
- Set covering problem; can be modeled as an IP
  - binary variables $x_{S,t}$: 1 iff solver S is scheduled for time t
  - penalty variables y_i : 1 iff no selected solver solves instance i

Minimize number of unsolved instances

Minimize runtime (secondary obj.)

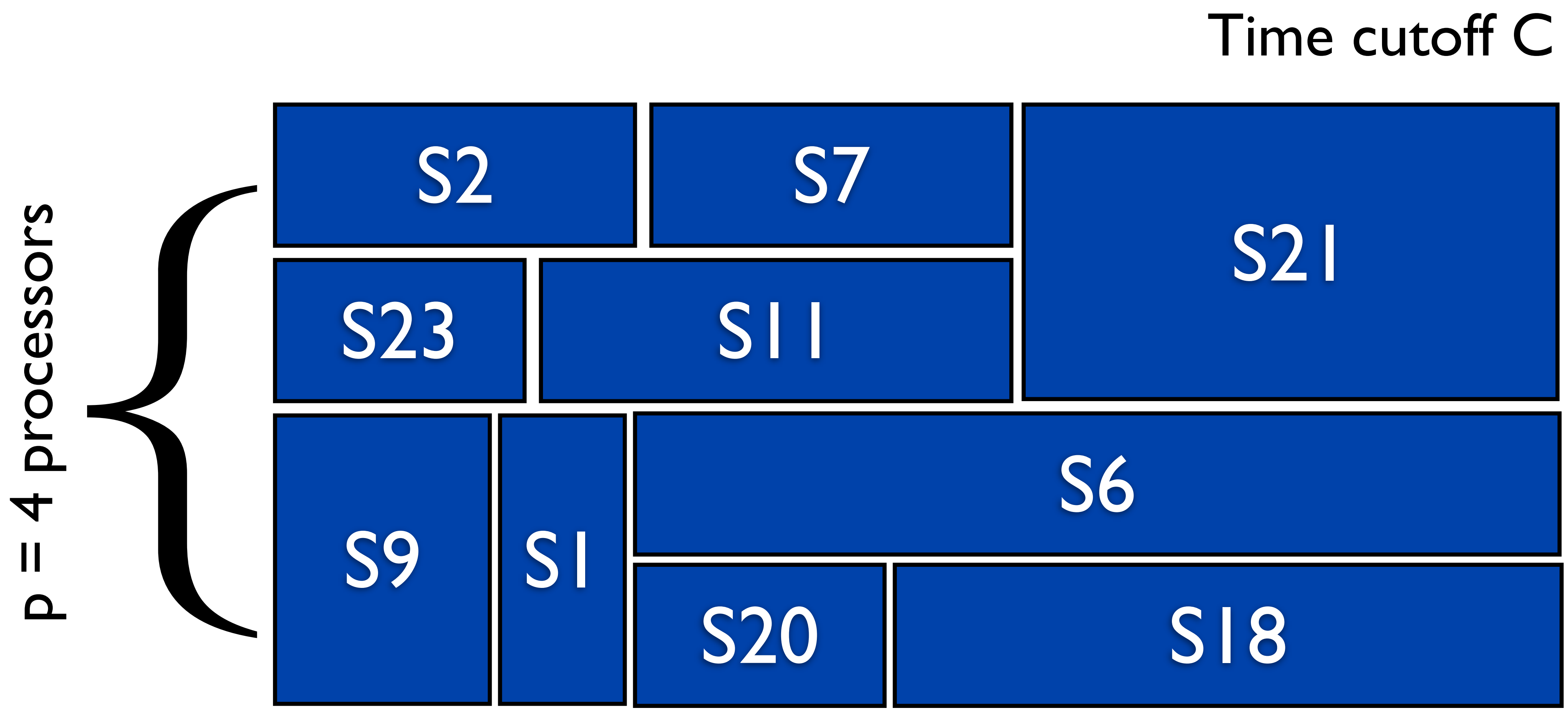
$$\begin{aligned} \min \quad & (C+1)\sum_i y_i + \sum_{S,t} t x_{S,t} & & (1) \\ s.t. \quad & y_i + \sum_{(S,t)\ \mid\ i \in V_{S,t}} x_{S,t} \geq 1 & \forall i \quad & (2) \\ & \sum_{S,t} t x_{S,t} \leq C & & (3) \end{aligned}$$

