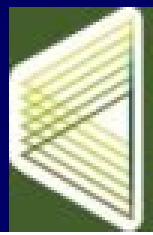
Memetic Pittsburgh Learning Classifier Systems

Jaume Bacardit
School of Computer Science
School of Biosciences
University of Nottingham

jaume.bacardit@nottingham.ac.uk

asap research
automated scheduling optimisation & planning



Multi-disciplinary Centre for Integrative Biology

The University of Nottingham

Outline

- Introduction
- Framework: GAssist LCS
- Development of MPLCS
  - Local Search operators
  - Integration of LS in Gassist
- Results
- Conclusions & further work

Introduction I

- For the last 10-15 years different families of approaches have been developed in the EC community for the principled and systematic improvement of how to explore the search space
  - Memetic Algorithms [Krasnogor & Smith, 05]
  - Estimation of Distribution Algorithms [Larrañaga & Lozano, 02]

Introduction II

- Beside some old precedents [Grefenstette, 91], this line of research is much more recent in the LCS field
- And mainly, EDA approaches have been employed
  - In Michigan LCS [Butz. 04]
  - In Pittsburgh LCS [Llorà et al. 05, Llorà et al. 06]

Introduction III

- Why the Memetic Approach is good for Pittsburgh LCS?
  - Pittsburgh LCS apply supervised learning approach
  - We know exactly which part of the solution (rule set) are performing well and which parts are performing bad
  - Also, the traditional Pittsburgh representation is semantically quite rich, providing a large volume of supervision information
  - All this performance information can be used to heuristically fine tune the solutions

Introduction IV

- In recent work [Bacardit & Krasnogor, 06] we integrated a local search operator into the crossover stage of the GAssist [Bacardit, 04] Pittsburgh LCS
- This operator takes the rules from multiple parents ($N \geq 2$) and heuristically selects a subset of them to generate a single offspring with maximum training accuracy
- That operator only recombined rules, it did not improve them

Introduction V

- In this work [accepted with revisions] our objective are
  - To develop some *rule-wise* local search mechanism to complement the *rule set-wise* previous operator
  - To study different policies to integrate these operators into GAssist
  - To exhaustively and rigorously evaluate all these combinations of operators and policies
- We name the new system that integrates all these LS mechanisms into GAssist “Memetic Pittsburgh Learning Classifier System (MPLCS)”

Framework: GAssist LCS

- GAssist is a descendant of GABIL [De Jong & Spears, 93]
- It implements a near-standard generational GA
- Each individual is a complete (and variable length) solution to the classification problem
- Fitness function is based on the MDL principle [Rissanen,78] to balance accuracy and complexity of each rule set

Framework: GAssist LCS

- Representation for nominal attributes (from GABIL)
  - Predicate (CNF) $\rightarrow$ Class
  - "If (attribute 1 takes values A or C) and (attribute 2 takes values A or B) and … and (attribute n takes value D) then predict class C"
  - The rules can be mapped into a binary string

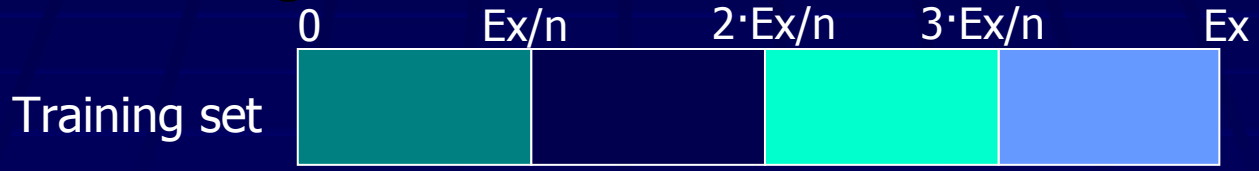
    1100|0010|1001|1
  - Size of rule: sum of the cardinalities of each attributre (binary) + 1 integer value for the class

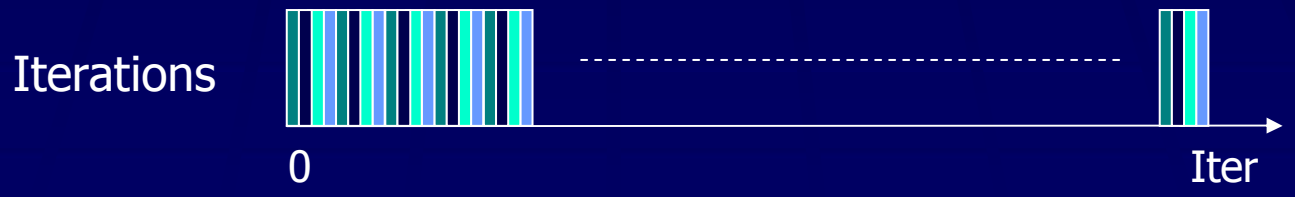
Framework: GAssist LCS

- Match process
  - Individuals are interpreted as a decision list [Rivest, 87]: an ordered rule set
  - Conflict resolution will choose always the first rule of the chromosome that matches the example
  - At the end of the rule set there is an static and explicit default rule
  - The class of the default rule will not be used by the other classes, reducing the search space

