Nonmonotonic Learning in Large Biological Networks

Stefano Bragaglia and Oliver Ray

Department of Computer Science, University of Bristol
{stefano.bragaglia,oliver.ray}@bristol.ac.uk

Abstract. This paper introduces a new open-source implementation of a nonmonotonic learning method called XHAIL and shows how it can be used for abductive and inductive inference on metabolic networks that are many times larger than could be handled by the preceding prototype. We summarise several implementation improvements that increase its efficiency and we introduce an extended form of language bias that further increases its usability. We investigate the system's scalability in a case study involving real data previously collected by a *Robot Scientist* and show how it led to the discovery of an error in a whole-organism model of yeast metabolism.

Key words: ILP, ALP, ASP, metabolic networks, completion, revision.

1 Introduction

The task of extending metabolic models to make them consistent with observed data is an important application of ILP which has been tackled in projects such as the *Robot Scientist* [7, 6]. The task of revising (as opposed to merely extending) such models is a nonmonotonic ILP problem addressed by methods like *eXtended Hybrid Abductive Inductive Learning* (XHAIL) [15, 16, 14]. To date, ILP has been applied to relatively small pathways, such as the AAA pathway used in the early *Robot Scientist* work [7], while numerical methods have been applied to much larger models, like the ABER model used in the subsequent work [6]. While a previous XHAIL prototype was able to successfully revise the pathway-specific AAA model in a variety of ways [16, 14], scalability limitations have prevented its application to the whole-organism ABER model until now.

This paper introduces a new implementation of XHAIL which scales from AAA to ABER. Our main contributions are to overhaul the implementation (from a single-threaded `Prolog` program to a multi-threaded *Java* application), to upgrade the inference engine (from the outdated `Smodels` system to the state-of-the-art `Clasp` solver), to support an extended language bias (which allows the weighting of both examples and mode declarations), to revise the logical encoding of metabolism (to better handle cellular compartments), to explore the scalability of the system (with respect to the number of examples, reactions and constraints), to make the new system open-source [1] (under GPLv3), and to use our system to find an error in ABER (and in its influential precursor).

[1] `https://github.com/cathexis-bris-ac-uk/XHAIL`

2 Metabolic Network Modelling

Metabolic networks are collections of interconnected biochemical reactions that mediate the synthesis and breakdown of essential compounds within the cell [8]. These reactions are catalysed by specific enzymes whose amino acid sequences are specified in regions of the host genome called *Open Reading Frames* (ORFs). Particular pathways within a network are controlled by regulating the expression of those genes on which the associated ORFs are located.

Much effort has been invested in the modelling of metabolic networks, leading to numerous qualitative reaction databases (such as the Kyoto Encyclopaedia of Genes and Genomes – KEGG [12]) and quantitative in-silico models (such as `iFF708` [1]). Such resources fall into two classes: highly detailed *pathway-specific* models of small reaction sequences called pathways (often curated by hand); and less detailed *whole-organism* models that synthesise information about all known reactions (typically by computerised data mining of the literature).

These complementary trends are exemplified by a project known as the *Robot Scientist* [7, 6] which integrates laboratory robotics and AI to automate high throughput growth experiments with the yeast *S. cerevisiæ*. The first phase of this project [7] used a pathway-specific model called AAA (of Aromatic Amino Acid bio-synthesis) while the second phase [6] used a whole-organism model called ABER (made at Aberystwyth University), as described below.

2.1 The AAA network

The AAA model is shown in Fig. 1. Nodes in the graph denote metabolites involved in the conversion of the start compound *Glycerate-2-phosphate* to the amino acids *Tyrosine*, *Phenylalanine*, and *Tryptophan*. Arrows denote chemical reactions from substrates to products. For convenience, nodes are annotated with their *Kyoto Encyclopaedia of Genes and Genomes* (KEGG) identifiers (in red); and arrows are labelled with 4-part *Enzyme Commission* (EC) numbers (in blue) and a set of ORF identifiers (in green).

ORFs appearing above each other denote iso-enzymes which all independently catalyse the same reaction (e.g., `YHR137W` and `YGL202W`), while ORFs appearing side by side denote enzyme-complexes built from several gene products that must be present simultaneously (e.g., `YER090W` and `YKL211C`). The dashed brown line shows the inhibition of an enzyme (`YBR249C`) by a metabolite (`C00082`). All reactions take place in the cell cytosol using nutrients imported from the growth medium; and they proceed at a standard rate, except for the import of two underlined compounds (`C01179` and `C00166`), which take longer.

Although this model is based on data in KEGG, its authors manually included extra information from the literature relevant to growth experiments performed by the *Robot Scientist*. As well as modelling slow metabolite imports, enzyme--inhibitions and enzyme-complexes, it includes more details than KEGG about the role of *Indole* as an intermediate compound in the final step of *Tryptophan* synthesis and it ensures all reactions are charge-balanced through the addition of protons denoted by the (non-KEGG) identifier `C00000`.

