Closed Loop Machine Learning: Complexity of ASE-Progol

Alireza Tamaddoni-Nezhad
Stephen Muggleton

Department of Computing, Imperial College
180 Queen's Gate, London SW7 2BZ, UK

email: {atn,shm}@doc.ic.ac.uk

January 2002

Abstract

In this report we study the complexity of each implemented component of the Closed Loop Machine Learning in ASE-Progol. In the first part of the report, we review each component of ASE-Progol and discuss the complexity of each component. In the second part, we perform an experimentation to compare the average run time of each component of ASE-Progol in each iteration of the closed loop machine learning. This experimentation is repeated to measure the average run time for both phase A and phase B data.

1 Introduction

The purpose of the Closed Loop Machine Learning project has been to develop a framework for "Automatic Experimentation" which involves Machine Learning for generating hypotheses and Robotics for devising trials to discriminate between hypotheses. In this framework there is a closed loop between the process of forming hypotheses and the collection of data. The long term goal is to use this framework in Functional Genomics to discover the function of genes.

According to the project proposal [1], the main objectives of the project are: "to test whether closed loop Machine Learning Systems can (i) efficiently converge to accurate hypotheses and (ii) be physically realised using robotics and successfully applied to a discovery task in functional genomics".

For this purpose a system called ASE-Progol (Active Selection of Experiments with Progol) has been developed. ASE-Progol is an Active Learning system which uses the Inductive Logic Programming (ILP) system Progol5.0 [7] for generating hypotheses together with a CART-like algorithm [2] to select trials which minimise the expected cost of experimentation. More details about the design and implementation of ASE-Progol can be found in [3, 4].

To date, ASE-Progol has been tested on: a) a small and simplified model of functional genomics and b) a metabolic pathway from the aromatic amino acid pathway of yeast. The results of these studies which correspond to phase A and phase B of the project are reported in [3] and [4] respectively.

In this report we study the complexity of each implemented component of ASE-Progol. In section 2 we review each component of ASE-Progol and discuss the complexity of each component. In the second part of the report, we perform an experimentation to compare the average run time of each component of ASE-Progol in each iteration of the closed loop machine learning. This experimentation is repeated for both phase A and phase B data. The experimental settings and the results of the experimentation are discussed in section 3.

2 Components of ASE-Progol

The Closed Loop Machine Learning (CLML) in ASE-Progol is shown in Figure 1. This figure shows the components of ASE-Progol and the relationship between these components. This closed loop machine learning is implemented as a Unix C Shell script and each component of ASE-Progol is called within this loop. In this figure two component are not implemented yet. These components are: a) Robot Instructor which instructs the robot to perform the next trial and b) Translator Analyser which converts the result of a trial into a form which can be used by ASE-Progol. The result of this component is added to the example set which is used by Hypotheses Generator. In the existing implementation, the result of a trial is determined by the oracle – a file which contains the results of all possible trials – rather than by the laboratory. In the following we review each implemented component of ASE-Progol.

2.1 Hypotheses Generator

As mentioned earlier, ASE-Progol uses Progol5.0 for generating hypotheses from the examples of an observable predicate and background knowledge.

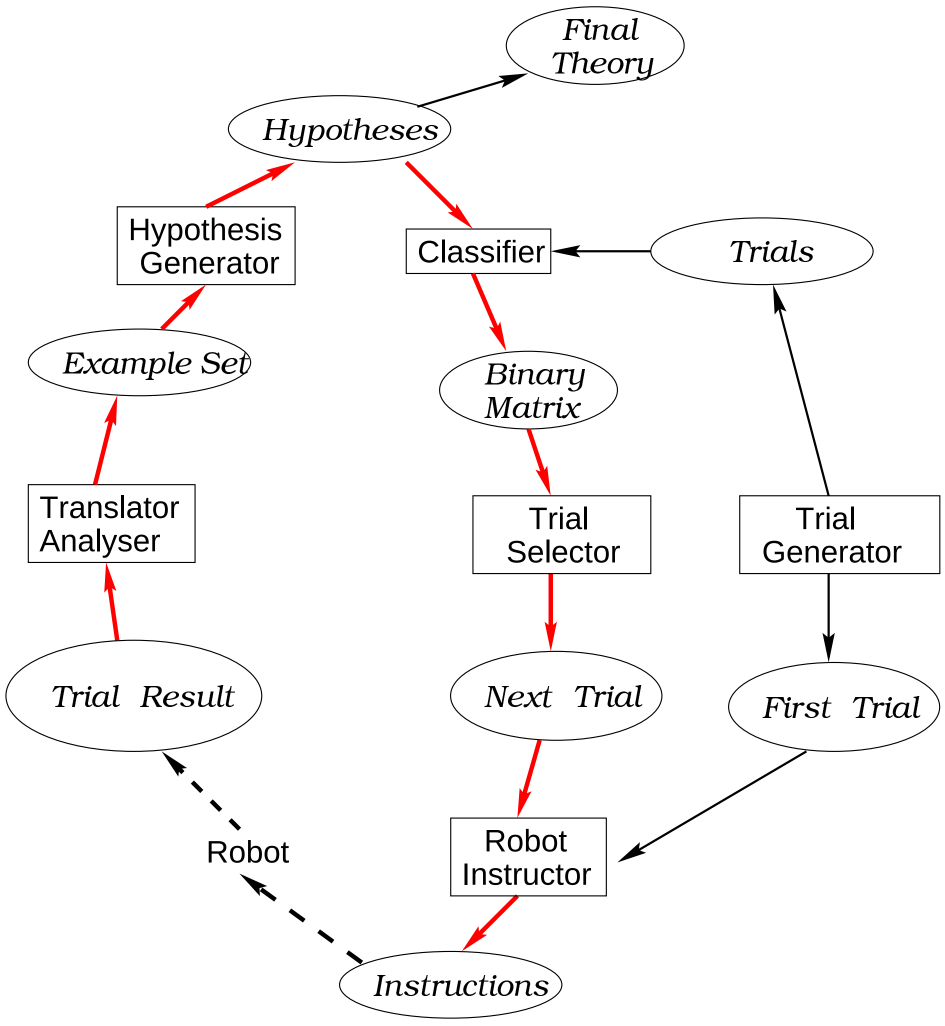


Figure 1: The Closed Loop Machine Learning (CLML) in ASE-Progol (as in [3]). The boxes represent components of ASE-Progol and the ellipses represent information which is passed between these components.

In each iteration of the closed loop learning Progol5.0 is given a training example-set which is added one example in each iteration of the closed loop (i.e. the result of a trial). Progol5.0 is also provided a background knowledge which contains the model to be completed. Progol5.0 uses 'Theory Completion using Inverse Entailment' for induction/abduction of hypotheses. More details about Theory Completion using Inverse Entailment and Progol5.0 can be found in [7].