Framework: GAssist LCS

- Costly evaluation process if dataset is big
- Computational cost is alleviated by using a windowing mechanism called ILAS





- This mechanism also introduces some generalization pressure

Development of MPLCS: Local Search operators

- Rule set-wise (RSW) local search
- Rule-wise operators
  - Rule Cleaning (RC) local search
  - Rule Splitting (RS) local search
  - Rule Generalizing (RG) local search

Development of MPLCS: Local Search operators

- RSW
  - Motivation: try to make sure that we construct the most appropriate offspring from good candidate rules
  - High level procedure
    1. We choose N parents
    2. We evaluate all rules with all examples
    3. We select the best subset of rules (and the correct order) to generate a new offspring

Development of MPLCS: Local Search operators

- RSW: Rule subset selection (RSS) heuristic
  - We start with an empty rule set (predicting the default rule)
  - For each candidate rule
    - We find the top-most position inside the rule set where this rule helps the rule set maximizing accuracy
    - If rule contributes to improve rule set accuracy it is inserted
  - After testing all candidate rules, final rule set is pruned
    - Of rules that do not contribute anymore to accuracy
    - Too specific rules

Development of MPLCS: Local Search operators

- Rule-wise local search operators
  - The supervised learning process of a GAssist+GABIL representation gives us a huge amount of performance information
  - We can use this information to deterministically tune the rules we have

Development of MPLCS: Local Search operators

- Give the following rule:

Position	0	1	2	3	0	1	Class
Rule	1	1	1	0	1	1	1

- And the following examples:
- Rule missclassifies 3 & 6
- We can count, for each condition the number of wrong and right examples
- If condition 3 of att 1 is set to 0, rule will not misclassify

1. $0, 0||1$
2. $1, 0||1$
3. $2, 0||0$
4. $0, 1||1$
5. $1, 1||1$
6. $2, 1||0$

Development of MPLCS: Local Search operators

- Rule cleaning local search
  1. Find activated condition with most wrongly classified examples and no correct ones
  2. Set to 0 the corresponding bit

Development of MPLCS: Local Search operators

- Rule splitting
  - Motivation: sometimes it is very difficult to find a predicate that classifies always wrong
  - We tentatively try to split the rule by some of the attributes to determine if we can clean any of the two splitted rules
    - **Rule 1100|0010|1001|1 is splitted as:**
      - **1000|0010|1001|1**
      - **0100|0010|1001|1**

Development of MPLCS: Local Search operators

- Rule generalization
  - Prior operators only were dropping conditions for the false positive examples
  - We still need to cover the false negatives

1. Find examples not matched by prior rules
2. Find the currently disabled condition that
   1. Does not classify any negative examples
   2. Classifies more positive examples than the other candidates
3. If that condition is found, set corresponding bit to 1

Development of MPLCS: Integration of LS in GAssist

- Two policies of integration:
  - **Probabilistic (P)** policy
    - **Operators are applied to the whole population, given some probability**
    - **In RSW, this happens in the crossover stage, some probability choses between this operator and the classic one**
    - **Rule-wise operators are applied to the offspring population after mutation with an individual-wise probability**

Development of MPLCS: Integration of LS in GAssist

- Two policies of integration:
  - **Elitist (E)** policy
    - **Operators are applied only to the best(s) individual(s) of the population at the end of the GA cycle**
    - **RSW: Select the N best individuals of populations and generate a new offspring from them**
    - **Rule-wise operators: applied to the best individual of the population**

Development of MPLCS: Integration of LS in GAssist

- GAssist with rule-wise operators: MPLCS-R
- GAssist with RSW: MPLCS-RS
- GAssist with both kind of operators: MPLCS-RS+R
  - Rule-wise operators are applied inside RSW, just after RSS
- A suffix (P) or (E) specifies policy of application. Eg. MPLCS-RS+R(P)
- A suffix :RC+RS+RG specifies the combination of rule-wise operators. Eg. MPLCS-R(E):RC+RG