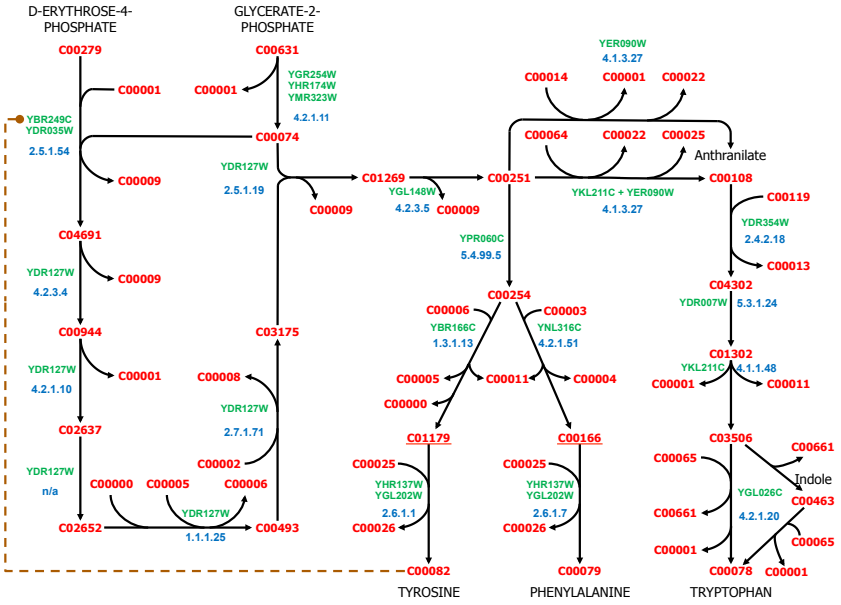

Fig. 1. Pathway-specific metabolic network represented by the AAA model.

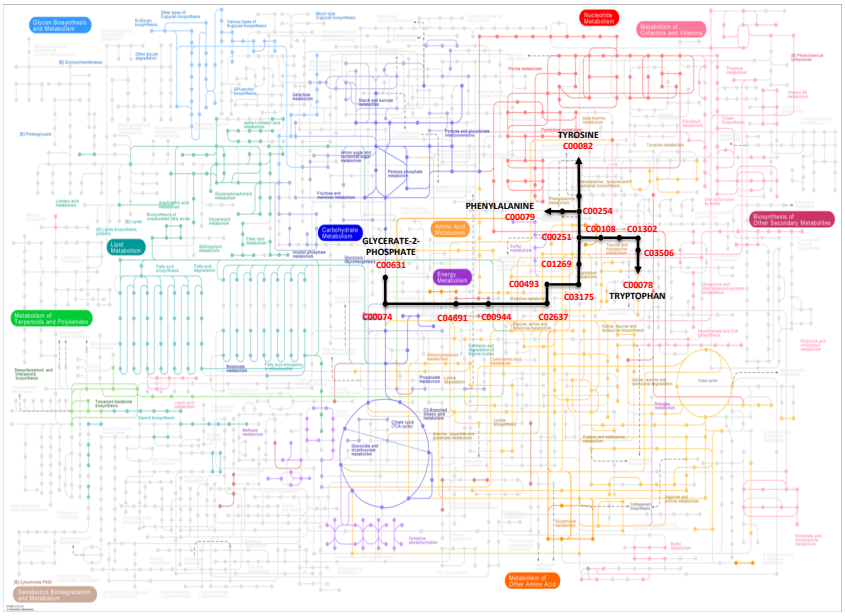

Fig. 2. Whole-organism metabolic map of yeast *S. cerevisiæ* with AAA pathway in black (adapted from an image at `http://www.genome.jp` © Kanehisa Laboratories).

2.2 The ABER network

The AAA pathway is just a tiny part of yeast's metabolism. Fig. 2 shows how the AAA pathway (of about 20 reactions) fits into the overall metabolic map (of about 1000 reactions). This shows the extent of the ABER model, which its authors obtained by augmenting the reactions from an earlier genome-wide flux balance model, called `iFF708`, with reactions from KEGG in order to provide a model with enough detail to allow the presence of essential metabolites to be logically inferred from the starting compounds.

Although ABER is much larger, it lacks some of the finer details of AAA, such as slow metabolite imports, enzyme inhibitions, enzyme complexes and some intermediate reaction steps (as shown in Fig. 5). By contrast, ABER distinguishes reactions occurring in a separate mitochondrial compartment from those in the cytosol (along with membrane reactions that interface between them and the external growth medium). The ABER model also comes with a clearly defined set of ubiquitous and essential compounds.

2.3 A declarative model of metabolism

To reason about the above networks we translated each graph into a set of ground facts in a common ASP representation with a unique identifier for every *metabolite*, ORF, *reaction*, *compartment* and enzyme *complex*. Facts of the form `component(orf,cplx)` and `catalyst(rctn,cplx)` state that an ORF `orf` is part of a complex `cplx` which catalyses the reaction `rctn`; while `substrate(mtbl,cmpt,rctn)` or `product(mtbl,cmpt,rctn)` state that a metabolite `mtbl` (located in a particular compartment `cmpt`) is a substrate or product of the reaction `rctn`.

To account for the fact that some reactions are more certain than others, our representation allows reactions to be flagged as *assertable* (i.e. initially out of the model, but can be included) or *retractable* (i.e. initially in the model, but can be excluded). This is done by declaring a set of facts of the form `assertable_reaction(rctn)` or `retractable_reaction(rctn)` which allow us to consider likely reactions from known reference pathways for inclusion within or exclusion from a revised network for a specific organism.

Facts of the form `inhibitor(cplx,mtbl,cmpt)` state that (when available in high concentrations as an additional nutrient) the presence of metabolite `mtbl` in a compartment `cmpt` inhibits an enzyme complex `cplx`. Facts of the form `start_compound(mtbl,cmpt)` and `ubiquitous_compound(mtbl,cmpt)` indicate that `mtbl` is always present in `cmpt`; while `essential_compound(mtbl,cmpt)` states that the presence of `mtbl` in `cmpt` is essential for cell growth.

We also encode the experimental results which indicate how yeast strains from which certain ORFs have been *knocked out* proliferate over a period of two days in growth media to which particular nutrients have been *added*. Each such experiment is given a unique identifier `expt` and the conditions are recorded by (zero or more) facts of the form `additional_nutrient(expt,mtbl,cmpt)` and `knockout(expt,orf)`. The outcome on some day `day` is given by a literal of the form `predicted_growth(expt,day)` or `not predicted_growth(expt,day)`.