Time limit

Column Generation for Scalability

- **Issue**: poor scaling, due to too many variables
  - e.g., 30 solvers, C = 3000 sec timeout 30000 $x_{S,t}$ vars
  - even being smart about "interesting" values of t doesn't help
- **Solution**: use *column generation* to identify promising (S,t) pairs that are likely to appear in the optimal schedule
  - solve LP relaxation to optimality using column generation
  - use only the generated columns to construct a smaller IP, and solve it to optimality (no branch-and-price)
- Results in fast but still high quality solutions (empirically)

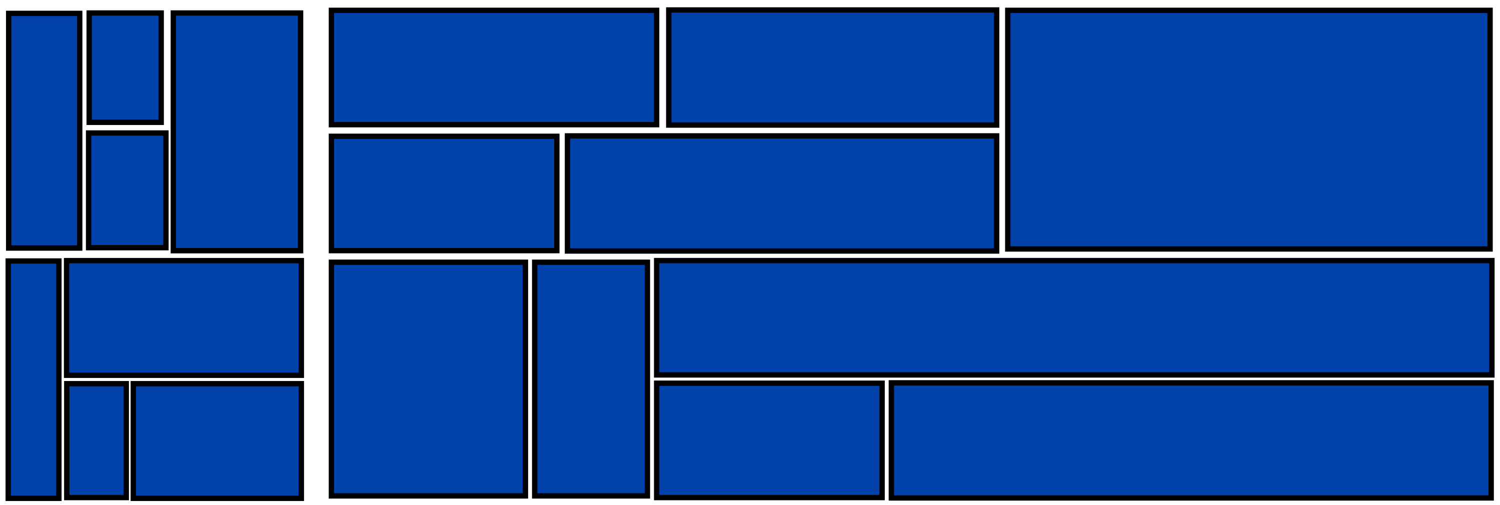
Building Parallel Portfolios



- Setting:
  - p processors
  - wall clock cutoff C
  - set of sequential and parallel solvers
  - training data (instances with solver performance)

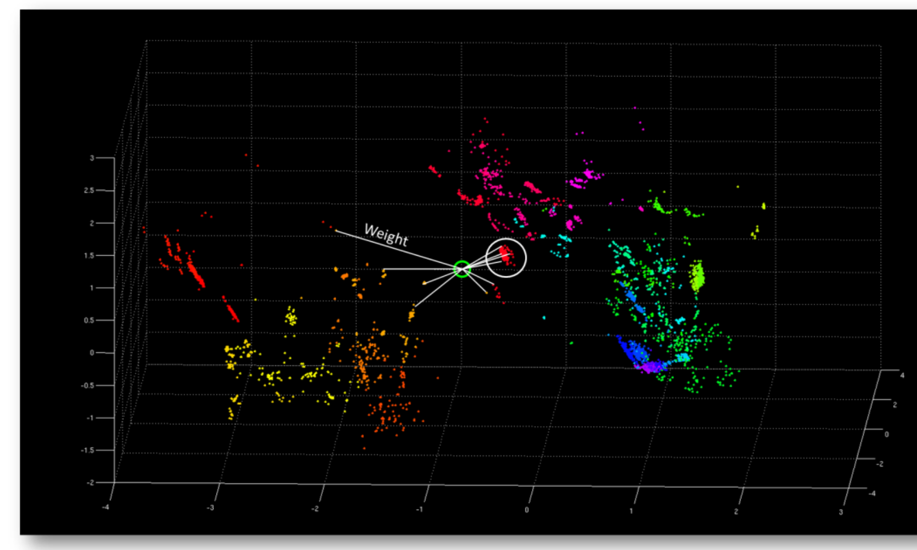
- Parallel Portfolio Design Problem
  - Given a SAT instance F, ***which solvers*** to schedule, for ***how long***, and on ***how many processors***?