Experimental setup

- First stage of experiments
  - We have performed a large-scale set of experiments to determine the behaviour and performance of all these operators
  - Experiments used only the MX20 datasets: our aim was to evaluate the scalability of these operators, which is critical for the Proteins datasets
  - Tested in total 75 combination of operators/probabilities/policies of application
  - ILAS windowing is set up to very mild settings, to prevent interactions with the LS operators

Experimental setup

- Second stage
  - This time using aggressive settings of ILAS
  - Only tested the most promising MPLCS variants identified in the previous stage
  - Testing more datasets
    - kDNF dataset
    - MX37 and MX70
    - Noisy MX20. Class was randomly flipped with probability 5% - 25%

Experimental setup

- What is a good performance measure?
  - Iterations until 100% accuracy
  - Example evaluations until 100% accuracy
  - Run-time until 100% accuracy
- From the results of these experiments, and until we develop a better measure, run-time is the most insightful measure
- GAssist was run without fixed number of iterations: only until achieving 100% accuracy
- Test were run in Opteron processors @2.2Ghz

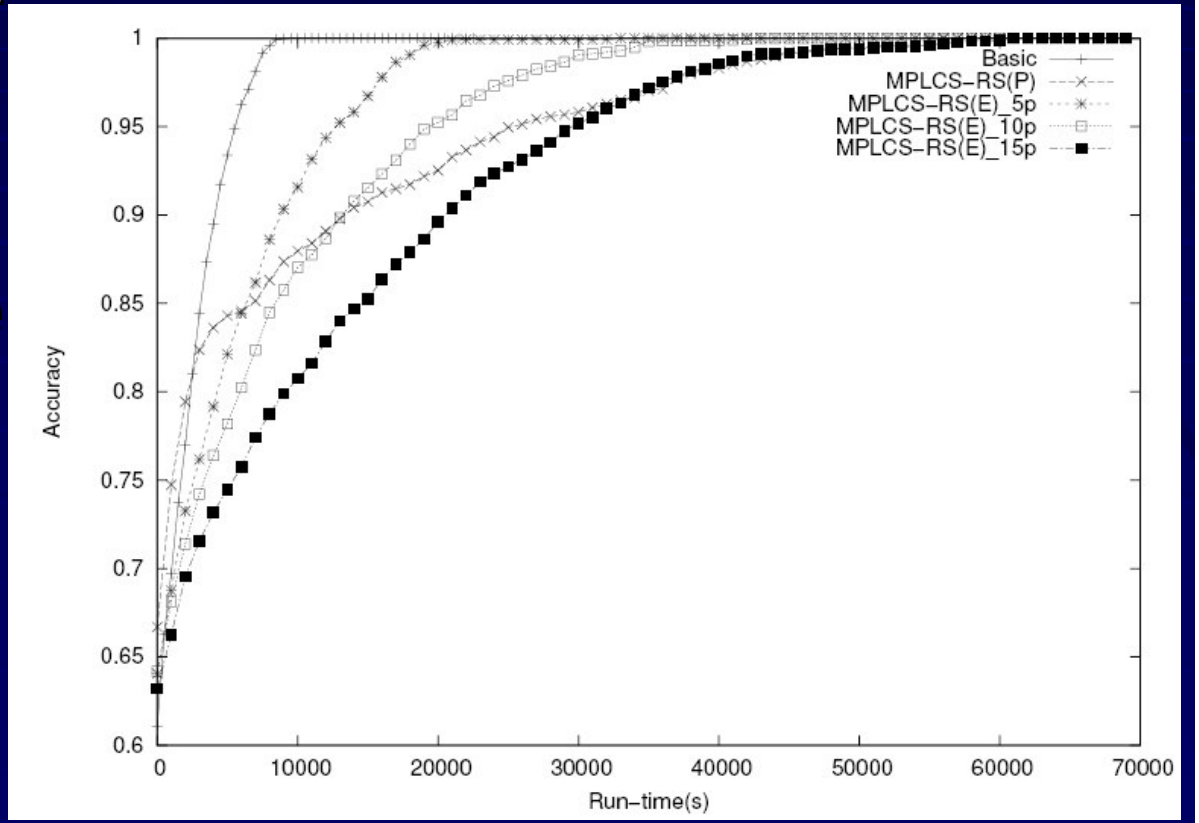
First stage of experiments

- Performance of MPLCS-RS
  - On this huge dataset this operator alone is actually worse than GAssist!!