At the heart of our approach is a set of rules for deciding which metabolites are in which compartments on which days in which experiments. These were taken from [16, 14] subject to a few changes: we used predicate `predicted_growth` in our examples instead of using another predicate `observed_growth` forced to have the same extent by integrity constraints with classical negation; we made a distinction between `ubiquitous_compound` and `start_compound`; we added an extra argument into `additional_nutrient`, `essential_compound` and `inhibitor` to localise them all to a given compartment; we added an extra argument into `inhibited` to localise its effect to a given day [2]; and we modelled slow imports by inhibiting the relevant import reactions on the first day of each experiment. As these changes are small, we refer the reader to [16, 14] for more details.

In brief, predicting the outcome of a given experiment amounts to checking if all essential metabolites are present in their required compartments. Apart from start compounds, ubiquitous compounds and additional nutrients, the only way a metabolite can be in a compartment is if it gets there as the product of an active reaction – whose substrates are all in their respective compartments, which has not been excluded (if it is retractable) or has been included (if it is assertable), and which is catalysed by a complex which is not inhibited on the day in question, and which has not been deleted (by having any of its component ORFs knocked out). This is represented by a set of ground rules like the following (which is one of two rules produced for reaction `2.5.1.19`, assuming it has been given the identifier `31` and a retractable status):

```
in_compartment(Expt,"C01269",cytosol,Day) :-
    in_compartment(Expt,"C00074",cytosol,Day),
    in_compartment(Expt,"C03175",cytosol,Day),
    catalyst(31,Cplx),
    not inhibited(Expt,Cplx,Day),
    not deleted(Expt,Cplx),
    not exclude(31).
```

3 Metabolic Network Revision

The fact that all the models we have mentioned so far (ABER, AAA, KEGG and `iFF708`) differ in the details of even just the tiny pathway shown above, and the fact that none of them fully agrees with experimental data, suggests that they are all approximations which are incomplete and/or incorrect. This raises the question of how such networks can be usefully revised (as was done manually in the creation of ABER and AAA). Prior work has shown that ILP can help to automate this process. In particular, the systems `Progol5` and XHAIL have been used to complete and revise the AAA model, as described below.

[2] In fact this change was already present in the model used in [16] but the description in the paper incorrectly reproduced an earlier version of the rule from [14].

3.1 ILP

This paper assumes a familiarity with *Inductive Logic Programming* (ILP) [11]. An atom a is a predicate p applied to a collection of terms $(t_1, \ldots, t_n)$. A literal l is an atom a (positive literal) or the negation of an atom *not* a (negative literal). A clause C is an expression of the form $a \ :- \ l_1, \ldots, l_m$ where a is an atom (head atom) and the l_i are literals (body literals). A clause with no body literals is called a fact and a clause with no head atom is called a constraint. A logic program P is a set of clauses. In this paper, the meaning of a logic program is given by the Answer Set or Stable Model semantics [3]. A program P entails a set of ground literals L, denoted $P \models L$, if there is an answer set of P that satisfies all of the literals l in L. A logic program H entails a set of ground literals E with respect to a logic program B if $B \cup H \models E$.

The task of ILP is to find a hypothesis H that entails some examples E with respect to a background theory B and also satisfies some form of language and search *bias*. The search bias typically used is that of minimising *compression* [9] of the hypothesis with respect to the examples (where compression is the number of literals in the hypothesis minus the number of examples covered). The language bias typically used is that of specifying a set of *mode declarations* [9] that allow the user to impose various constraints on which literals may appear in the heads and bodies of hypothesis clauses (as illustrated later).

3.2 Progol5

`Progol5` [9, 10] is a popular ILP system which was used in the first phase of the *Robot Scientist* work to extend the AAA model with missing ORF-enzyme mappings [7]. It uses a greedy covering approach that explains one example at a time by constructing and generalising a ground clause called a *Bottom Set* (BS) that efficiently bounds the search space. First the head of the BS is computed by a contrapositive method, then the body of the BS is computed by a saturation method, and finally the BS is generalised by a subsumption search.

While `Progol5` has been used to monotonically extend AAA with missing information, its limited ability to reason abductively or inductively through negation means it is not well-suited for tackling revision problems – which are inherently nonmonotonic. Its greedy covering approach assumes the addition of a future hypothesis cannot undermine the truth of an earlier example and, even in this limited setting, it is (semantically and procedurally) incomplete [13].

3.3 XHAIL

eXtended Hybrid Abductive Inductive Learning (XHAIL) [15] overcomes the incompleteness of `Progol5` (at the expense of solving a harder problem). For soundness it does not process examples one-by-one, but solves them all at once. For completeness it does not generalise a single BS, but a *set* of such clauses, K, known as a *Kernel Set* (KS). For efficiency it uses iterative deepening to consider progressively larger Kernel Sets.

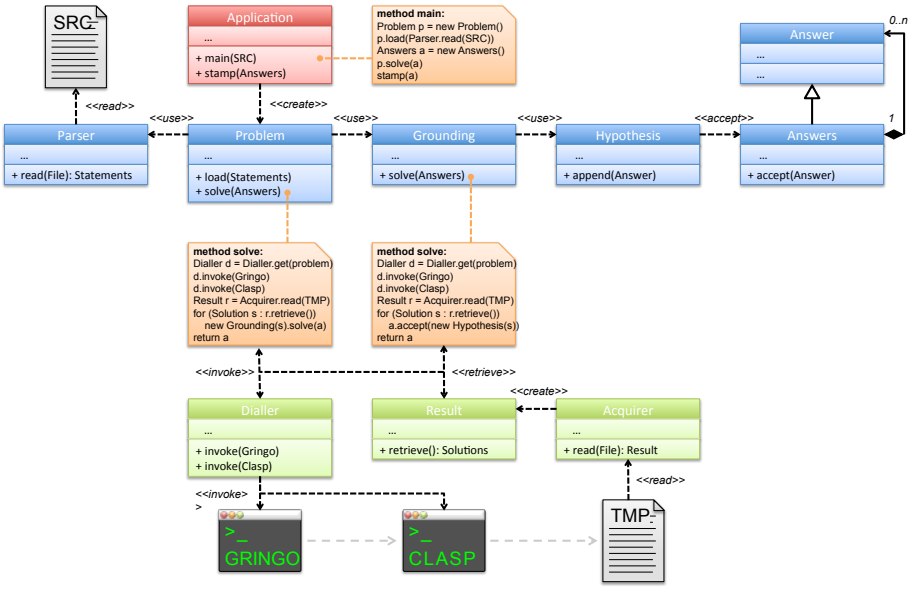

Fig. 3. Architectural overview of our new XHAIL implementation.