Note that unlike experiments in [7], in ASE-Progol it is assumed that the system is given only one example in each iteration of the closed loop (i.e. the result of a trial). The result of the hypotheses generator (Progol5.0) is passed to ASE-Progol by extracting from Progol's log file, hypotheses which have a positive compression. ASE-Progol then selects clauses which have the maximum compression for each predicate and passes them to the Classifier. This means that in each iteration, ASE-Progol selects each trial to discriminate the space of 'compressive' clauses.

Complexity of the Hypotheses Generator Progol constructs a most specific clause for each example and then performs a A^*-like search in a subsumption lattice which is bounded below by this bottom clause. This search is repeated for each example and each clause in the final theory. For this search to be polynomially tractable, a bound is required on the length of any hypothesised clause (c). The Prolog interpreter inside Progol (which is used to compute the coverage of each hypothesised clause) is also bounded in its number of resolution steps (r) and its Proof depth (h). Parameters c, r and h are controllable by the user. Given these bounds, the complexity of Hypotheses Generator increases linearly in both:

1. the number of training examples in each iteration of the closed loop and
2. the number of clauses in the final theory

A more detailed analysis of Progol can be found in [5]

2.2 Trial Generator

The closed loop learning normally begins with this component which generates a random sample of maximum 20 trials from the space of possible trials. It has been suggested that using the upper bound of 20 on the number of candidate trials ensures that there is a low probability that the sample will not contain a new trial [4].

In the first iteration of the closed loop, first trial is determined by selecting the cheapest trial in the sample generated by the trial generator. In the subsequent iterations of the loop, the trial generator removes from the sample any duplicate trials or trials which have already been performed by the robot. The result which is a set of maximum 20 trials is then passed to the Classifier which constructs a binary matrix. This binary matrix is used by Trial Selector which selects the next trial to be performed by the robot in the subsequent iterations of the closed loop.

Complexity of the Trial Generator The Trial Generator generates a random sample of the space of possible trials. In ASE-Progol version 0.0, an Stochastic Logic Program (SLP) [6] has been used to sample from the trials which are represented as Prolog ground facts. Whilst, in ASE-Progol version 1.0, a Prolog implementation of Knuth's random subset algorithm has been used for this purpose. In both sampling methods, the complexity increases linearly in the cardinality of the random sample. As mentioned before, in ASE-Progol the cardinality of this sample is bounded to 20.

2.3 Classifier

The result of the Hypotheses Generator (i.e. a set of hypotheses) and the result of Trial Generator (i.e. a set of trials) are given as input to the Classifier. The purpose of the Classifier is to determine whether the outcome of each trial is consistent with each hypothesis. This is a theorem proving problem and for this purpose ASE-Progol uses the Progol interpreter inside Progol5.0.

The result of the Classifier is a binary matrix in which entry ij is 1 if the outcome of the trial i is logically consistent with the hypothesis j and 0 otherwise. This binary matrix is then used to determine how the outcome of a trial partitions the set of hypotheses into 'consistent hypotheses' and 'inconsistent hypotheses' with respect to that trial.

Complexity of the Classifier In each iteration of the closed loop, the Classifier constructs the binary matrix which for each candidate trial determines whether the outcome of this trial is consistent with each candidate hypothesis or not. For this purpose, the Classifier repeatedly calls the theorem prover (i.e. the Progol interpreter) for each candidate hypothesis (i.e. the output of the Hypotheses Generator) and each candidate trial (i.e. the output of the Trial Generator). It is assumed that the Prolog interpreter is bounded in the number of resolution steps (r) and the proof depth (h). Given these bounds, the complexity of the Classifier increases linearly in both:

1. the number of candidate hypotheses (i.e. the output of the Hypotheses Generator) and
2. the number of candidate trials (i.e. the output of the Trial Generator)

2.4 Trial Selector

Given the cost of each trial and the binary matrix, the Trial Selector computes the expected cost of experimentation for each candidate trial. The next trial to be performed by the robot is then selected as the trial which minimises the expected cost of the experimentation.

Given the set of candidate hypotheses H and the set of candidate trials T, the expected cost of experimentation for each trial is estimated by the following heuristic function:

$$\begin{aligned} EC(H,T) \approx min_{t \in T}[C_t &+ p(t)(mean_{t' \in (T-t)} C_{t'}) J_{H_{[t]}} \\ &+ (1-p(t))(mean_{t' \in (T-t)} C_{t'}) J_{H_{[\bar{t}]}}] \end{aligned}$$

where

$$J_H = -\sum_{h \in H} p(h) \lfloor \log_2(p(h)) \rfloor$$

In this formula C_t is the cost of the trial t, $p(t)$ is the probability that the outcome of the trial t is positive, $p(h)$ is the prior probability that the hypothesis h is correct, $mean_{t' \in (T-t)} C_{t'}$ is the mean value of all trial costs excluding the trial t, $H_{[t]}$ is the set of all hypotheses which are consistent with the trial t and $H_{[\bar{t}]}$ is the set of hypotheses which are not consistent with the trial t.

It has been suggested that this approximation can be used to minimise the expected cost of eliminating all but one of hypotheses [3].

Complexity of the Trial Selector The trial selector computes the expected cost of experimentation for each trial and selects a trial with a minimum expected cost. In computing the expected costs, it is required to compute the prior probabilities for consistent and inconsistent hypotheses. The complexity of this computation increases linearly in the number of hypotheses. Hence, as in the Classifier, the complexity of the Trial Generator increases linearly in the number of candidate trials and the number of hypotheses. However, the Trial selector only involves simple numerical commutation and a lower complexity is expected in comparison to the Classifier which involves a Prolog interpreter for theorem proving.

2.5 Termination Condition

In the existing implementation of ASE-Progol, the closed loop learning terminates if one of the following conditions is met:

1. all possible trials have been tried or
2. the experimental resources have been exhausted or
3. the number of trials exceeds a limit specified by the user

The number of possible trials for each gene is equal to the number of possible growth media for that gene. Thus the number of iterations of the closed loop learning is equal to the number of possible growth media unless either the experimental resources have been exhausted or the number of trials exceeds a limit specified by the user.

3 Experiment: Time Complexity of ASE-Progol

The experimentation in this section aims to compare the average run time of each component of ASE-Progol in each iteration of the closed loop machine learning. This experimentation is repeated for phase A and phase B data.

In the following sections, we first explain the experimental materials and method and then represent and discuss the results.

