Slps for Probabilistic Pathways: Modelling and Parameter Estimation

Nicos Angelopoulos and Stephen Muggleton
Department of Computing,
Imperial College,
South Kensington, London.
{nicos,shm}@doc.ic.ac.uk

August 9, 2002

Contents

Plots

Tables

Chapter 1

Introduction

This document describes experimental results derived from running the Failure Adjusted Maximisation, FAM, algorithm (Cussens, 2001) on some biologically inspired Slps, Stochastic Logic Programs (Muggleton, 1996; Cussens, 2000).

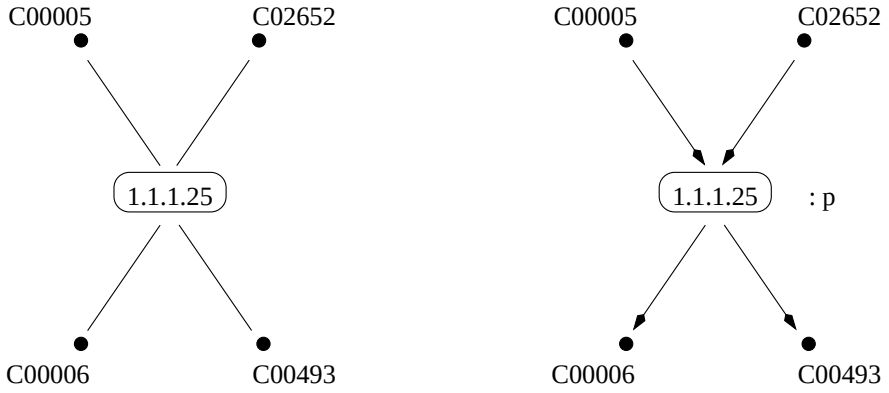
1.1 Probabilistic Pathways

We extend a Logic Program (LP) used in (Bryant, Muggleton, Oliver, Kell, Reiser, & King, 2001) for modelling biological Pathways to incorporate reasoning with uncertainty. Slps are used to encode this information.

Pathways are graph approximations of biological functions. They are used to encapsulate current knowledge about cellular interactions. A simple node of the aromatic amino acid pathway for Yeast, presented in (Bryant et al., 2001) is given in Figure 1.1(a).

Metabolites $C00005$ with $C02652$ and $C00006$ with $C00493$ can be, irrespectively, the inputs and outputs of the reaction facilitated by enzyme 1.1.1.2. In principle, enzymes facilitate reaction in both directions. The probabilistic version, Fig 1.1(b), requires that reactions occur in one direction. This is because the semantics of the attached probability are "Given the inputs are present, the reaction will happen with probability p". Note that the probability is attached to the reaction, Fig 1.1(c), not to the enzyme. Thus, when we need both the directions we have to explicitly attach each direction to the relevant enzyme. In Machine Learning one is not very often interested in both directions of enzyme reactions, because a set of input metabolites is usually given and the focus is in whether certain outputs can be produced.

In general attaching a probability to a reaction can account for a multitude of reasons for which a reaction might not always occur. For example, rates of reactions in relation to availability of metabolites, competing reactions, secondary paths and variations of physical chemistry assumptions.




```
enzyme( '1.1.1.25', rea_1_1_1_25, [c00005,c02652], [c00006,c00493] ).
0.80 :: rea_1_1_1_25( yes, yes, yes, yes ).
0.20 :: rea_1_1_1_25( yes, yes, no, no ).
```

(c)

Figure 1.1: At top, single node of two pathways. Top left, the non-probabilistic case. Top right, reaction has direction and probability. Bottom, the Slp code for the probabilistic case.

1.2 Experimental set-up.

One of the objectives is to compare performance of FAM when non determinacy is involved. Thus, the experiments were run twice. Once against a pathway Slp that is deterministic, version 6 (v6, see Appendix A). Version 6, is, in effect, a node/edge representation of a chain network of length 21. The second Slp to consider, version 7 (v7, Appendix B), enhances version 6 with an, extra, alternative path of length 5. Probability p_j of a reaction attached to some node, mean that the reaction will happen with probability p_j while it will not with probability $1 - p_j$.

The core parameter estimation (PE) scenario is as follows. An Slp with its N true parameters $p = \langle p_1, p_2, \ldots, p_N \rangle$ is used to sample S samples of a single top-goal. The parameters are then replaced by uniformly distributed ones. Fam is then run to obtain $\bar{p} = \langle \bar{p}_1, \bar{p}_2, \ldots, \bar{p}_N \rangle$. We use the software available from (Angelopoulos & Cussens, 2001) to perform the sampling and the parameter estimation.

There are two measures of interest. (a) how good an estimate of $p, \bar{p}$ is, and (b) how long it took to find. All calls to Fam were made with a termination condition of $1.0e-09$ for the log likelihood. In all cases the corresponding $\bar{p}$

Table 1.1 Dimensions of all experiments.

x	t	I_x^t	$S_x^{l^t}$	$S_x^{u^t}$	$S_x^{i^t}$
a	1	10	100	1000	100
	2	20	110	1010	100
b	1	10	200	2000	200
	2	20	210	2010	200
c	1	5	100	3300	400
	2	10	110	3310	400
d	1	20	100	4900	600
	2	40	150	4950	600

was found within a single iteration.

Comparisons and comments are made about experiments of the following form. Each experiment E_x is identified by a single lower case letter from the beginning of the alphabet. Here we mostly present results on experiments E_a and E_c. There are two sets of numbers ($t \in \{1, 2\}$) associated with each E_X. Number of iterations (I_x^t) a samples lower limit ($S_x^{l^t}$), a samples upper limit ($S_x^{u^t}$) and a samples step interval ($S_x^{i^t}$). For example the values for E_a are $I_a^1 = 10, S_a^{l^1} = 100, S_a^{u^1} = 1000, S_a^{i^1} = 100$ and $I_a^2 = 20, S_a^{l^2} = 110, S_a^{u^2} = 1010, S_a^{i^2} = 100$. See Table 1.2 for the dimensions of the other experiments considered. The intuition is that $S_x^{i^1} = S_x^{i^2}$ while $I_x^1 \neq I_x^2$ so to compare the effect of multiple iterations on the mean errors, and $S_x^{l^1}$ very near but not equal to $S_x^{l^2}$ so both sets can be viewed on a single graph without the error bars being cluttered.