Existing Prototype While its semantics is defined using the *Generalised Stable Model* (GSM) notion from *Abductive Logic Programming* (ALP) [5], the existing XHAIL prototype uses *Answer Set Programming* (ASP) [3] to perform the three key phases of the algorithm. Firstly, the head atoms of K are computed *abductively* to give a set of ground atoms Δ that entail E with respect to B. Then, the body atoms of K are computed *deductively* to give a set of ground atoms Γ entailed by $B \cup \Delta$. Finally, K is generalised *inductively* to give a compressive hypothesis H subsuming K.

As described in [16] the existing prototype, which used `Smodels` as the ASP system [17], was able to successfully revise AAA to make it consistent with the *Robot Scientist* data from [7] including a number of results known to contradict the final AAA model (as noted in *Section 3* of the *supplementary material* for [7]). Unfortunately, scalability limitations meant the existing prototype could not be used on the ABER. Hence, in this paper one of our key aims was to develop a new implementation of XHAIL that can be applied to ABER.

New Architecture To make our new system portable, extensible and efficient, we wrote it from scratch in *Java 8* using a modular architecture that exploits multi-threading and has been made open-source under *GNU General Public License v3.0* (GPLv3). We also upgraded the ASP engine from the (now unsupported) `Smodels` to the (current ASP competition winner) `Clasp` [2] and, to make the system more usable and noise-tolerant, we also introduced an extended language bias that allows mode declarations and examples to be weighted.

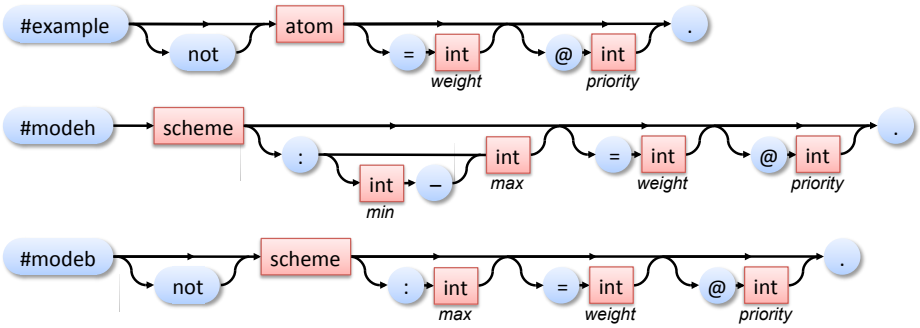

Fig. 4. Syntax diagrams for the new XHAIL's extended language bias.

An overview of the system is given in Figure 3. The `Application` class acts as the user interface. When invoked, it generates an instance of a `Problem` and activates a bespoke `Parser` (which can recognise both `Clasp` and XHAIL-specific statements) to load the desired knowledge base.

The `solve` method in `Problem` triggers the procedure by invoking a `Dialler` on an instance of the `Problem`. The `Dialler` calls `Clasp` and collects the results in a temporary file (interpreted by a specialised parser `Acquirer`) to return the set of solutions as a `Result`. The `Problem` and `Clasp` are loosely-coupled so they can be invoked as separate threads to achieve more parallelism. Now, at the end of the (*abductive*) stage, the `Result` contains the head atoms Δ of the KS K.

An instance of `Grounding` is then used to *deductively* derive the body atoms Γ of K. Its `solve` method is called to activate the `Dialler` on the current `Grounding` to perform *induction*. The `Result` read at this point from the temporary file is used to generate as many `Hypotheses` as needed, each representing an H. Each `Hypothesis` found in this fashion is converted into an `Answer` and appended to the set of `Answers`. If any weights have been applied to examples or mode declarations, each `Answer` will have a score, and any non-optimal ones will be discarded. Finally, the `Application` presents the `Answers` to the user.

New Language Bias As shown in Fig. 4 we have extended the new XHAIL's language bias to allow integer weights and priorities (familiar concepts in ASP) to be associated with individual examples and mode declarations. Adding a weight to an example has the effect of making that example defeasible (so the example can remain uncovered at the expense of paying a penalty set by the weight). Adding a weight to a mode declaration has the effect of incurring a penalty set by the weight every time the mode declaration is used.

The overall effect of these weights is to give preference to certain hypotheses over others; and the task of the new XHAIL is to find hypotheses that incur the smallest penalty. We believe this offers the user a finer level of control than the default compression metric. Of course, the standard bounds on the maximum and minimum number of times a mode declaration can be used are still available.

4 Case Study

To validate the approach, we applied our new XHAIL system to our logical encoding of the ABER model. This was done in two stages. The first stage was to revise the ABER model to incorporate the extra information in the more detailed AAA model. The second stage was to learn some additional hypotheses (originally found in [16]) to make the model consistent with real *Robot Scientist* data. In both cases, we studied how the approach scaled to progressively bigger fragments of the model with increasingly large sets of observations. We began with the fragment shown in Fig. 5, where the purple highlights show the four pieces of information we wanted to learn in the first validation phase (to make this sub-network of ABER functionally equivalent to AAA).