3.1 Materials

In the experiments of phase A and phase B, one approach to functional genomics, namely the effect of single-gene-deletion growth trials, has been modeled by logic programs. These logic programs for phase A and phase B are shown in Table 1 and Table 3 and the relevant metabolic pathways are shown in Figure 2 and Figure 3 respectively. During these experimentations some of `code/2` facts (which correspond to the function of unknown genes) are removed from the model and the ability of ASE-Progol to 'cost-efficiently' recover the performance of the model (or learn the function of the

Table 1: A logic program which represents the functional genomics model which has been used in phase A experiments.

```
phenotypic_effect(Gene, Growth_medium):-
     nutrient_in(Nutrient, Growth_medium),
     metabolic_path(Nutrient, Mi),
     enzyme(E, Mi, Mj),
     codes(Gene, E),
     metabolic_path(Mj, Mn),
     essential_molecule(Mn),
     not(path_without_E(Growth_medium, Mn, E)).

nutrient_in(Nutrient, Growth_medium):- element(Nutrient, Growth_medium).

metabolic_path(A, A).
metabolic_path(A, B):- enzyme(_, A, B).
metabolic_path(A, B):- enzyme(_, A, X), metabolic_path(X, B).

path_without_E(Growth_medium, Mn, E):-
     nutrient_in(Nutrient, Growth_medium),
     path_without_E(Nutrient, Mn, E).
path_without_E(A,A,_).
path_without_E(A,B,E):- enzyme(E2,A,B),not(E=E2).
path_without_E(A,B,E):- enzyme(E2,A,X),not(E=E2),path_without_E(X,B,E).

essential_molecule(ess_mol_1).      essential_molecule(ess_mol_2).
essential_molecule(ess_mol_3).      essential_molecule(ess_mol_4).

enzyme(enzyme_a, nut_1, metabolite_1).
enzyme(enzyme_b, nut_2, metabolite_2).
enzyme(enzyme_c, nut_3, metabolite_3).
enzyme(enzyme_d, metabolite_1, metabolite_4).
enzyme(enzyme_e, metabolite_1, metabolite_5).
enzyme(enzyme_f, metabolite_1, metabolite_6).
enzyme(enzyme_g, metabolite_2, metabolite_6).
enzyme(enzyme_l, metabolite_6, metabolite_7).
enzyme(enzyme_h, metabolite_2, ess_mol_4).
enzyme(enzyme_i, metabolite_3, ess_mol_4).
enzyme(enzyme_j, metabolite_4, ess_mol_1).
enzyme(enzyme_k, metabolite_5, ess_mol_2).
enzyme(enzyme_m, metabolite_7, ess_mol_3).

codes(gene_a, enzyme_a).   codes(gene_b, enzyme_b).   codes(gene_c, enzyme_c).
codes(gene_d, enzyme_d).   codes(gene_e, enzyme_e).   codes(gene_f, enzyme_f).
codes(gene_g, enzyme_g).   codes(gene_h, enzyme_h).   codes(gene_i, enzyme_i).
codes(gene_j, enzyme_j).   codes(gene_k, enzyme_k).   codes(gene_l, enzyme_l).
codes(gene_m, enzyme_m).
```

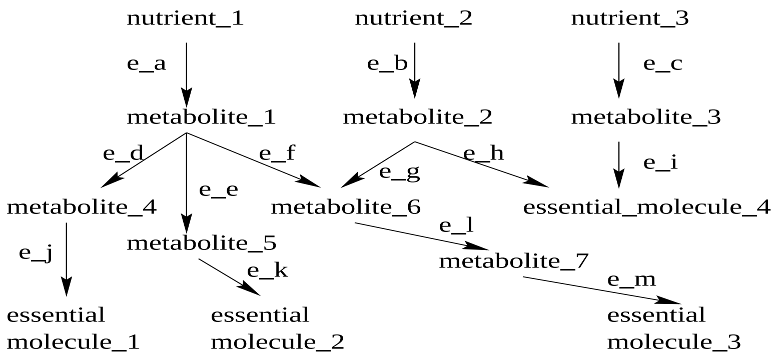


Figure 2: A graph which represents the metabolic pathway of the genomics model in phase A (as in [3]).

Table 2: Cost A and Cost B of Growth Media which has been used in the experiments in phase A (as in [3]).

Growth medium	Cost of Trial	
	Costs A	Costs B
nutrient_1	10	10
nutrient_2	20	100
nutrient_3	30	1000
nutrient_1, nutrient_2	30	110
nutrient_1, nutrient_3	40	1010
nutrient_2, nutrient_3	50	1100
nutrient_1, nutrient_2, nutrient_3	60	1110
Sum of costs	240	4440