For experiment x and index t, FAM is run I_x^t times for each S in $Sset$

$$Sset = \{S \cdot 0 \leq k \wedge S = S_x^{l^t} + kS_x^{i^t} \wedge S \leq S_x^{u^t}\}$$

1.3 Plots

There are two kinds of plots displayed. One for fitness and one for computational time. In both cases x-axis plots size of sample S, and quantity on y-axis. Y coordinates are drawn with error bars. The error bars are the *standard error of the mean* (Hogg & Tanis, 1988), p.411, for the iterations run. So for given x and t, y-axis errorbars are equal to e_x^t

$$e_x^t = \frac{s_x^t}{\sqrt{I_x^t}}$$

where s_x^t is the *sample standard deviation* (Hogg & Tanis, 1988), p.32, and the way to compute in the two cases is detailed in the next two subsections.

1.3.1 Rms error

To estimate fitness of $\bar{p}$ to p we use *root mean square error* (Rms, $R_x^{t_i}$) for the means of squares for each $\bar{p}_k$ to p_k. Let $\bar{p}_k$ be a shorthand for $\bar{p}_{\langle x,t,i,k\rangle}$, the k learned parameter when experiment x set t and iteration i are obvious in some context.

$$R_x^{t_i} = \sqrt{\left(\frac{\sum_{j=1}^{N}(p_j - \bar{p}_j)^2}{N}\right)}$$

For the complete set of iteration of t, the mean and the sample standard deviation are :

$$\bar{y}_x^t = \frac{\sum_{j=1}^{I_x^t} R_x^{t_j}}{I_x^t}$$

$$(s_x^t)^2 = \frac{(I_x^t \sum_{j=1}^{I_x^t}(R_x^{t_j})^2) - (\sum_{j=1}^{I_x^t} R_x^{t_j})^2}{I_x^t(I_x^t - 1)}$$

1.3.2 Runtimes

Runtimes are measured in milliseconds and the raw values are used, $T_x^{t_j}$. Thus,

$$\bar{y}_x^t = \frac{\sum_{j=1}^{I_x^t} T_x^{t_j}}{I_x^t}$$

$$(s_x^t)^2 = \frac{(I_x^t \sum_{j=1}^{I_x^t}(T_x^{t_j})^2) - (\sum_{j=1}^{I_x^t} T_x^{t_j})^2}{I_x^t(I_x^t - 1)}$$

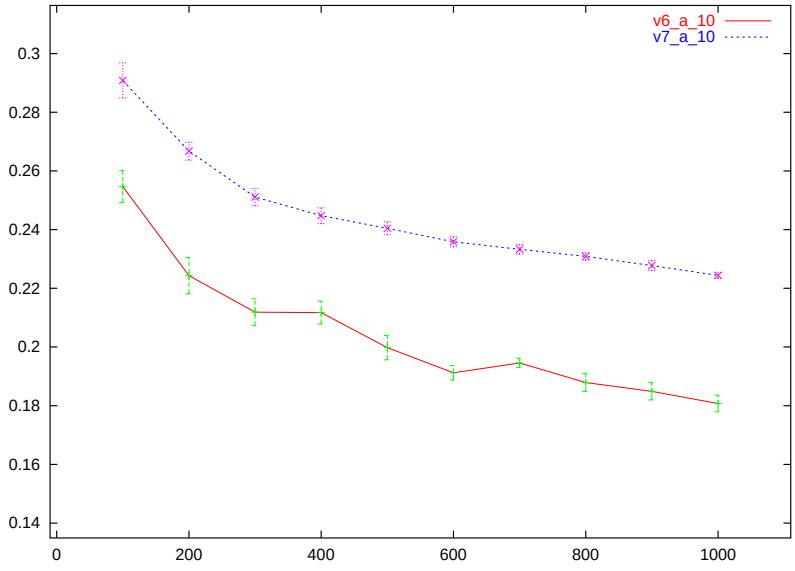
Chapter 2

Comparing v6 and v7

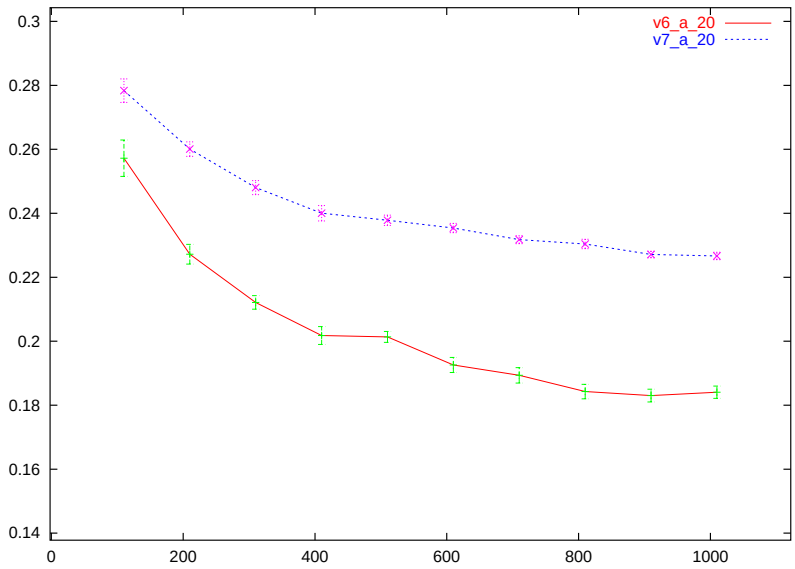
Each Section in this Chapter compares a single experiment run under version 6 and version 7. Our objective is to observe how the addition of the alternative path effects accuracy and rutime behaviour. The various plots are presented without text although some general analysis remarks can be found in Chapter 4.