4.1 Making ABER consistent with AAA

To revise ABER with the information from AAA that it lacks, we created suitable *examples* and *mode declarations* for the four tasks described below. The XHAIL code for these is shown in Listing 1 (Tasks *A*-*D*). We also added a further task (Task *E*) to illustrate how our system can handle noise. In addition to considering these tasks individually, we also considered various combinations of them in order to show the utility of our (optional) extended language bias.

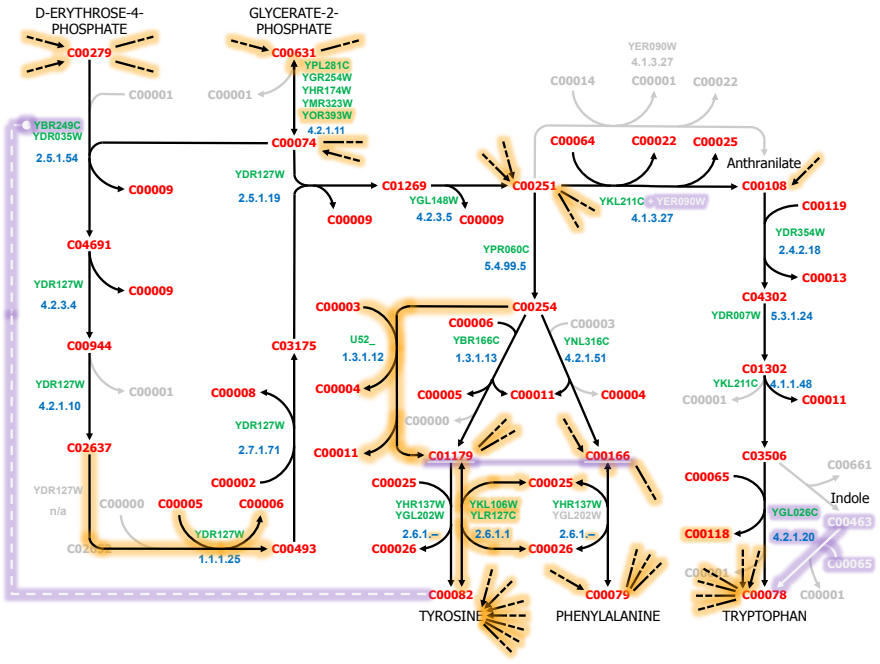

Fig. 5. Fragment of ABER corresponding to AAA pathway with missing information in grey and other changes highlighted in orange and purple.

Task A (lines $1-6$) consists of two experiments that allow XHAIL to infer that YER090W is required in all enzyme complexes catalysing reaction 4.1.3.27. *Experiment 1* simply knocks out YER090W. In this case the AAA model predicts no growth since all enzymes producing *Anthranilate* are blocked and *Tryptophan* cannot be produced. By contrast, ABER predicts growth as it has that YKL211C can catalyse 4.1.3.27 alone. XHAIL proposes several minimal hypotheses which restore consistency by complexing YER090W with various enzymes, but the introduction of *experiment 2*, which simply adds the additional nutrient *Anthranilate* to this knockout restricts the possibilities to just one, YKL211C.

Task B (lines $7-12$) consists of two experiments that allow XHAIL to infer that *Tyrosine*, when used as an additional nutrient, inhibits the enzyme YBR249C catalysing reaction 2.5.1.54. *Experiment 3* simply knocks out the other enzyme YDR035W catalysing this reaction and adds the nutrient *Tyrosine*. While AAA predicts no growth since the only available enzyme for 2.5.1.54 is inhibited, ABER predicts growth as it has no information about inhibition. XHAIL again proposes several hypotheses which restore consistency by inhibiting various enzymes, but the addition of *experiment 4*, which simply adds *Tyrosine* to the wild type restricts the possibilities to just the intended one, YBR249C.

```
1  % Task A: YER090W as enzyme complex in 4.1.3.27
2  knockout(1,"YER090W").
3  #example not predicted_growth(1,1) =4.
4  knockout(2,"YER090W").additional_nutrient(2,"C00108",medium).
5  #example predicted_growth(2,1) =4.
6  #modeh component(#orf,#enzymeID) :1 =3.
7  % Task B: C00082 inhibits YBR249C in 2.5.1.54
8  knockout(3,"YDR035W").additional_nutrient(3,"C00082",medium).
9  #example not predicted_growth(3,1) =4.
10 additional_nutrient(4,"C00082",medium).
11 #example predicted_growth(4,1) =4.
12 #modeh inhibitor(#enzymeID,#metabolite,cytosol) :1 =2.
13 % Task C: C00463 contamination in 4.2.1.20
14 knockout(5,"YKL211C").additional_nutrient(5,"C00463",medium).
15 #example predicted_growth(5,1) =4.
16 knockout(6,"YGL026C").additional_nutrient(6,"C00463",medium).
17 #example not predicted_growth(6,1) =4.
18 knockout(7,"YKL211C").
19 #example not predicted_growth(7,1) =4.
20 #modeh include(#assertable_reaction) :1 =2.
21 #modeh catalyst(#assertable_reaction,#enzymeID) :1 =1.
22 % Task D: slow import of C00166 and C01179
23 knockout(8,"YBR166C").additional_nutrient(8,"C01179",medium).
24 #example not predicted_growth(8,1) =4.
25 knockout(9,"YNL316C").additional_nutrient(9,"C00166",medium).
26 #example not predicted_growth(9,1) =4.
27 #example predicted_growth(10,1) =4.
28 #modeh inhibited(+experiment,#enzymeID,#day) :2 =1.
29 % Task E: defeasible example
30 #example not predicted_growth(11,1) =4.
```

Listing 1. *Tasks A-E* with optional weights and constraints highlighted.