Approach: Extending 3S



Static Schedule for 10% of C, computed **offline**, oblivious to F, using **all** training data

Dynamic Schedule for 90% of C, computed **online** using **k nearest neighbors** of F in the training data



- Schedules computed using **IP formulation**
  - Number of integer vars reduced using Column Generation for root LP
- **kNN** based on the 48 normalized features in Euclidean Space

Parallel Semi-Static Scheduling

Minimize number of unsolved instances

Minimize runtime (secondary objective)

Solve all Instances

Do not exceed time limit on each processor

ensures that all solvers that have a minimal solver time are included in the schedule, with an appropriate time limit

Limit number of solvers in the schedule

$$\min\ (pC+1)\sum_i y_i + \sum_{S,t,P} t n_S x_{S,t,P}$$

$$s.t.\ y_i + \sum_{(S,t)\ \mid\ i \in V_{S,t}, P \subseteq \{1,\ldots,p\}, |P|=n_S} x_{S,t,P} \geq 1 \qquad \forall i$$

$$\sum_{S,t,P \subseteq \{1,\ldots,p\}\ q \in P\ ,|P|=n_S} t x_{S,t,P} \leq C \qquad \forall q \in \{1,\ldots,p\}$$

$$\sum_{S,t,P \subseteq \{1,\ldots,p\},|P|=n_S, t \geq t_S} x_{S,t,P} \geq 1 \qquad \forall S \in M$$

$$\sum_{S,t,P \subseteq \{1,\ldots,p\},|P|=n_S} x_{S,t,P} \leq N$$

$$y_i, x_{S,t,P} \in \{0,1\} \qquad \forall i, S, t, P \subseteq \{1,\ldots,p\}, |P| = n_S$$