2.1 E_a

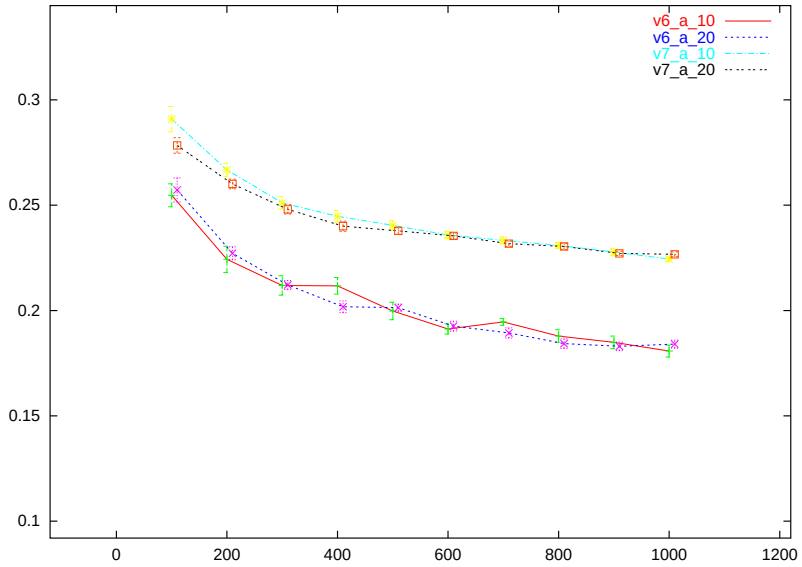
2.1.1 Rms



Plot 2.1: Rms, v6vs7 on 10 repetitions, of E_a.

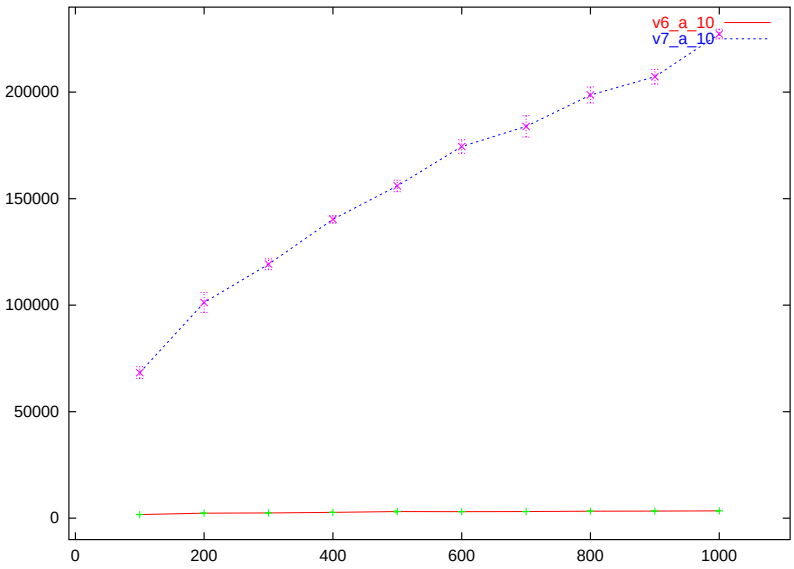


Plot 2.2: Rms, v6vs7 on 20 repetitions, of E_a.

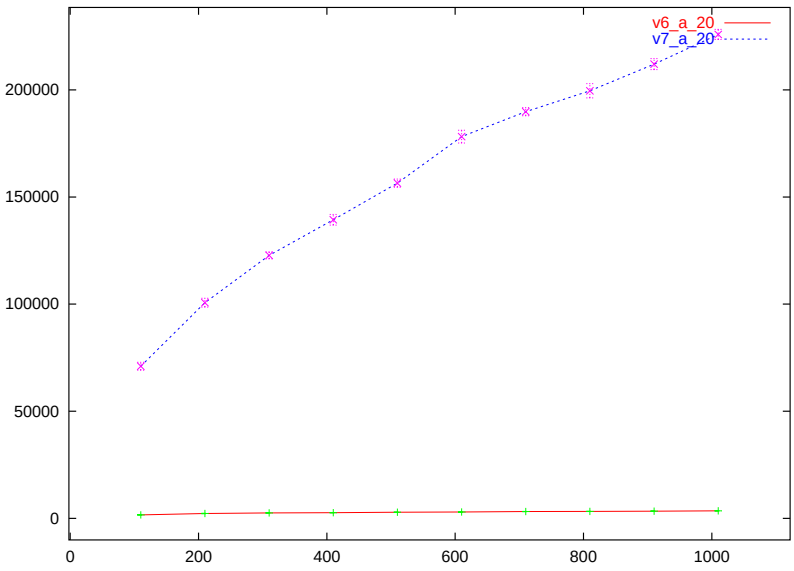


Plot 2.3: Rms, v6vs7 on 10 and 20 repetitions, of E_a.

2.1.2 Runtimes



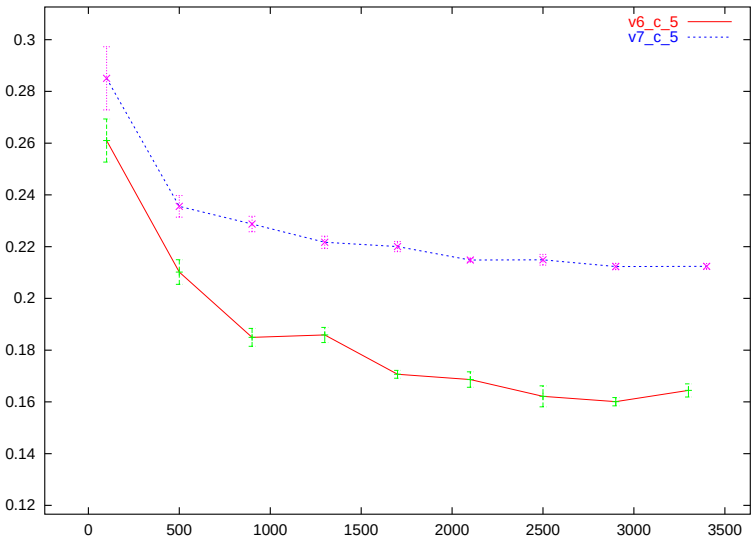
Plot 2.4: Runtimes, v6vs7 on 10 repetitions, of E_a.