Tasks A and B (lines 1 – 12) can both be given to XHAIL at the same time. In this case it successfully learns the union of the two individual hypotheses, but it also learns several other hypotheses resulting from the interaction of the two tasks. The additional hypotheses all introduce two enzyme complexes, as opposed to one enzyme complex and one inhibitor. Instead of adding further examples to eliminate the unwanted hypotheses, we can exploit the ability of our new system to specify weights on mode declarations. By giving a lower cost (`=2`) to the mode declaration for inhibitors (line 12) than the cost (`=3`) for the mode declaration for complex components (line 6), we ensure only the intended hypothesis has minimal cost and is returned by XHAIL. Alternatively we can set the number of times each mode declaration is used to exactly once (`:1`). As shown below, such biases reduce the search space and the execution time.

Tasks C and D (lines 13-28) are similarly designed to rediscover a missing reaction for `C00463` along with the slow import of `C00166` and `C01179`.

Task E was also added as an obviously incorrect example (as it simply asserts that the wild type does not grow under normal conditions – which is false). Ordinarily, this would result in logical inconsistency, but by exploiting the ability of XHAIL to give defeasible weights to *examples*, we can still learn all the intended changes when running *tasks E* by paying a simple cost (`=4`) associated with this last example. Although the inconsistent *example* is discarded, XHAIL finds it is cheaper to construct hypotheses that explain all the other examples (as opposed to paying the penalty for discarding them).

Table 1 shows the time taken to run these tasks on various fragments of the ABER model. The x-axis shows the 5 *tasks A-E*. The y-axis shows the increasing sizes of model we used, starting with the initial network in Fig. 5 (which has 52 reactions including imports) all the way up to the whole model (with $1,106$ reactions). Experiments were run on a MacBook Pro featuring a 2.3 GHz Intel Core i7 with 4 cores and 16 GB DDR3 RAM.

Size	Task A		Task B		Task C		Task D		Task E	
52	0.3	(1)	0.4	(1)	0.3	(1)	7.2	(1)	0.166	(0)
87	0.5	(1)	0.7	(1)	0.7	(1)	10.5	(1)	0.214	(0)
127	0.8	(1)	1.9	(1)	1.5	(1)	18.7	(1)	0.294	(0)
207	1.8	(1)	3.9	(1)	2.3	(1)	39.7	(1)	0.449	(0)
367	4.8	(1)	6.0	(1)	3.9	(1)	76.3	(1)	0.814	(0)
687	14.1	(1)	15.6	(1)	9.5	(1)	186.0	(1)	1.802	(0)
1106	38.3	(1)	59.7	(1)	17.6	(1)	339.1	(0)	3.898	(0)
52	0.3	(1)	0.3	(1)	0.3	(1)	7.5	(1)	0.169	(1)
87	0.5	(1)	0.5	(1)	0.4	(1)	11.0	(1)	0.22	(1)
127	0.8	(1)	1.0	(1)	0.6	(1)	19.4	(1)	0.295	(1)
207	1.8	(1)	2.0	(1)	1.1	(1)	39.9	(1)	0.44	(1)
367	5.2	(1)	5.7	(1)	2.5	(1)	72.5	(1)	0.793	(1)
687	14.2	(1)	13.7	(1)	5.8	(1)	180.4	(1)	1.726	(1)
1106	38.4	(1)	29.4	(1)	14.7	(1)	342.8	(181)	3.827	(1)

Table 1. Execution time (and number of hypotheses) vs. number of tasks and reactions as optional language bias is excluded (top half, shaded blue) or included (bottom half, shaded green).

The top half of the table refers to experiments using simple *mode declarations* without number constraints or weights, while the bottom half refers to experiments using our enhanced *mode declarations* with number constraints and weights. Each cell contains the execution time in seconds together with the number of answers returned (in parentheses). These tests show that the experiments become more computationally intensive as the number of reactions and tasks increases; and they also show that the language bias effectively helps to narrow down the number of answers and to reduce the execution times.

Fig. 6 describes what happens during the first 2 minutes of execution when XHAIL tackles a problem, specifically all the *tasks A–D* simultaneously run on the ABER fragment in Fig. 5. XHAIL still finds the expected hypotheses, but also several others resulting from the interaction between the *tasks*, making it a much harder problem to solve. In order to eliminate the unwanted hypotheses and show the utility of our extended language bias, we have introduced further constraints and weights on the mode declarations of the problem (as highlighted in yellow in Listing 1). This graph compares the optimality score and the elapsed time of each answer found when the optional language bias is excluded (in red) or included (in green) with respect to the score expected for the optimal solution (in blue). The triangles indicate when the parsing and generation of the abductive phase problem terminates and `Gringo`'s grounding starts. Similarly, the squares show when the grounding is over and `Clasp` takes over. Finally the circles show plot the improvement in the score of the current best solution (y-axis) against time (x-axis). The lines preceding the first answers are dashed because at that point is not possible to know in advance the first optimality value but it is convenient to show where triangles and squares sit. Notice how the extended language bias results in higher quality answers being found in less time.

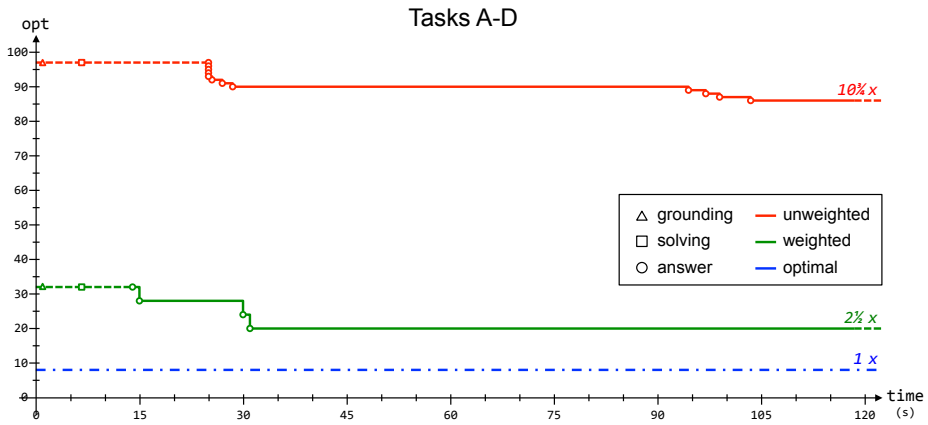

Fig. 6. Elapsed times vs. scores of answers found during resolution with language bias excluded (in red) or included (in green) with respect to the optimal (in blue).