1 iff solver S is scheduled for time t on processor P

penalty variable is 1 iff no selected solver solves instance i

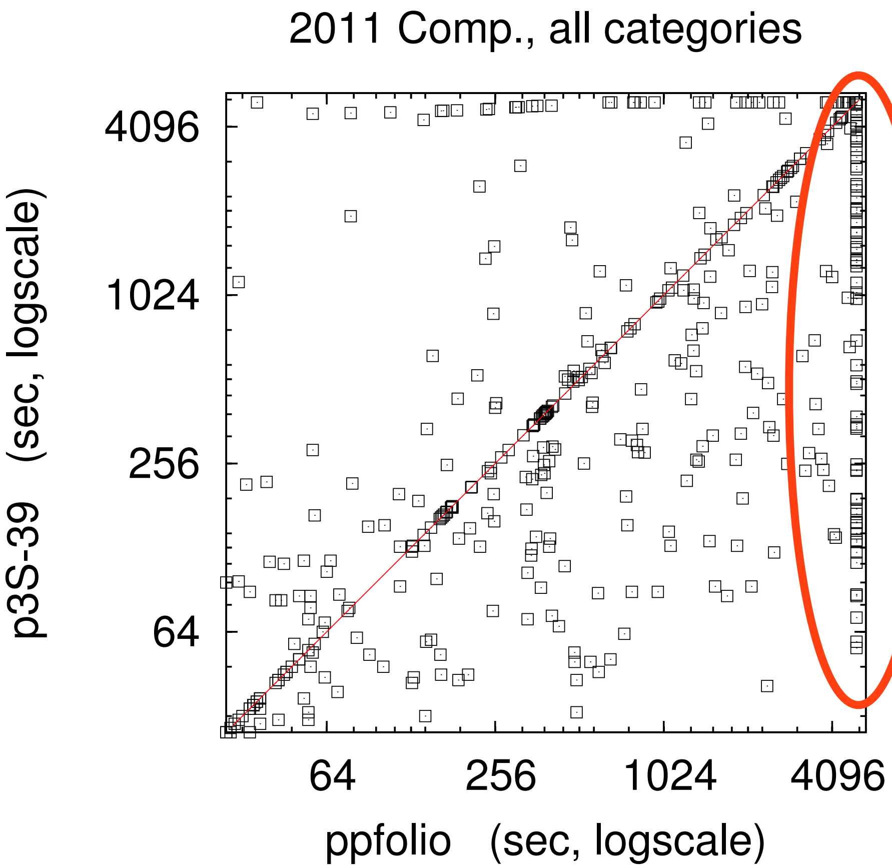
Challenges

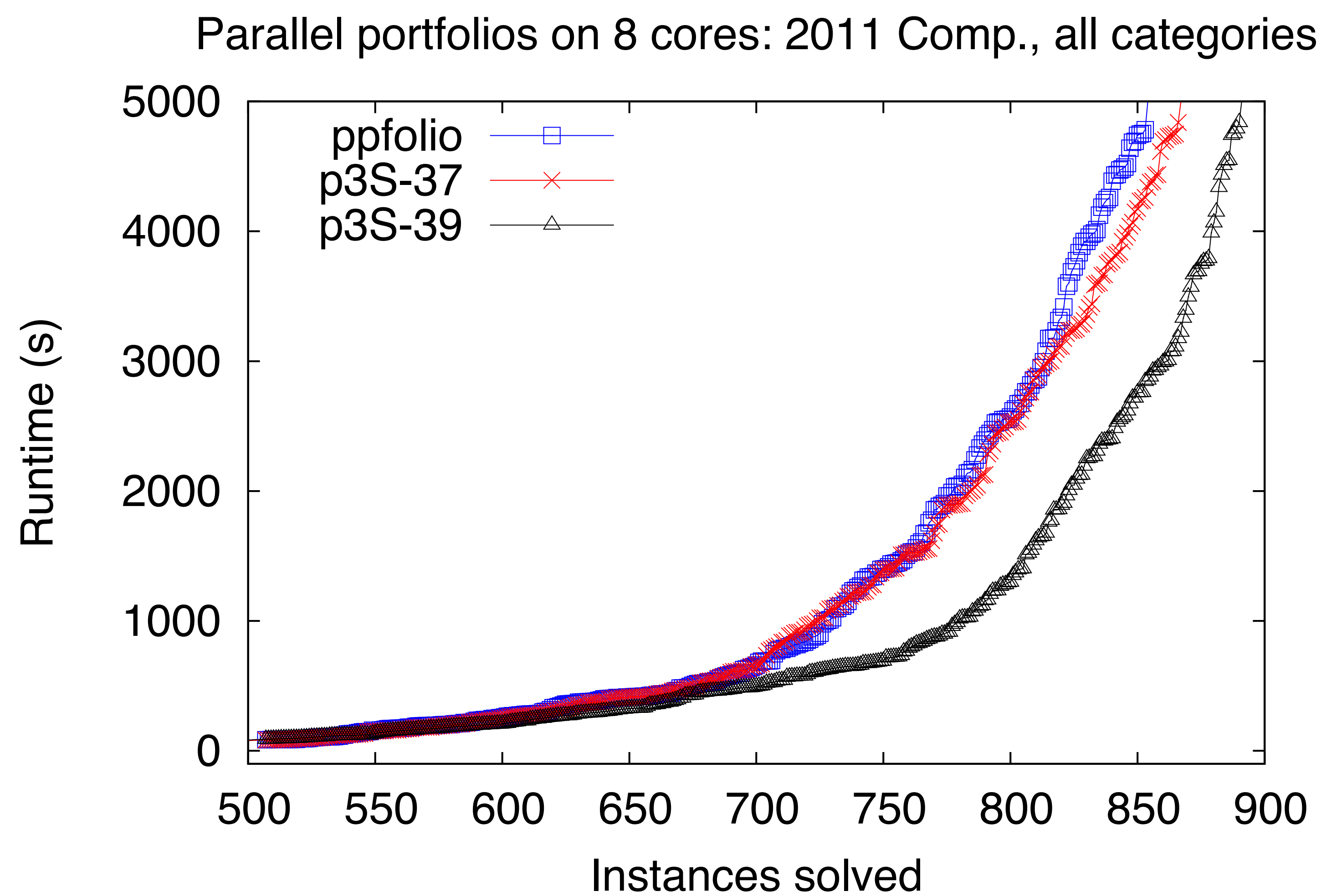
- Scheduling **IP formulation is more complex**, *esp.* with *parallel solvers*
- Unlike typical sequential scheduling, computing a *single long running solver* is insufficient => must **solve Scheduling IP online** at runtime!
  - Column Generation based variable reduction heuristic is critical
- Processor **symmetry** artificially increases search space
  - E.g., on 8 processors, 8! = 40,000 equivalent versions of every schedule => 0.5 sec optimization could take over 5 hours!
  - Column Generation naturally alleviates this problem to a large extent
- Scheduling IP may not necessarily directly generate **executable schedule**
  - Best-effort post-processing to synchronize parallel solvers on multiple cores