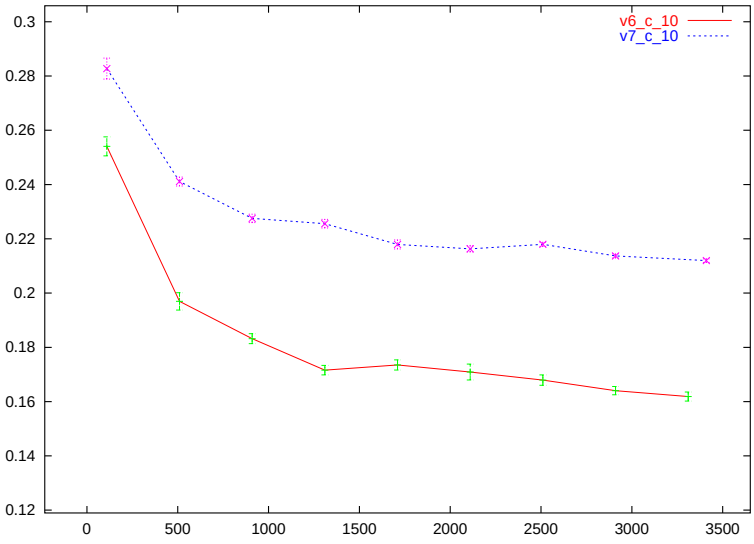
Plot 2.5: Runtimes, v6vs7 on 20 repetitions, of E_a.

2.2 E_c

2.2.1 Rms

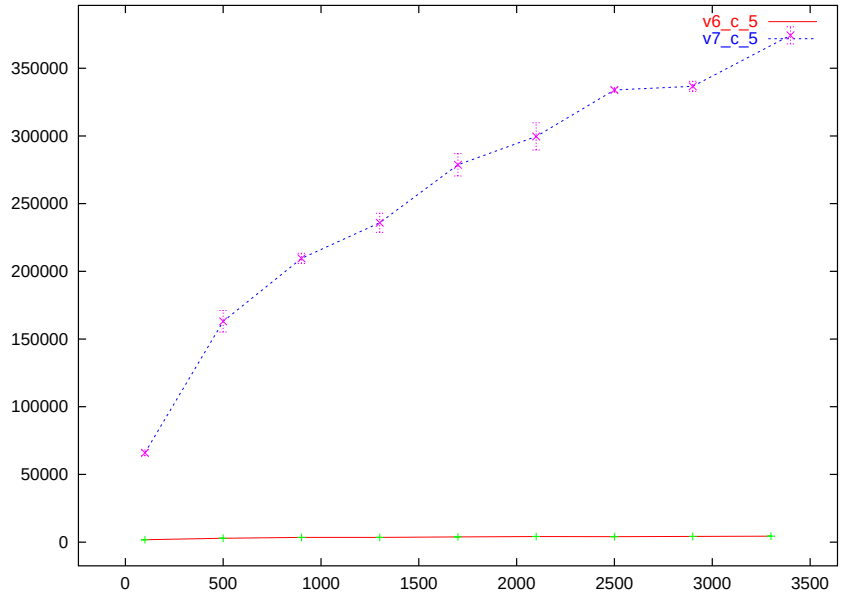


Plot 2.6: Rms, v6vs7 on 5 repetitions, of E_c.

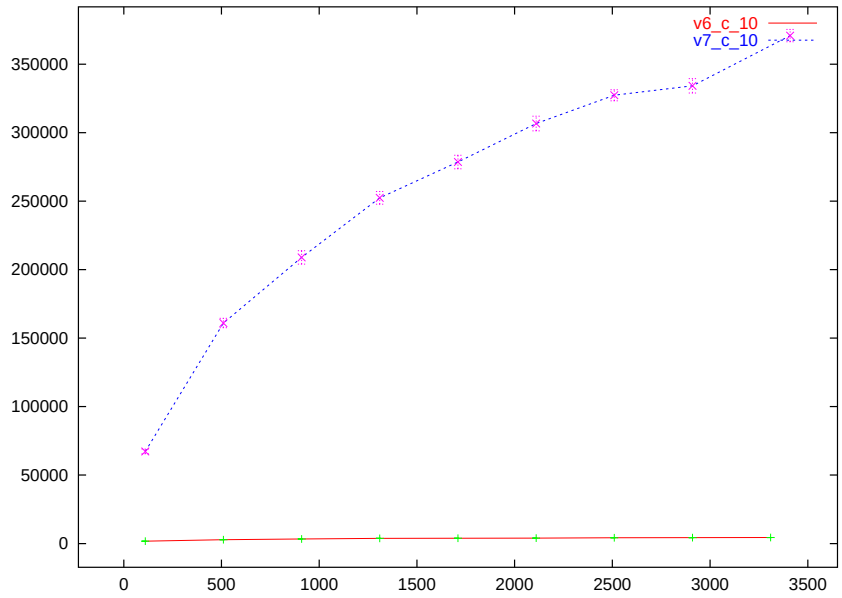


Plot 2.7: Rms, v6vs7 on 10 repetitions, of E_c.

2.2.2 Runtimes



Plot 2.8: Runtimes, v6vs7 on 5 repetitions, of E_c.



Plot 2.9: Runtimes, v6vs7 on 10 repetitions, of E_c.