Conf.	Iter.	#Rules	Run-time(s)
Basic	1073.70±99.11	23.1±2.9	**7950.3±792.1**
MPLCS-RS(P)	**695.80±92.17**	**17.0±0.0**	44758.7±6775.6
MPLCS-RS(E)_5p	1032.40±188.87	21.8±7.4	19127.4±4842.1
MPLCS-RS(E)_10p	1035.30±176.22	19.2±4.4	31459.5±6912.6
MPLCS-RS(E)_15p	1132.40±121.38	17.0±0.0	49356.1±7659.9

First stage of experiments

- Performance of MPLCS-RS
- MPLCS-RS(P) is only effective in early iterations
- Local search schedulers



First stage of experiments

- MPLCS-RS+R(P)
- Best results are obtained when combining all rule-wise LS operators

Conf.	Iter.	#Rules	Run-time(s)
Basic	1073.70±99.11	23.1±2.9	7950.3±792.1
MPLCS-RS+R(P):RC	323.30±81.83	**17.0±0.0**	20791.4±5080.3
MPLCS-RS+R(P):RS	648.40±413.90	23.3±12.4	65040.0±41849.0
MPLCS-RS+R(P):RG	385.70±70.00	**17.0±0.0**	32441.2±6324.4
MPLCS-RS+R(P):RC+RS	13.00±3.38	**17.0±0.0**	1861.2±295.2
MPLCS-RS+R(P):RC+RG	34.30±13.35	**17.0±0.0**	2873.1±1092.7
MPLCS-RS+R(P):RS+RG	6.70±1.10	**17.0±0.0**	1399.4±128.2
MPLCS-RS+R(P):RC+RS+RG	**4.40±0.49**	**17.0±0.0**	**1028.5±94.0**

First stage of experiments

- Global comparison

Category	Iter.	#Rules	Run-time(s)
Basic	1073.70±99.11	23.1±2.9	7950.3±792.1
MPLCS-R(P)	53.20±11.14	19.3±1.4	1243.0±204.5
MPLCS-R(E)	862.50±102.45	23.5±3.9	7264.5±909.8
MPLCS-RS	1032.40±188.87	21.8±7.4	19127.4±4842.1
MPLCS-RS+R(E)	93.70±34.35	**17.0±0.0**	5201.5±1900.1
MPLCS-RS+R(P)	**4.40±0.49**	**17.0±0.0**	**1028.5±94.0**

- (MPLCSRS+R(P), MPLCS-R(P)) → (Basic, MPLCS-RS+R(E), MPLCS-R(E)) → MPLCS-RS
- Best configuration is 7.73 faster than GAssist