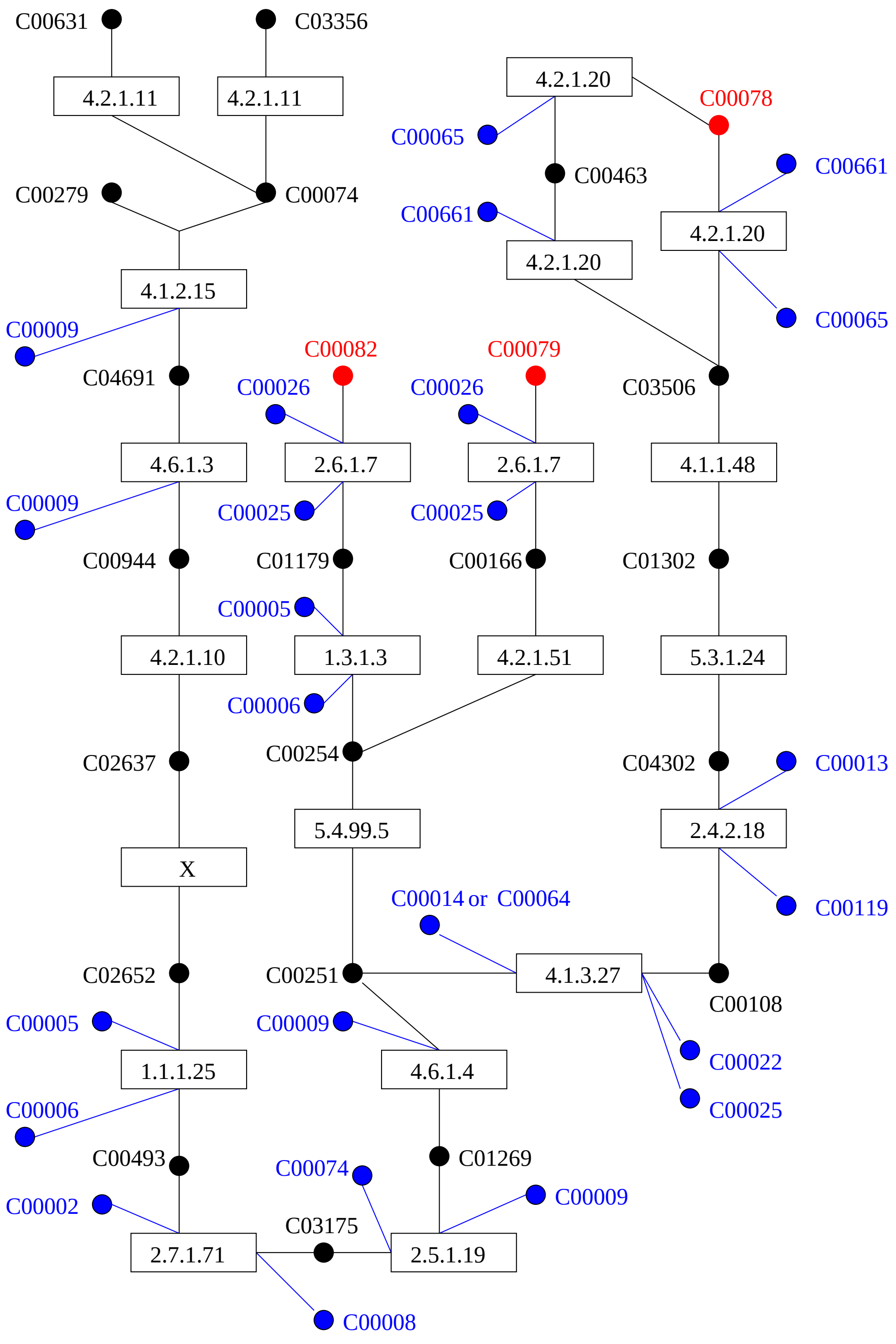


Figure 3: Part of the aromatic amino acid pathway of yeast which has been used in the experiments of phase B (as in [4]).

Table 3: Part of the logic program which represents the functional genomics model which has been used in the experiments of phase B (as in [4]).

```
phenotypic_effect(ORF, Growth_medium):-
    generated_by_other_pathways(Ubiquitous_metabolites),
    union(Ubiquitous_metabolites, Growth_medium, Starts),
    connected(Starts, Wild_products),
    ends(Ends),
    subset(Wild_products, Ends),
    enz(Enzyme, Reactants, Products),
    encodes(ORF, Enzyme, Reactants, Products),
    connected_without_this_step(Starts, Mutant_products,
                                Enzyme, Reactants, Products),
    not(subset(Mutant_products, Ends)).

enz(Ec, R2, R1) :- enzyme(Ec, R1, R2).
enz(Ec, R1, R2) :- enzyme(Ec, R1, R2).

encodes(ORF, Enzyme, Reactants, Products):-
    codes(ORF, Enzyme, Reactants, Products).

encodes(ORF, Enzyme, Reactants, Products):-
    codes(ORF, Enzyme, Products, Reactants).

generated_by_other_pathways(['C00002', 'C00005', 'C00006',
'C00014', 'C00025', 'C00064', 'C00065', 'C00074', 'C00119',
'C00279', 'C00631', 'C03356']).

ends(['C00078', 'C00079', 'C00082']).
```

genes) is measured. The cost of the experimentation is assumed to be equal to the total cost of the growth media (nutrients) which have been used in the growth trials during the experimentation. For example, in the experimentation in phase A it is assumed that the costs of the growth media are as shown in Table 2. It is also assumed that the result of each trial can be represented by positive and negative instances of the predicate `phenotypic_effect/2`. For example `phenotypic_effect(gene_a, [nutrient_1, nutrient_2])` represents a growth trial in which a mutant strain of an organism is created by removing `gene_a` and the growth media contains `nutrient_1` and `nutrient_2`. The result of a trial can be positive (growth) or negative (no growth). For example, $\neg$ `phenotypic_effect(gene_a, [nutrient_2])` shows that no growth is observed for `gene_a` and `nutrient_2`.

In the experimentation in phase A, there are only 3 nutrient in the model, so there are 7 possible growth media for each of the 13 gene in the model. Hence, we have 91 examples for the predicate `phenotypic_effect/2`. According to the metabolic pathway in Figure 2, 45 of theses examples are positive and the other 46 are negative examples. In the experimentation in phase B, there are 13 optional nutrients which can be added to a basal medium. Thus there are $2^{13} = 8192$ of these optional nutrients, however, to make the experimentation tractable, only 3 of the possible 13 nutrients were used in the experimentation of phase B. Hence, the number of possible media is $\sum_{i=0}^{3} \binom{13}{i} = 378$. In other words, in the experimentation in phase B there are 378 examples for each ORF in the model. The class of each example (positive or negative) was deduced from the complete model. Table 4 shows the distribution of positives and negatives for each ORF. There are no positive examples of the `phenotypic_effect/2` predicate for the ORFs `YDR254W, YER090W, YGL026C, YHR174W,` and `YMR323W`. Hence phenotypic effects can be observed for just 12 of the 17 ORFs in the model. Thus when conducting rediscovery experiments in phase B, only these 12 ORFs are used.

More details about the experimentations of phase A and phase B can be found in [3] and [4].

3.2 Method

As mentioned earlier, the purpose of this experiment is to study the complexity of ASE-Progol when recovering the incomplete genomics models of phase A and phase B. ASE-Progol uses 'Theory Completion' which is implemented in Progol5.0. Theory Completion, Progol5.0 and the results of testing Progol5.0 on the functional genomics model are presented in [7]. Un-

Table 4: Distributions of positive and negative examples in the experiments of phase B (as in [4]).

ORF	Number of examples	
	Positive	Negative
YBR166C	232	146
YBR249C	70	308
YDR007W	232	146
YDR035W	70	308
YDR127W	104	274
YDR254W	0	378
YDR354W	232	146
YER090W	0	378
YGL026C	0	378
YGL148W	104	274
YGL202W	366	12
YHR137W	366	12
YHR174W	0	378
YKL211C	232	146
YMR323W	0	378
YNL316C	232	146
YPR060C	151	22

like the experiments with Progol5.0, in the experiments with ASE-Progol it is assumed that ASE-Progol is given one example in each iteration (the result of a growth trial). In the existing implementation, the result of a trial is determined by the oracle – a file which contains the results of all possible trials – rather than by the laboratory.

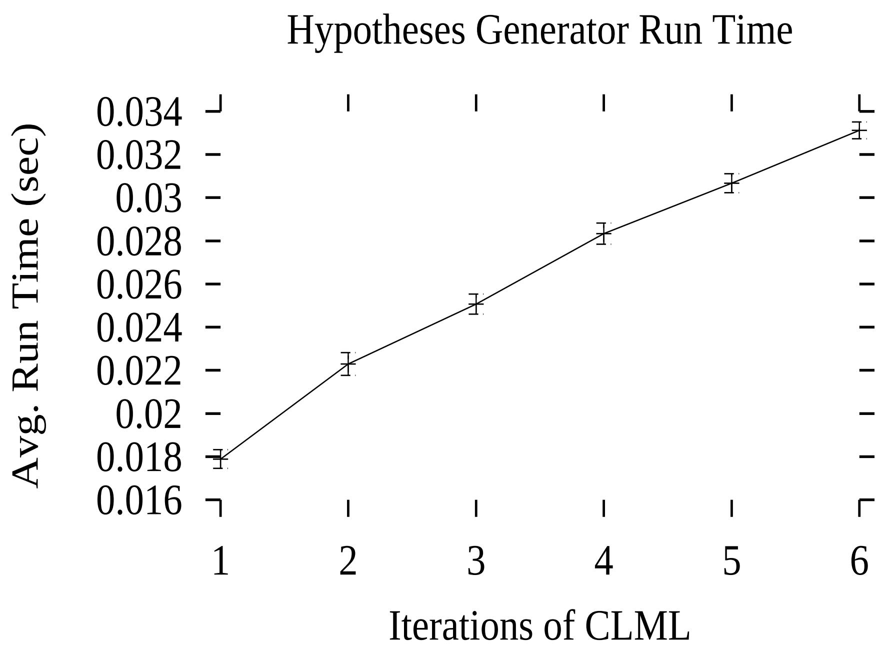
During the experimentations, the genomics models shown in Table 1 and Table 3 are made incomplete by randomly removing a number of `code/2` facts. In the phase A, this number varies between 5, 9 and 13 in different experiments. In the phase B, all of `code/2` facts are removed from the model. In the present experimentation, ASE-Progol is executed on the incomplete model such that trials are selected which minimise the expected cost of experimentation. The run time is then plotted for each component of ASE-Progol in each iteration of the closed loop. In addition, the average number of candidate hypotheses and the average number of the candidate trials, in each iteration of the closed loop, are also plotted.