Chapter 3

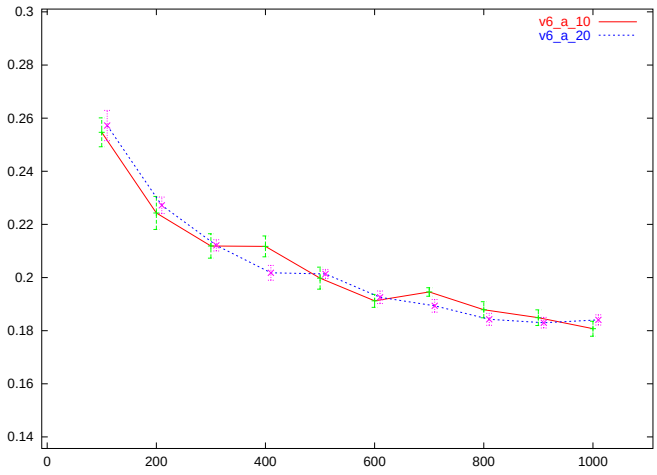
Comparing Repetitions

This Chapter investigates the influence of repeated runs and pseudorandomess to Rms and runtimes. Each of the two sections deals with a single Slp (v6 and v7). Within each section Rms and runtimes are compared for two runs that used different number of repetitions.

3.1 v6

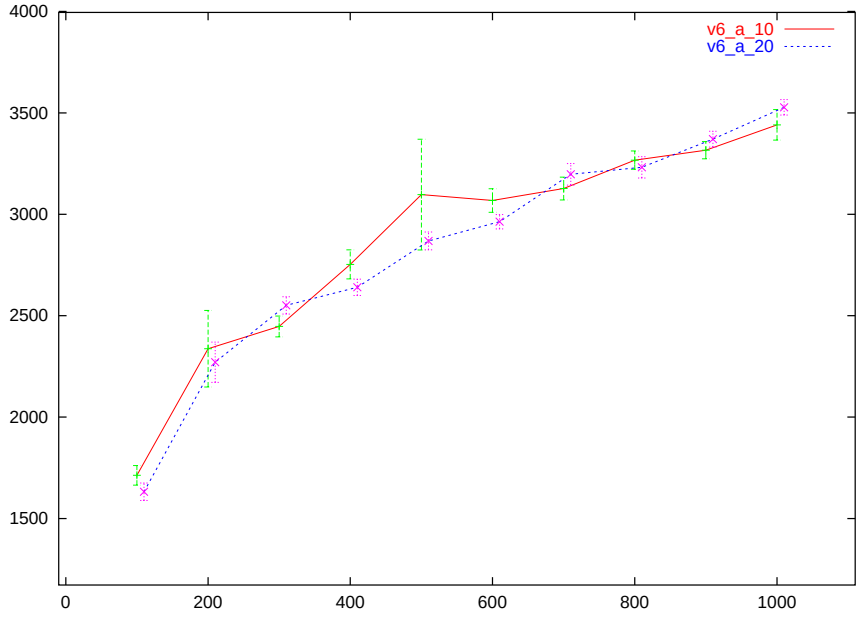
3.1.1 E_a

Rms



Plot 3.1: Rms, v6, 10 and 20 repetitions, of E_a.

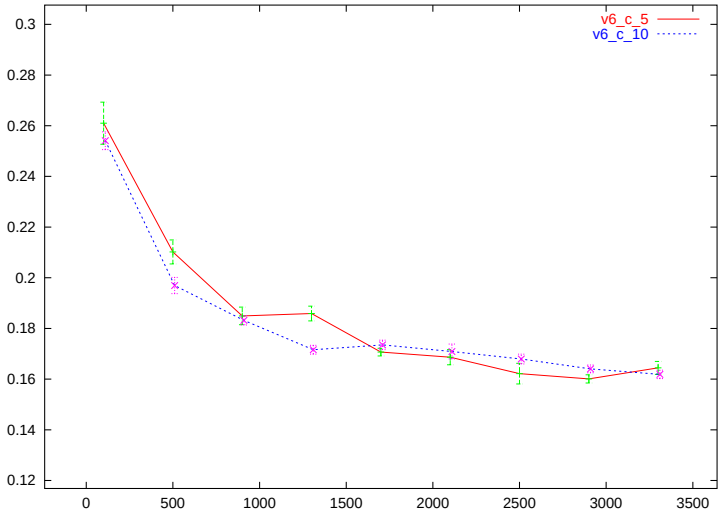
Runtimes



Plot 3.2: Runtimes, v6, 10 and 20 repetitions, of E_a.

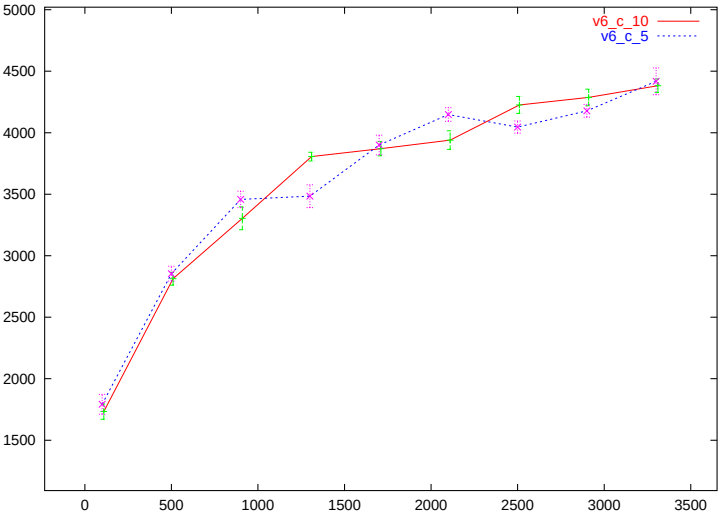
3.1.2 E_c

Rms



Plot 3.3: Rms, v6, 5 and 10 repetitions, of E_c.