4.2 Making ABER consistent with *Robot Scientist* data

We used the revised ABER model to run 40 additional experiments for which growth data had been previously obtained by a *Robot Scientist*. We set the mode declarations as follows (where the highlighted expressions :min-max denote optional bias which constrains how many times each mode declaration is used):

```
#modeh inhibited(+experiment,#modifiable_enzyme,#day) :0-2.
#modeh additional_nutrient(+experiment,#metabolite,medium) :0-1.
#modeb additional_nutrient(+experiment,#metabolite,medium) :0-1.
```

This allows XHAIL to learn the hypothesis from [16] consisting of the clause "`additional_nutrient(E,"C00078",medium) :- additional_nutrient(E, "C00463", medium)`" – which suggests the contamination of *Indole* with *Tryptophan* (as the latter becomes an additional nutrient whenever the former is) – and a fact "`inhibited(E,10,1)`" – which suggests the slow import of *Anthranilate* (as the hypothetical enzyme catalysing its import is inhibited on the first day of each experiment). The time taken to run more and more experiments on larger and larger networks is shown in Table 2, which is formatted like Table 1 with the shading of each cell increasing with execution time.

It is interesting to note that the number of solutions drops from 1 to 0 when experiments $1-8$ are run on the fragment of ABER with 687 reactions (without the optional bias). We suspected one of the additional reactions was interfering with the initial AAA fragment so we added an extra mode declaration to allow XHAIL to exclude zero or more reactions. This led to a hypothesis which excluded one reaction with EC `1.3.1.12` and ORF `U52_` which, although it is shown in Fig. 5, was in fact accidentally left out of the initial fragment because we did not notice it when carrying out our initial analysis by hand.

Since we could not find any hypothesis in which that reaction was not excluded, we examined the literature support for this reaction and found that, while it does exist in the KEGG reference pathway, it does not exist in any of the pathways specific to yeast. We then determined that it was imported into ABER from `iFF708`. In this way, we suggest our method has led to the discovery of a mistake in both ABER and `iFF708`.

Size	Ex 1-4		Ex 1-8		Ex 1-16		Ex 1-32		Ex 1-40	
52	0.2	(1)	0.8	(1)	3.6	(1)	19.1	(1)	65.8	(10)
87	0.5	(1)	3.5	(1)	18.3	(1)	110.2	(1)	≥600	(≥1)
127	0.8	(1)	7.1	(1)	64.5	(1)	510.3	(1)	≥600	(≥1)
207	1.8	(1)	12.5	(1)	80.0	(1)	≥600	(≥1)	≥600	(≥1)
367	3.5	(1)	26.3	(1)	265.0	(1)	≥600	(≥1)	≥600	(≥1)
687	8.1	(1)	101.9	(0)	≥600	(≥1)	≥600	(≥1)	≥600	(≥1)
1106	14.5	(1)	262.0	(0)	≥600	(≥1)	≥600	(≥1)	≥600	(≥1)
52	0.2	(1)	0.6	(1)	1.5	(1)	5.8	(1)	11.4	(10)
87	0.5	(1)	1.6	(1)	5.4	(1)	14.5	(1)	40.0	(10)
127	0.8	(1)	3.5	(1)	12.9	(1)	43.4	(1)	202.4	(10)
207	1.8	(1)	8.7	(1)	42.0	(1)	234.4	(1)	440.1	(10)
367	3.9	(1)	19.5	(1)	≥600	(≥1)	≥600	(≥1)	≥600	(≥1)
687	8.3	(1)	110.1	(24)	531.8	(4)	≥600	(≥1)	≥600	(≥1)
1106	15.6	(1)	300.5	(24)	≥600	(≥1)	≥600	(≥1)	≥600	(≥1)

Table 2. Execution time (and number of hypotheses) vs. number of experiments and reactions as optional language bias is excluded (top half, shaded blue) or included (bottom half, shaded green).

5 Comparison with Progol5

It is interesting to see why the Progol5 system used in the early *Robot Scientist* work, cannot be used on tasks like those solved in Section 4. For this purpose, it suffices to consider the following Progol5 input file which simplifies the task of including and excluding reactions to its absolute bare principles:

```
% Background Knowledge
product(m,ass).
product(n,ret).
present(MN) :- product(MN,ass), include(ass).
present(MN) :- product(MN,ret), not exclude(ret).
% Examples
:- observable(present/1)?
present(m).
:- present(n).
% Language Bias
:- modeh(1,include(ass))?
:- modeh(1,exclude(ret))?
```

The background theory declares a metabolite `m` as the `product` of an assertable reaction `ass` and a metabolite `n` as the `product` of a retractable reaction `ret`. It then states the `product` of an assertable reaction will be `present` if the reaction is `included`; while the `product` of a retractable will be `present` if the reaction is *not* `excluded`. The bias allows Progol5 to `include` an assertable reaction and/or to `exclude` a retractable reaction.

While XHAIL easily learns the hypothesis `include(ass)` and `exclude(ret)` from these inputs, Progol5 simply states that this input is inconsistent and it fails to learn anything. This is because the example atom `present(m)` depends negatively upon the intended hypothesis `exclude(ret)`, which is therefore not amenable to Progol5's limited form reasoning with negation.