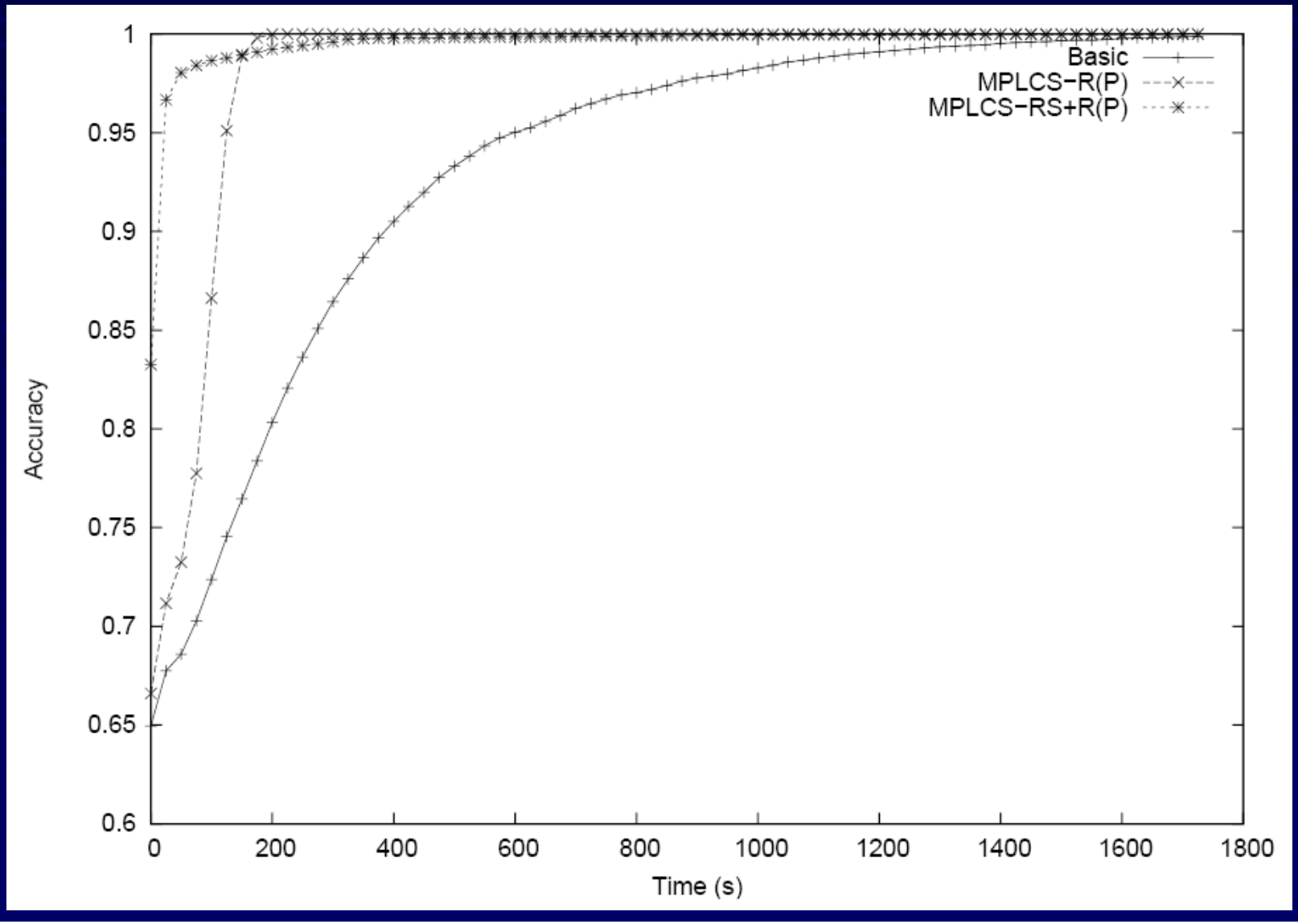
Second stage of experiments

- MX20 using 200 strata of ILAS

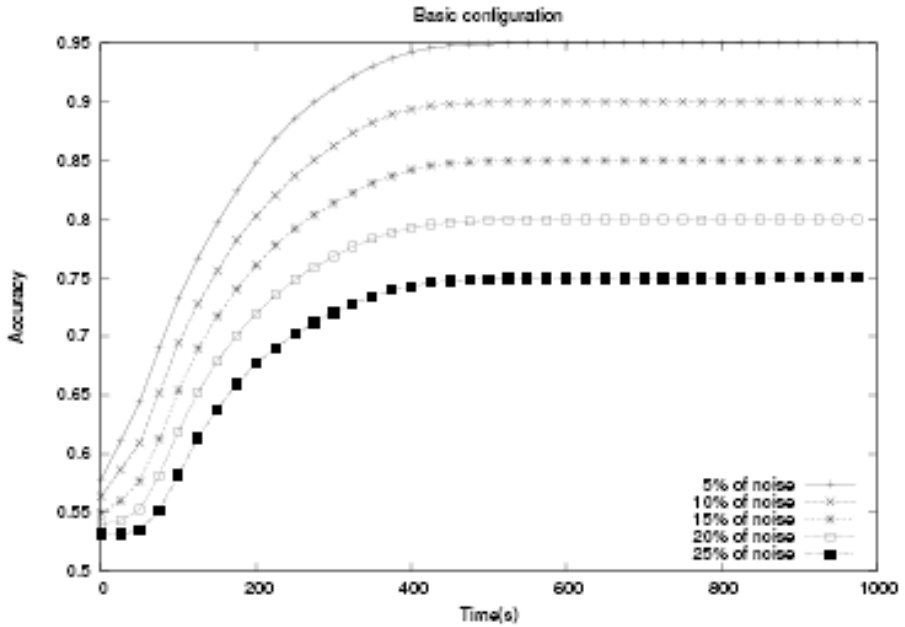
Category	Iter.	#Rules	Run-time(s)
Basic	1198.70±150.75	30.0±2.9	452.3±67.5
MPLCS-R(P)	23.70±4.65	24.8±5.2	30.4±4.4
MPLCS-RS+R(P)	**4.80±0.75**	**17.0±0.0**	**17.5±1.2**