Runtimes

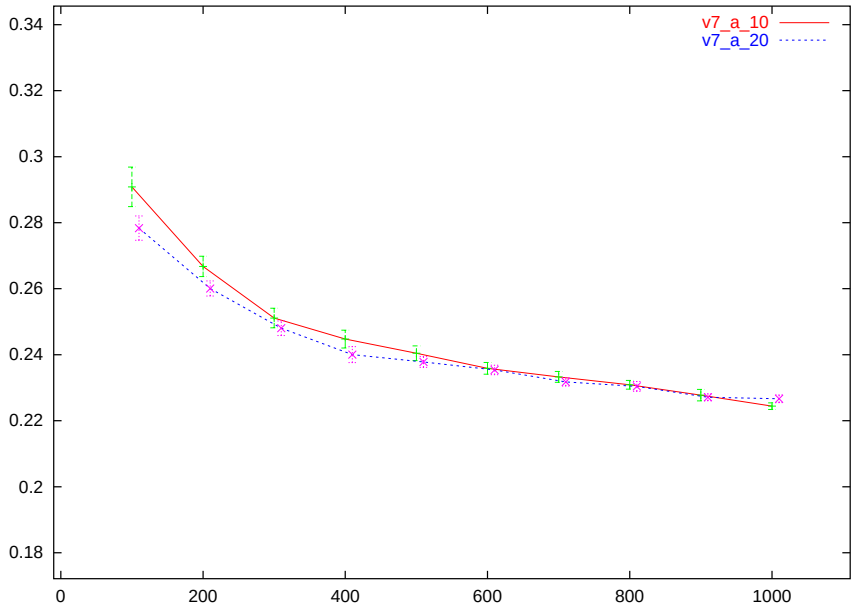


Plot 3.4: Runtimes, v6, 5 and 10 repetitions, of E_c.

3.2 v7

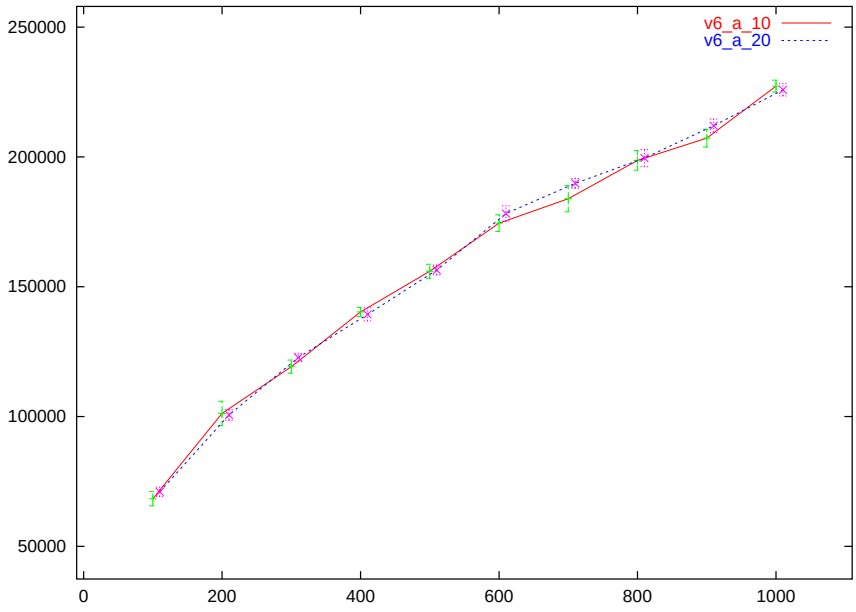
3.2.1 E_a

Rms



Plot 3.5: Rms, v7, 10 and 20 repetitions, of E_a.

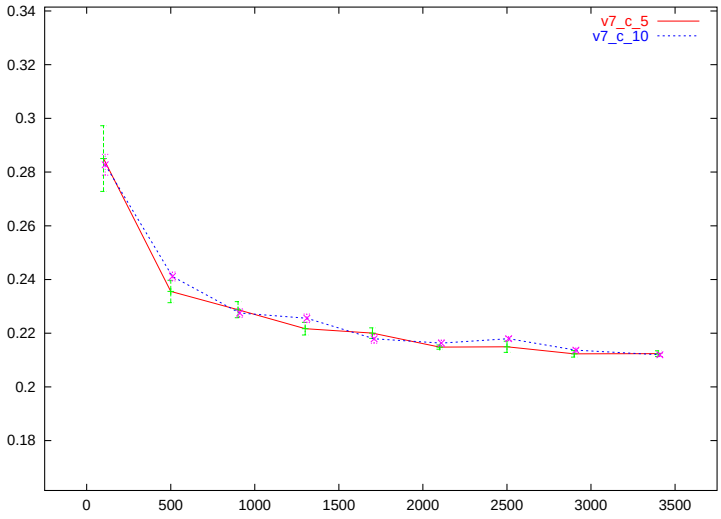
Runtime



Plot 3.6: Runtimes, v7, 10 and 20 repetitions, of E_a.

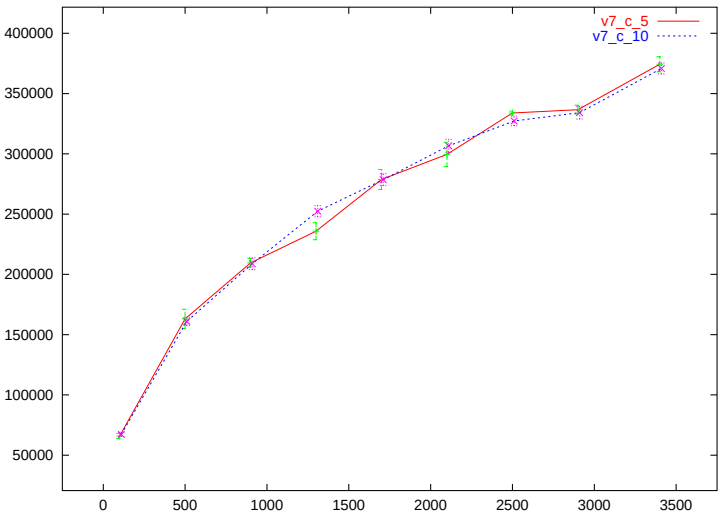
3.2.2 E_c

Rms



Plot 3.7: Rms, v7, 5 and 10 repetitions, of E_c.