It might seem Progol5 can be made to find the answer by transforming this nonmonotonic problem into a monotonic one by reifying the literals `include(ass)` and `not exclude(ret)` to the atoms `include(ass,true)` and `exclude(ret,false)` – while adding the following constraints to avoid meaningless solutions where a reaction is both included and not included at the same time (which is the very approach taken in another Progol5 application called Metalog [18]):

```
:- include(R,true),include(R,false).
:- exclude(R,true),exclude(R,false).
```

Although Progol5 can now learn one of the hypotheses `include(ass,true)`, this has the semantically problematic result of requiring the exclusion of the retractable reaction to be neither `true` nor `false`. And this is unavoidable as adding the following complementary constraints to force the issue merely re-establishes the logical inconsistency that the reification was introduced to avoid:

```
:- not include(R,true), not include(R,false).
:- not exclude(R,true), not exclude(R,false).
```

It can be shown that any semantically meaningful solution to this problem will require explicitly formalising the rules to determine when a compound is *not* in a compartment – which is much harder than formalising when a compound is in a compartment and is not practical in more realistic versions of this task.

6 Conclusions

This paper introduced a new implementation of XHAIL that is able to usefully revise a whole-organism model of yeast metabolism. It summarised some of the key implementation improvements and investigated their scalability with respect to the number of examples and the size of the network. It also introduced some new language bias extensions and analysed their impact on execution times and the number of hypotheses returned. The new system is much faster than the old prototype and can be applied to much larger networks.

To the best of our knowledge, this is the first application of ILP to a whole-organism model of metabolism; and it has led to the discovery of a reaction that we believe is incorrectly included in both ABER and its influential precursor `iFF708` (which is the forerunner of most whole-organism flux-balance models of yeast developed to this day). These preliminary results suggest that XHAIL is a useful method for biological network revision (and probably for other non-monotonic ILP tasks as well). Both the system and all of the input files used in this work have been made fully open-source under GPLv3.

In future work we plan to investigate in more detail how our execution times are distributed across the different phases (adductive, deductive and inductive) of XHAIL and to exploit a parallel implementation of `Clasp` to help further speed up execution of our system. We also need to test the system in harder case studies where the required language bias is not set in advance and see if we can apply our method to more recent models of yeast metabolism, such as `Yeast5` [4], which are not designed to be logically executable.

Acknowledgments

This work is supported by EPSRC grant *EP/K035959/1*.

References

1. Förster, J., Famili, I., Fu, P., Palsson, B., Nielsen, J.: Genome-scale reconstruction of the saccharomyces cerevisiae metabolic network. Gen. Res. 13(2), 244–53 (2003)
2. Gebser, M., Kaufmann, B., Neumann, A., Schaub, T.: clasp: A conflict-driven answer set solver. In: Logic Programming and Nonmonotonic Reasoning, pp. 260–265. Springer (2007)
3. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Comp. 9(3/4), 365–386 (1991)
4. Heavner, B.D., et al.: Yeast 5 an expanded reconstruction of the saccharomyces cerevisiae metabolic network. BMC Systems Biology 6(55) (2012)
5. Kakas, A., Kowalski, R., Toni, F.: Abductive Logic Programming. Journal of Logic and Computation 2(6), 719–770 (1992)
6. King, R.D., Rowland, J., Oliver, S.G., Young, M., Aubrey, W., Byrne, E., Liakata, M., Markham, M., Pir, P., Soldatova, L.N.: The automation of science. Science 324(5923), 85–89 (2009)

7. King, R.D., Whelan, K.E., Jones, F.M., Reiser, P.G., Bryant, C.H., Muggleton, S.H., Kell, D.B., Oliver, S.G.: Functional genomic hypothesis generation and experimentation by a robot scientist. Nature 427(6971), 247–252 (2004)
8. Lehninger, A.: Biochemistry: The Molecular Basis of Cell Structure and Function. Worth Publishers, 2 edn. (1979)
9. Muggleton, S.: Inverse entailment and Progol. New Gen. Comp. 13, 245–286 (1995)
10. Muggleton, S., Bryant, C.: Theory Completion Using Inverse Entailment. In: Proc. of the 10th Int. Conf. on ILP, LNCS, vol. 1866, pp. 130–146. Springer (2000)
11. Muggleton, S., De Raedt, L.: Inductive Logic Programming: Theory and Methods. Journal of Logic Programming 19,20, 629–679 (1994)
12. Ogata, H., Goto, S., Sato, K., Fujibuchi, W., Bono, H., Kanehisa, M.: Kegg: Kyoto encyclopedia of genes and genomes. Nucleic Acids Res. 27(1), 29–34 (1999)
13. Ray, O.: Hybrid Abductive-Inductive Learning. Ph.D. thesis, Department of Computing, Imperial College London, UK (2005)
14. Ray, O., Whelan, K., King, R.: A nonmonotonic logical approach for modelling and revising metabolic networks. In: Proc. 3rd Int. Conf. on Complex, Intelligent and Software Intensive Systems. pp. 825–829. IEEE (2009)
15. Ray, O.: Nonmonotonic abductive inductive learning. Journal of Applied Logic 7(3), 329–340 (2009)
16. Ray, O., Whelan, K., King, R.: Automatic revision of metabolic networks through logical analysis of experimental data. In: Proc. of the 19th Int. Conf. on ILP. pp. 194–201. Springer (2010)
17. Simons, P., Niemelä, I., Soininen, T.: Extending and implementing the stable model semantics. Artificial Intel. 138(1-2), 181–234 (2002)
18. Tamaddoni-Nezhad, A., Kakas, A., Muggleton, S., Pazos, F.: Modelling inhibition in metabolic pathways through abduction and induction. In: Proc. of the 14th Int. Conf. on ILP. LNCS, vol. 3194, pp. 305–322. Springer (2004)