Experimental Results

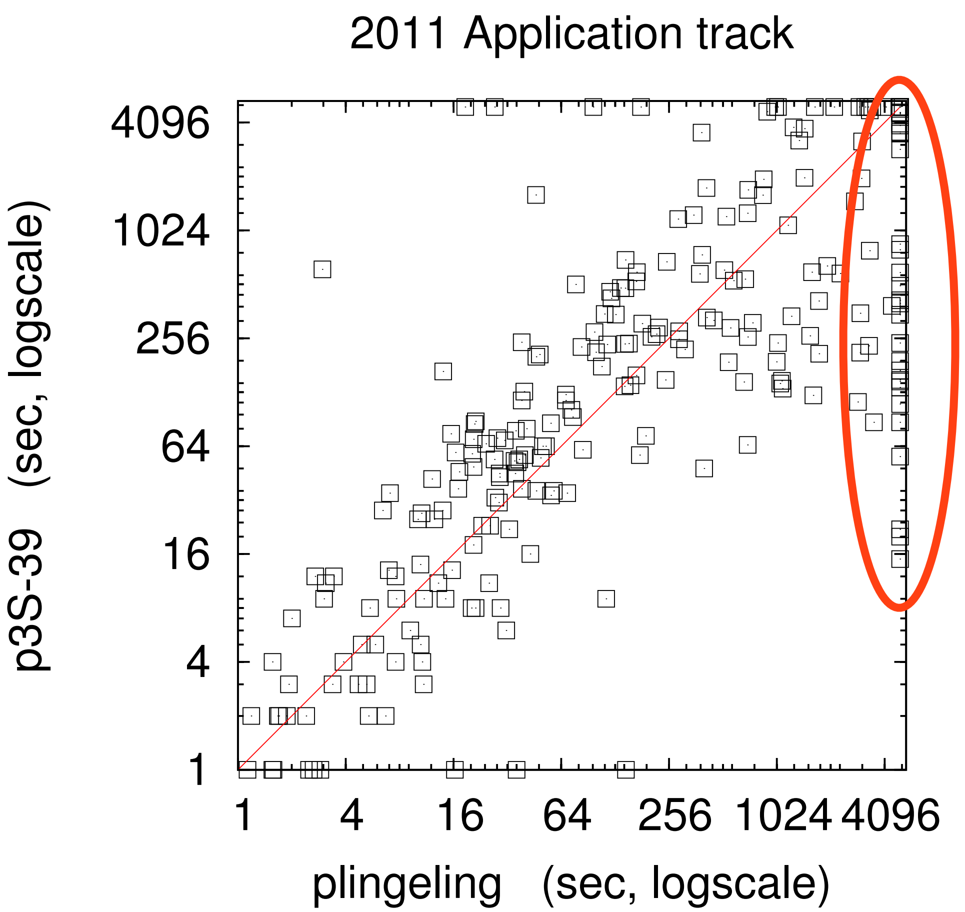
- Our parallel portfolios
  - p3S-37: 37 sequential constituent solvers
  - p3S-39: 37 sequential and 2 **parallel** solvers
    - cryptominisat 2.9.0 and plingeling 276
- Comparison against 2011 winners
  - parallel portfolio: **ppfolio**
  - parallel solver: **plingeling**
- Instances:
  - Training: 5,466 from SAT Competitions and Races 2002-2010
  - Testing: 1,200 from SAT 2011 Competition