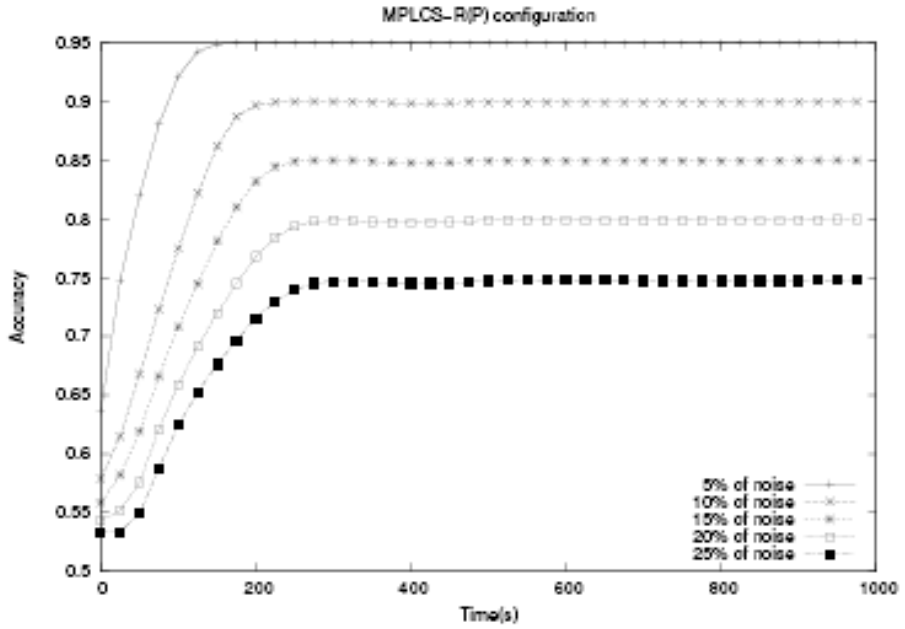
- MPLCS is 25.84 times faster than GAssist
- The LS operators interact well with ILAS
- So far so good….
  - However, the results on the new datasets will show the excessive exploitation power of MPLCS-RS+R(P)