Runtime



Plot 3.8: Runtimes, v7, 5 and 10 repetitions, of E_c.

Chapter 4

Remarks

Although we do not present formal results here, still, the presented Rms Plots suggest that the algorithm, and its implementation, cope well with precision. Indeed, a few hundred examples are enough to give a fairly accurate estimate of the parameters. On the other hand more accurate learning requires fairly substantial datasets.

One unexpected feature of the comparative Rms Plots, is that the introduction of the alternative path in v7, leads to a marginal deterioration of accuracy. If the alternative path was introduced, towards the end of the chain then this feature will be easier to explain. But, since this is not the case we are currently unable to provide an explanation. We believe that this feature warrants further investigation.

Finally the presented experiments have being useful in identifying the more serious deterioration in performance of the algorithm and/or its implementation. Further work is needed to, firstly, identify and secondly, suggest means of addressing the performance deterioration that stems from the presence of alternative paths. However, one need to note that Slps and Fam are capable of handling disjunctive information whereas comparable representations such as Bayesian Networks are not.

Appendix A

Version 6

```
:- ensure_loaded( library(ordsets) ).
:- ensure_loaded( library(lists) ).
:- ensure_loaded( ord_subset_difference ).

enzyme( '4.2.1.1' , rea_4_2_1_1_a, [c00631], [c00074] ).
enzyme( '4.1.2.15', rea_4_1_2_15, [c00074,c00279], [c00009,c04691] ).
enzyme( '4.6.1.3' , rea_4_6_1_3 , [c04691], [c00009,c00944] ).
enzyme( '4.2.1.10', rea_4_2_1_10, [c00944], [c02637] ).
enzyme( 'x'       , rea_x       , [c02637], [c02652] ).
enzyme( '1.1.1.25', rea_1_1_1_25, [c00005,c02652], [c00006,c00493] ).
enzyme( '2.7.1.71', rea_2_7_1_71, [c00002,c00493], [c00008,c03175] ).
enzyme( '2.5.1.19', rea_2_5_1_19, [c00074,c03175], [c00009,c01269] ).
enzyme( '4.6.1.4' , rea_4_6_1_4 , [c01269], [c00009,c00251] ).
enzyme( '5.4.99.5', rea_5_4_99_5, [c00251], [c00254] ).
enzyme( '1.3.1.13', rea_1_3_1_13, [c00006,c00254], [c00005,c01179] ).
enzyme( '2.6.1.7' , rea_2_6_1_7_a, [c00025,c01179], [c00026,c00082] ).
enzyme( '4.2.1.51', rea_4_2_1_51, [c00254], [c00166] ).
enzyme( '2.6.1.7' , rea_2_6_1_7_b , [c00025,c00166], [c00026,c00079] ).
enzyme( '4.1.3.27', rea_4_1_3_27, [c00014,c00251], [c00022,c00025,c00108] ).
enzyme( '2.4.2.18', rea_2_4_2_18, [c00108,c00119], [c00013,c04302] ).
enzyme( '5.3.1.24', rea_5_3_1_24, [c04302], [c01302] ).
enzyme( '4.1.1.48', rea_4_1_1_48, [c01302], [c03506] ).
enzyme( '4.2.1.20', rea_4_2_1_20_a, [c03506], [c00463,c00661] ).
enzyme( '4.2.1.20', rea_4_2_1_20_b, [c00065,c03506], [c00078,c00661] ).
enzyme( '4.2.1.20', rea_4_2_1_20_c, [c00065,c00463], [c00078] ).

0.95 :: rea_4_2_1_1_a( yes, yes ).
0.05 :: rea_4_2_1_1_a( yes, no ).

% 0.60 :: rea_4_2_1_1_b( yes, yes ).
% 0.40 :: rea_4_2_1_1_b( yes, no ).
```

```
0.75 :: rea_4_1_2_15( yes, yes, yes, yes ).
0.25 :: rea_4_1_2_15( yes, yes, no, no ).

0.85 :: rea_4_6_1_3( yes, yes, yes ).
0.15 :: rea_4_6_1_3( yes, no, no ).

0.90 :: rea_4_2_1_10( yes, yes ).
0.10 :: rea_4_2_1_10( yes, no ).

0.85 :: rea_x( yes, yes ).
0.15 :: rea_x( yes, no ).

0.80 :: rea_1_1_1_25( yes, yes, yes, yes ).
0.20 :: rea_1_1_1_25( yes, yes, no, no ).

0.95 :: rea_2_7_1_71( yes, yes, yes, yes ).
0.05 :: rea_2_7_1_71( yes, yes, no, no ).

0.85 :: rea_2_5_1_19( yes, yes, yes, yes ).
0.15 :: rea_2_5_1_19( yes, yes, no, no ).

0.90 :: rea_4_6_1_4( yes, yes, yes ).
0.10 :: rea_4_6_1_4( yes, no, no ).

0.75 :: rea_5_4_99_5( yes, yes ).
0.25 :: rea_5_4_99_5( yes, no ).

0.85 :: rea_1_3_1_13( yes, yes, yes, yes ).
0.15 :: rea_1_3_1_13( yes, yes, no, no ).

0.80 :: rea_2_6_1_7_a( yes, yes, yes, yes ).
0.20 :: rea_2_6_1_7_a( yes, yes, no, no ).

0.85 :: rea_4_2_1_51( yes, yes ).
0.15 :: rea_4_2_1_51( yes, no ).

0.95 :: rea_2_6_1_7_b( yes, yes, yes, yes ).
0.05 :: rea_2_6_1_7_b( yes, yes, no, no ).

0.90 :: rea_4_1_3_27( yes, yes, yes, yes, yes ).
0.10 :: rea_4_1_3_27( yes, yes, no, no, no ).

0.85 :: rea_2_4_2_18( yes, yes, yes, yes ).
0.15 :: rea_2_4_2_18( yes, yes, no, no ).
```

```
0.80 :: rea_5_3_1_24( yes, yes ).
0.20 :: rea_5_3_1_24( yes, no ).

0.80 :: rea_4_1_1_48( yes, yes ).
0.20 :: rea_4_1_1_48( yes, no ).

0.75 :: rea_4_2_1_20_a( yes, yes, yes ).
0.25 :: rea_4_2_1_20_a( yes, no, no ).

0.85 :: rea_4_2_1_20_b( yes, yes, yes, yes ).
0.15 :: rea_4_2_1_20_b( yes, yes, no, no ).

0.90 :: rea_4_2_1_20_c( yes, yes, yes ).
0.10 :: rea_4_2_1_20_c( yes, yes, no ).

can_produce( Metabolites, Products ) :-
        can_produce( Metabolites, [], Products ).

can_produce( Metabolites, Stalled, Products ) :-
        ( possible_reaction( Metabolites, Stalled, Reaction, Ins, Outs, RestM ) ->
                reaction_call( Reaction, Ins, Outs, Call ),
                sif( call( s-Call ),
                        can_produce( RestM, [Reaction|Stalled], Products )
                        ,
                        can_produce( Metabolites, [Reaction|Stalled], Products )
                )
                ;
                Products = Metabolites
        ).

possible_reaction( InMets, Stall, React, Ins, Outs, OutMets ) :-
        enzyme( _Enz, React, Ins, Outs ),
        \+ memberchk( React, Stall ),
        ord_subset_difference( Ins, InMets, _Diff ),
        ord_union( Outs, InMets, OutMets ).

reaction_call( Reaction, Ins, Outs, Call ) :-
        enzyme_arg_to_reaction_args( Ins, Yins ),
        enzyme_arg_to_reaction_args( Outs, Youts ),
        append( Yins, Youts, Args ),
        Call =.. [Reaction|Args].

enzyme_arg_to_reaction_args( [], [] ).
enzyme_arg_to_reaction_args( [_H|T], [yes|Targs] ) :-
        enzyme_arg_to_reaction_args( T, Targs ).
```