4 Results and discussion

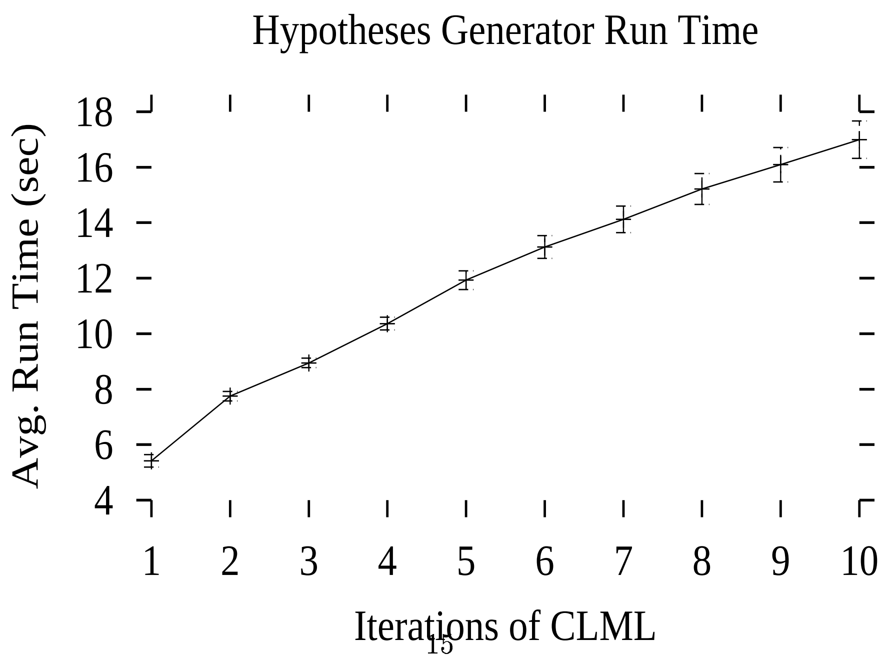
The average run time of the Hypotheses Generator in the experimentations of phase A and phase B are shown in Figure 4. This figure suggests that in both phase A and phase B, the run time of the Hypotheses Generator has a roughly linear increase in the iterations of the closed loop learning. This is because the cardinality of the training examples is incremented in each iteration and as mentioned in section 2.1, the complexity of the Hypotheses Generator increases linearly in the number of training examples. The difference between the average run time in phase A and phase B shows that the search which is performed by Progol in each iteration of the closed loop (see section 2.1) in phase B is computationally more expensive than the search which is performed in phase A.

The average run time of the Trial Generator in the experimentations of phase A and phase B are shown in Figure 6. This figure shows that in both phase A and phase B the average run times for the Trial Generator are relatively low and independent of the iterations of the closed loop. As mentioned in section 2.2, the complexity of the Trial Generator is linear in the cardinality of the random sample which has an upper bound of 7 and 20 in the experimentation of phase A and phase B respectively.

The average run times of the Classifier and the Trial Selector are shown in Figures 8 and 9 respectively. These figures suggest that the average run time for the Classifier and the Trial Selector follow a similar pattern though

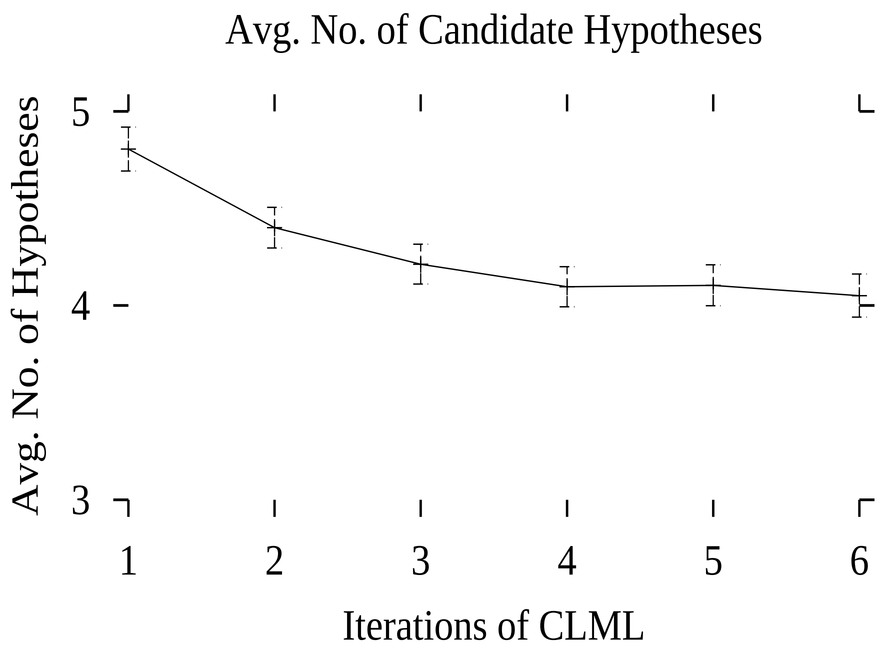


(a)