Results on the kDNF dataset

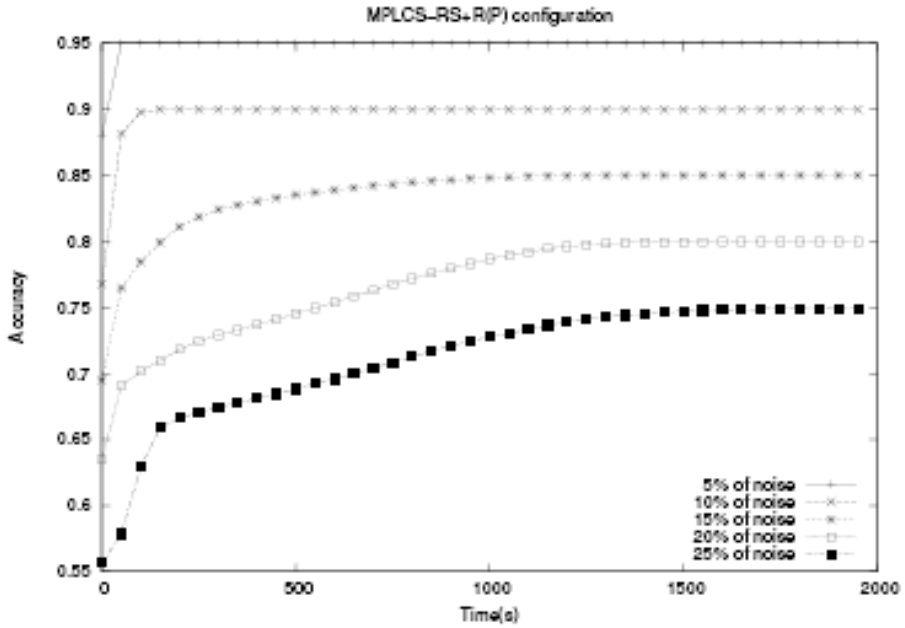


Results on the Noisy MX20





GAssist



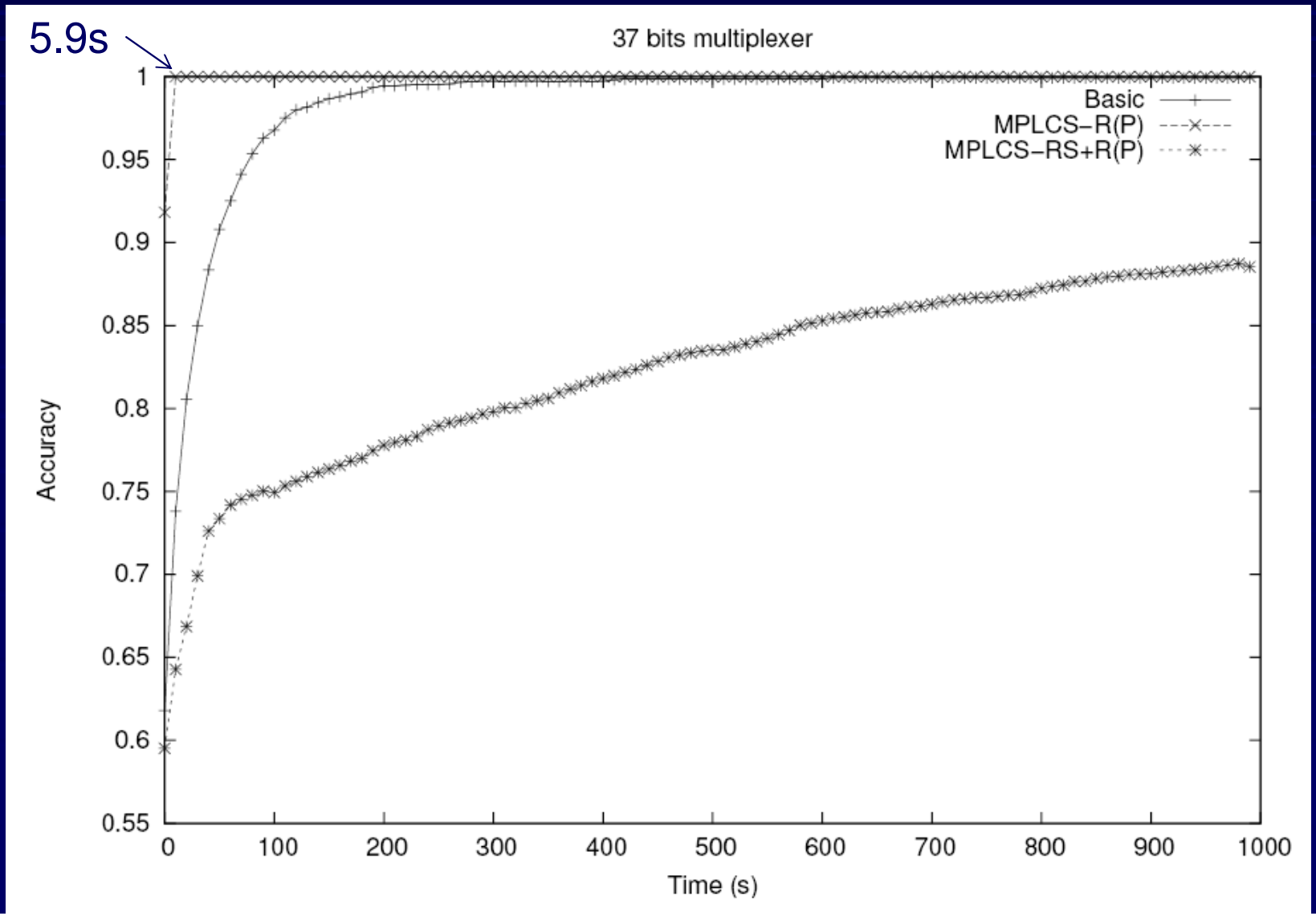
MPLCS-RS+R

MPLCS-R

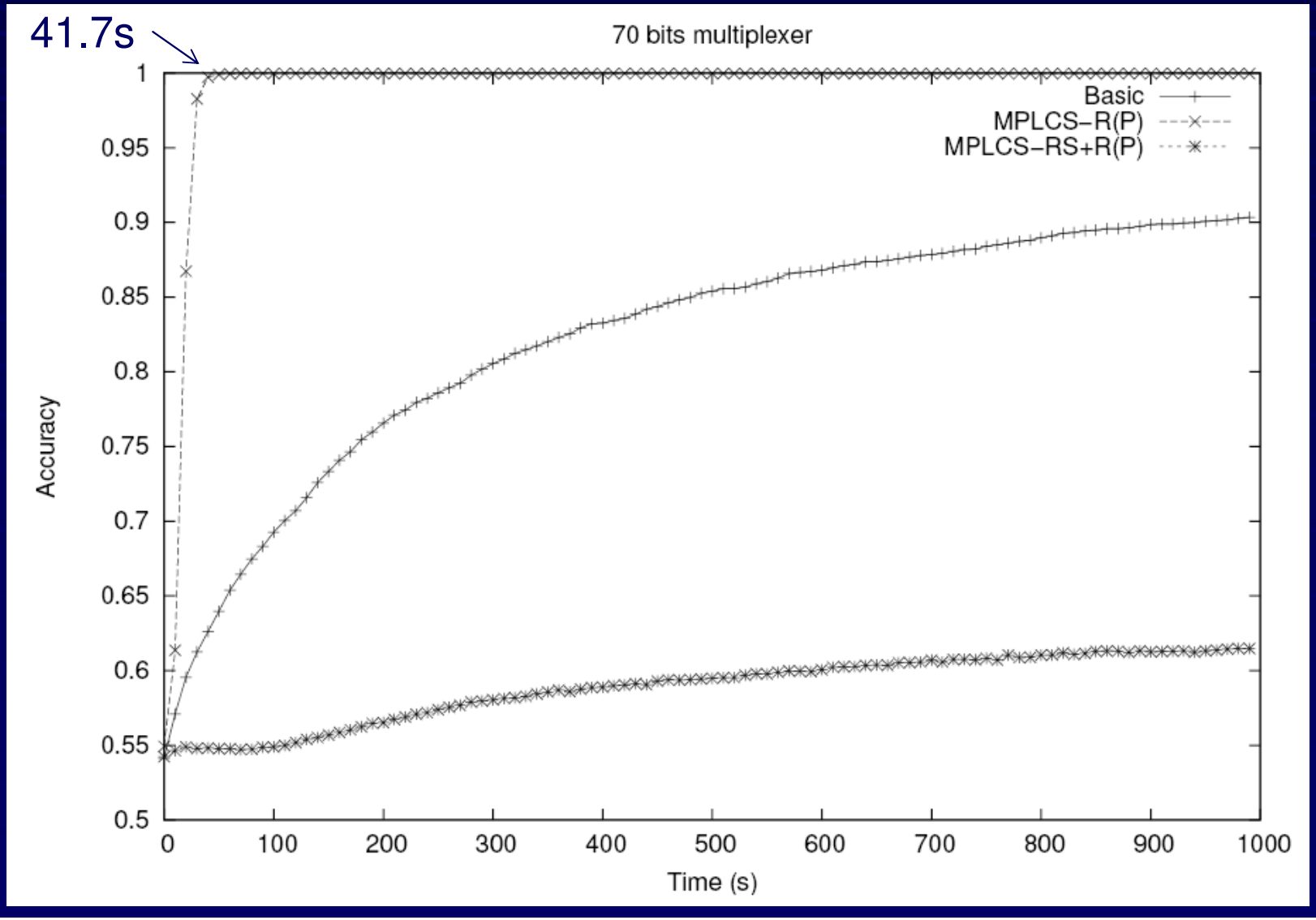
Preparation for MX37 and MX70

- These datasets have 2^{37} and 2^{70} instances, respectively
- Impossible to hold the whole dataset in memory
- Implemented them as a virtual dataset, with a generation function (as in XCS)
- ILAS in each generation uses a random sample (with replacement) of the dataset
- Sample was tuned to be as small as possible (while still letting the system learn) (<2000 inst)
- This tuning makes GAssist be quite "incremental"

Results on the MX37



Results on the MX70



Comparison with XCS(BOA)

- XCS and XCS(BOA) have reported results for all the tested datasets
- But before comparing, what metric do we use?
- Each GA iteration of GAssist (with ILAS) uses n=|T|/#strata examples.
- Learning steps ≈ n · #iterations until convergence

Comparison with XCS(BOA)

Dataset	#instances per iteration	#learning steps
20 bits multiplexer	5243	124259
37 bits multiplexer	1373	254190
70 bits multiplexer	1574	1167751
noisy 20 bits multiplexer - 5% noise	5243	1830856
noisy 20 bits multiplexer - 10% noise	5243	3307809
noisy 20 bits multiplexer - 15% noise	5243	4007225
noisy 20 bits multiplexer - 20% noise	5243	4489057
noisy 20 bits multiplexer - 25% noise	5243	4834046
kDNF	5243	2584275

- There are almost no actual figures of XCS performance, only plots, so comparison is only approximate
- Usually XCS performed better. Only in MX37 and MX70 (with extreme tuning of ILAS) performance of MPLCS is quite comparable
- 200000 vs. 254191 and 900000 vs. 1167751

Conclusions and further work

- Development of MPLCS
  - Following the design principles of Competent MA
  - Studying various kinds of operators
  - Studying various policies of application
- Large scale experiments
  - Best results when proper equilibrium between the various rule-wise operators
  - RSW, while performing well on “small” datasets, shows excessive exploitation power in the cases when the good rules have not been discovered yet
  - Operators interact well with ILAS

Conclusions & further work

- Further work
  - Test these operators on other datasets (hierarchical decomposable datasets)
  - Overcome the limitations of the RSW operator
  - Integrate these operators in other flavours of LCS/GBML
  - Adapt operators to real datasets and continuous attributes
  - Study other design issues of MA