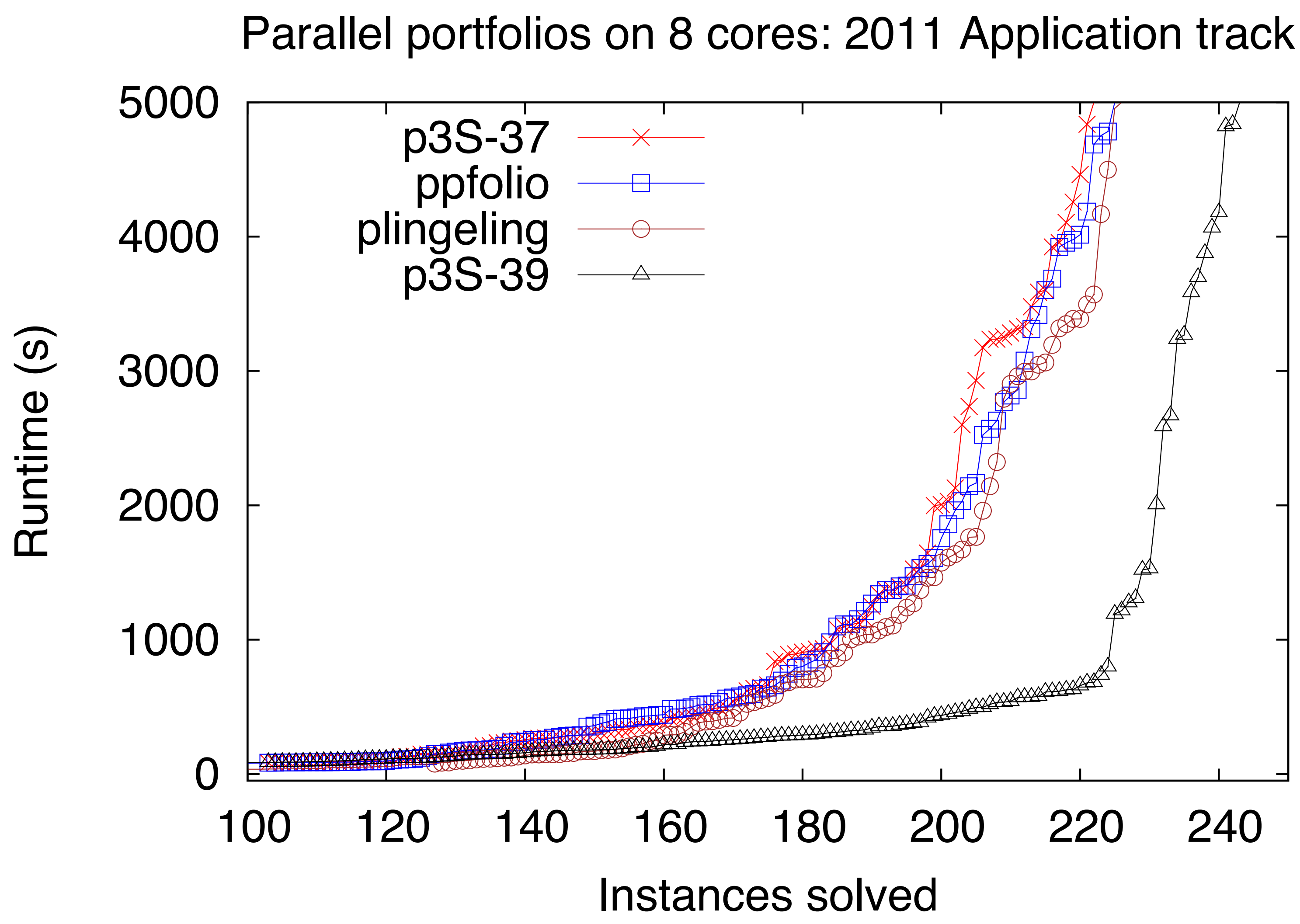
Experimental Results: All Categories





Experimental Results: Application Track





Summary

- Introduced a novel method for devising ***dynamic parallel solver portfolios that accommodate parallel solvers***

- Produce parallel solver schedules at **runtime**

- **p3S** is a highly competitive solver

- Combining **Machine Learning** and **OR** technologies to create a better solver