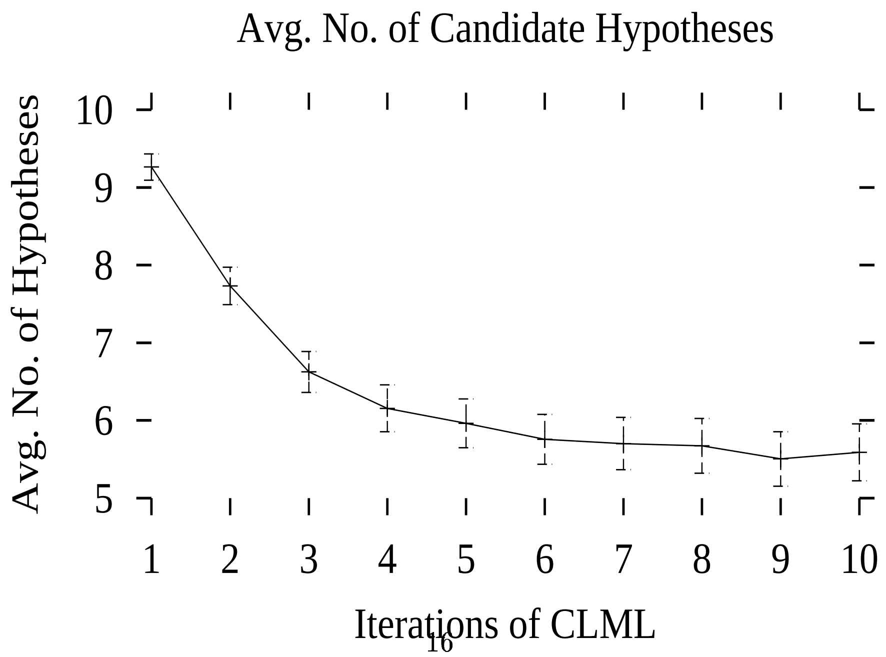


(b)

Figure 4: The Hypotheses Generator run time on: a) Phase A data b) Phase B data.

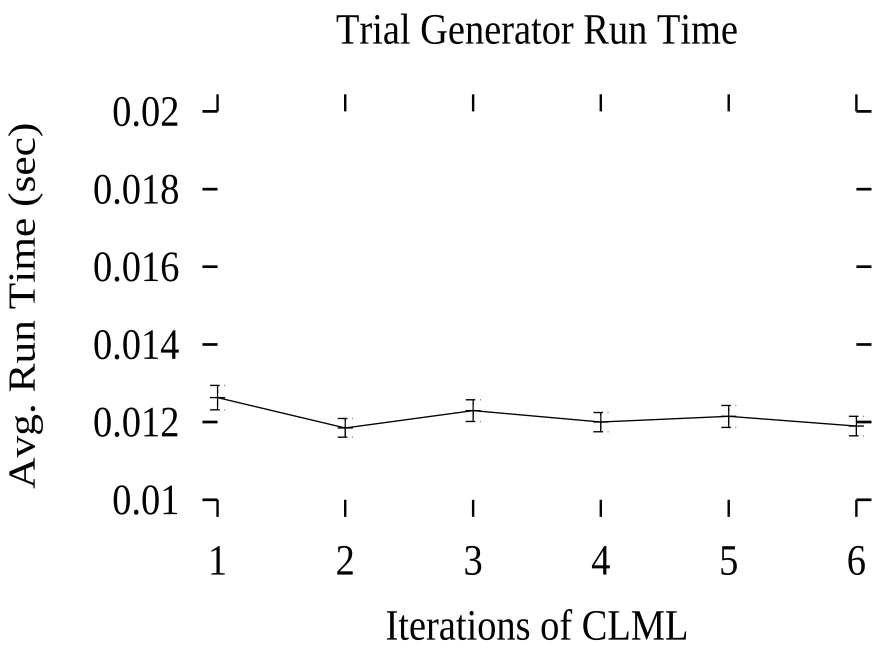


(a)

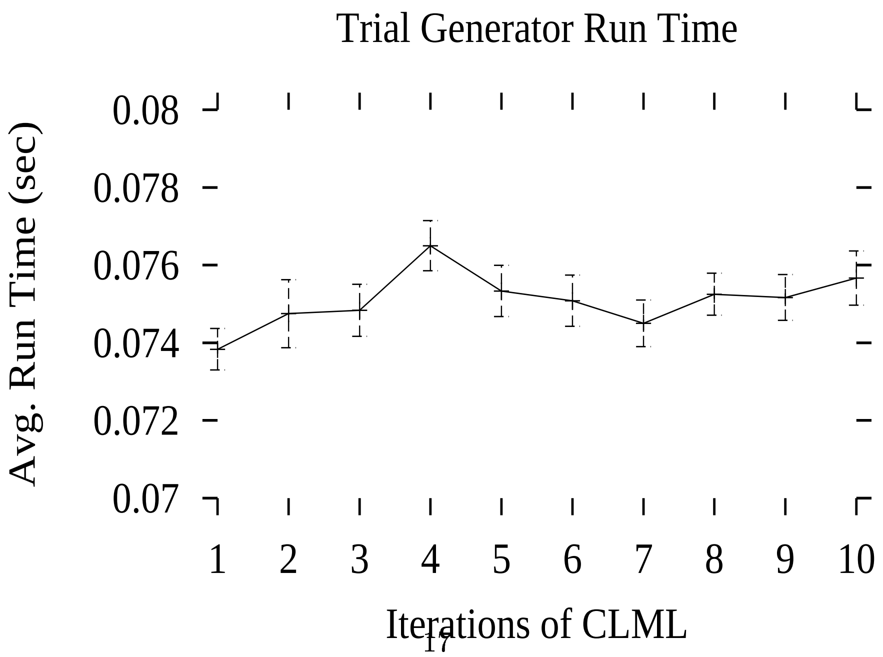


(b)

Figure 5: The average number of the generated hypotheses in each iteration of ASE-Progol on: a) Phase A data b) Phase B data.

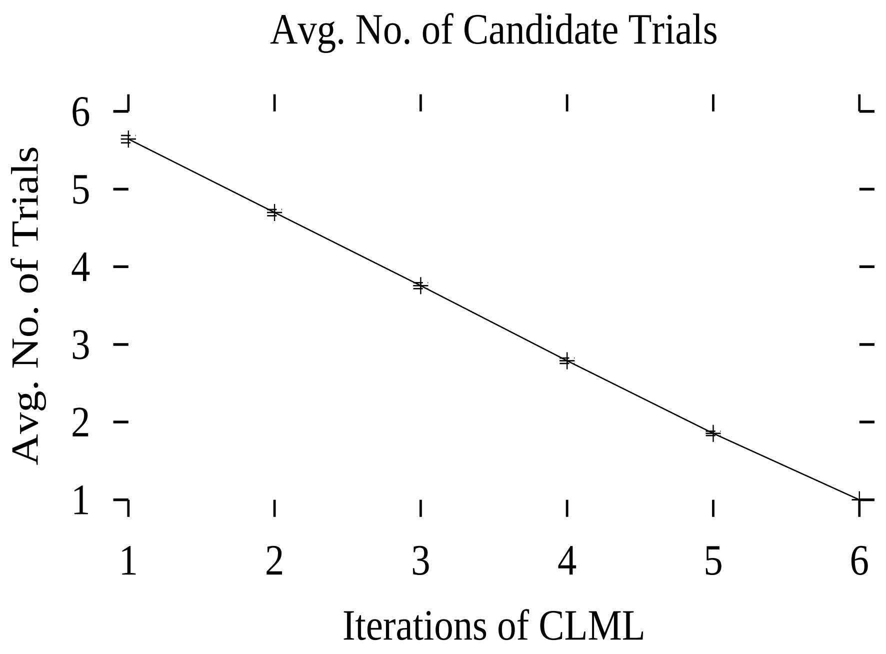


(a)

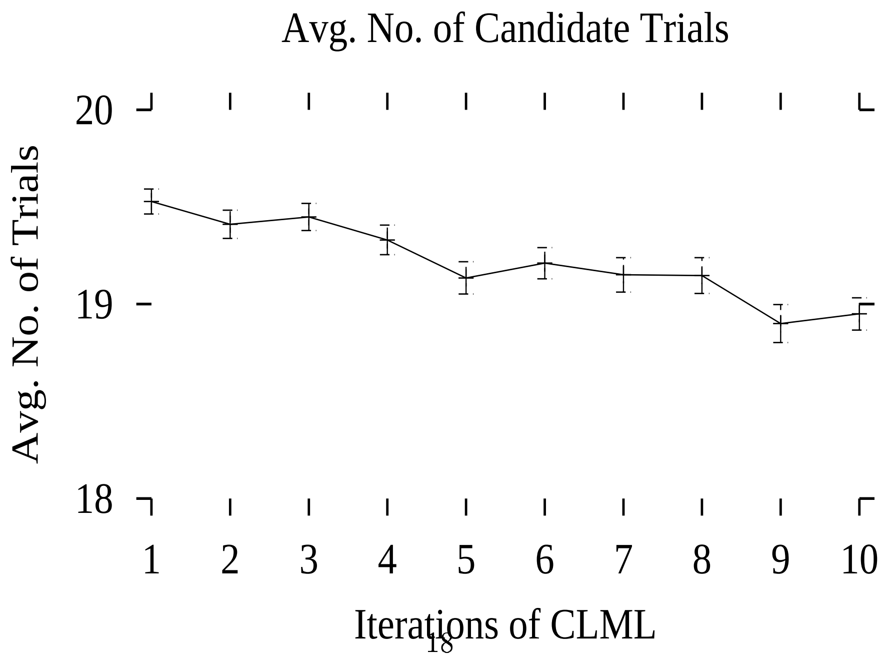


(b)

Figure 6: The Trial Generator run time on: a) Phase A data b) Phase B data.

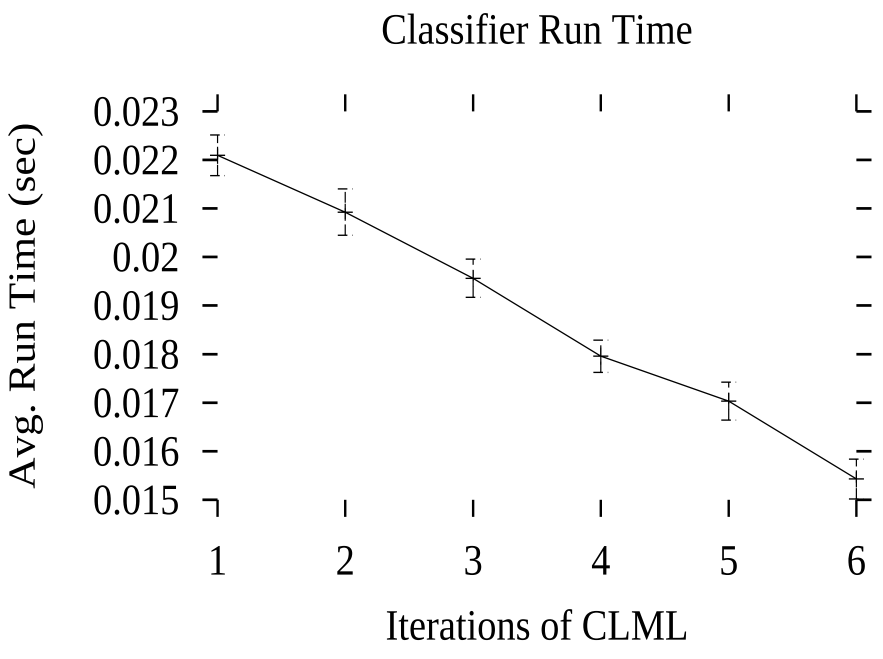


(a)

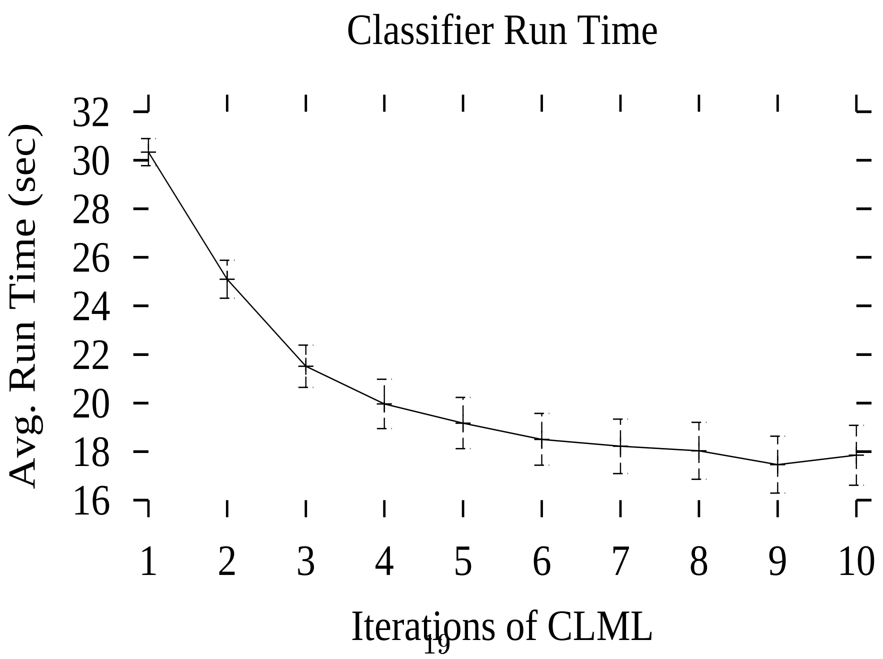


(b)

Figure 7: The average number of the generated trials in each iteration of ASE-Progol on: a) Phase A data b) Phase B data.

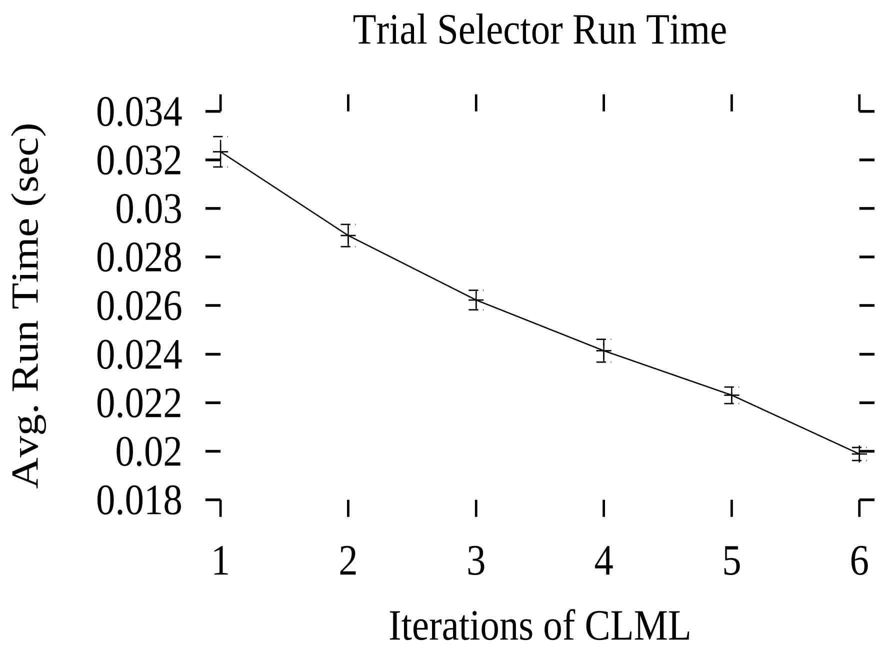


(a)

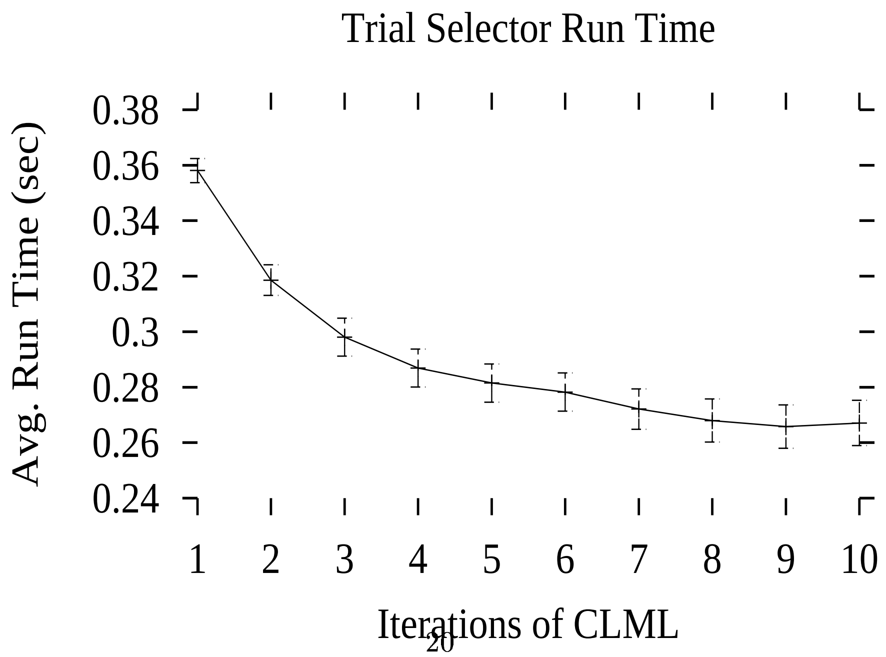


(b)

Figure 8: The Classifier run time on: a) Phase A data b) Phase B data.



(a)



(b)

Figure 9: The Trial Selector run time on: a) Phase A data b) Phase B data.

the run time of the Classifier is significantly higher than the Trial Selector. As mentioned in sections 2.3 and 2.4, the complexity of both the Classifier and the Trial Generator are linear in the number of candidate hypotheses and the number of candidate trials. The average number of candidate hypotheses and the average number of candidate trials in each iteration of the closed loop are shown in Figure 5 and Figure 7 respectively. According to these figures, in the experimentation in phase A the average number of candidate hypotheses has a small change and decreases between 5 and 4 and the average number of candidate trials decreases linearly in the iterations of the closed loop learning. This explains why the average run time of the Classifier and the Trial Selector in the experimentation of phase A are roughly linear in the iterations of the closed loop learning. Whilst, in the experimentation of phase B the average number of candidate hypotheses decreases exponentially between 10 and 5 but the average number of candidate trials has small changes and decrease between 20 and 19. This explains the decrease in the average run time of the Classifier and the Trial Selector in the experimentation of phase B.

Even though both the average run time of the Classifier and the Trial Selector follow a similar pattern, the average run time of the Classifier is significantly higher than the average run time of the Trial Selector. This is because the Classifier involves repeated theorem proving (see section 2.3) which is relatively more expensive in comparison with the computation which is needed in the Trial Selector (see section 2.4).

5 Conclusion

In this report we have provided an experimental analysis of each implemented component of ASE-Progol. The experimental results suggest that the complexity of ASE-Progol is dominated by the Classifier. As discussed earlier, the complexity of the Classifier increases linearly in the number of candidate hypothesis and the number of candidate trials which the latter has an upper bound of 20.

As mentioned earlier, ASE-Progol selects trials to discriminate the space of 'compressive' clauses rather than the space of 'consistent' clauses. It can be shown that the space of compressive clauses has a slower reduction in the subsequent iterations of ASE-Progol than the space of consistent clauses [1].

[1]Note that in each iteration of ASE-Progol a compressive clause h with compression $f_h = p_h - (n_h + c_h)$ can repeatedly become compressive and non-compressive in subsequent iterations where either p_h or n_h is incremented in each iteration.

However, the space of consistent clauses could be very sensitive to noise in the training data. Hence, it is likely that introducing a new control parameter which lets a combination of both compression and consistency of hypotheses can provide a faster convergence in ASE-Progol.

References

[1] Case for support: Closed loop machine learning. ESPRC Research Proposal GR/M56067, 1998.

[2] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and Regression Trees.* Wadsworth, Belmont, 1984.

[3] C. H. Bryant and S. H. Muggleton. Closed loop machine learning. Technical Report YCS 330, University of York, Department of Computer Science, Heslington, York, YO10 5DD, UK., 2000.

[4] C.H. Bryant, S.H. Muggleton, S.G. Oliver, D.B. Kell, P. Reiser, and R.D. King. Combining inductive logic programming, active learning, and robotics to discover the function of genes. In *Machine Intelligence 18.* Electronic Transactions in Artificial Intelligence, 2001. (in press).

[5] S. Muggleton. Inverse entailment and Progol. *New Generation Computing*, 13:245–286, 1995.

[6] S. Muggleton. Stochastic logic programs. *Journal of Logic Programming*, 1999.

[7] S.H. Muggleton and C.H. Bryant. Theory completion using inverse entailment. In *Proc. of the 10th International Workshop on Inductive Logic Programming (ILP-00)*, Berlin, 2000. Springer-Verlag.