Appendix B

Version 7

```
:- ensure_loaded( library(ordsets) ).
:- ensure_loaded( library(lists) ).
:- ensure_loaded( ord_subset_difference ).

enzyme( '4.2.1.1' , rea_4_2_1_1_a, [c00631], [c00074] ).
% enzyme( '4.2.1.1' , rea_4_2_1_1_b, [c03356], [c00074] ).
enzyme( '4.1.2.15', rea_4_1_2_15, [c00074,c00279], [c00009,c04691] ).
enzyme( '4.6.1.3' , rea_4_6_1_3 , [c04691], [c00009,c00944] ).
enzyme( '4.2.1.10', rea_4_2_1_10, [c00944], [c02637] ).
enzyme( 'x'       , rea_x       , [c02637], [c02652] ).
enzyme( '1.1.1.25', rea_1_1_1_25, [c00005,c02652], [c00006,c00493] ).
enzyme( '2.7.1.71', rea_2_7_1_71, [c00002,c00493], [c00008,c03175] ).
enzyme( '2.5.1.19', rea_2_5_1_19, [c00074,c03175], [c00009,c01269] ).
enzyme( '4.6.1.4' , rea_4_6_1_4 , [c01269], [c00009,c00251] ).
enzyme( '5.4.99.5', rea_5_4_99_5, [c00251], [c00254] ).
enzyme( '1.3.1.13', rea_1_3_1_13, [c00006,c00254], [c00005,c01179] ).
enzyme( '2.6.1.7' , rea_2_6_1_7_a, [c00025,c01179], [c00026,c00082] ).
enzyme( '4.2.1.51', rea_4_2_1_51, [c00254], [c00166] ).
enzyme( '2.6.1.7' , rea_2_6_1_7_b , [c00025,c00166], [c00026,c00079] ).
enzyme( '4.1.3.27', rea_4_1_3_27, [c00014,c00251], [c00022,c00025,c00108] ).
enzyme( '2.4.2.18', rea_2_4_2_18, [c00108,c00119], [c00013,c04302] ).
enzyme( '5.3.1.24', rea_5_3_1_24, [c04302], [c01302] ).
enzyme( '4.1.1.48', rea_4_1_1_48, [c01302], [c03506] ).
enzyme( '4.2.1.20', rea_4_2_1_20_a, [c03506], [c00463,c00661] ).
enzyme( '4.2.1.20', rea_4_2_1_20_b, [c00065,c03506], [c00078,c00661] ).
enzyme( '4.2.1.20', rea_4_2_1_20_c, [c00065,c00463], [c00078] ).

% the alternative path
enzyme( '2.7.1.40', rea_2_7_1_40 , [c00074,c00008], [c00022,c00002] ).
% you can substitute 5 different values for c00008 to get different
% alternatives to c00002.
```

```
enzyme( '4.1.3.27', rea_4_1_3_27a, [c00108,c00022,c00025], [c00251,c00064,c00014] ).
enzyme( '4.6.1.4', rea_4_6_1_4b, [c00251,c00009], [c01269] ).
enzyme( '2.5.1.19', rea_2_5_1_19b, [c01269,c00009], [c03175,c00074] ).
enzyme( '2.7.1.71', rea_2_7_1_71b, [c00008,c03175], [c00002,c00493] ).

0.95 :: rea_2_7_1_40( yes, yes, yes, yes ).
0.05 :: rea_2_7_1_40( yes, yes,  no,  no ).

0.90 :: rea_4_1_3_27a( yes, yes, yes, yes, yes, yes ).
0.10 :: rea_4_1_3_27a( yes, yes, yes, no, no, no ).

0.80 :: rea_4_6_1_4b( yes, yes, yes ).
0.20 :: rea_4_6_1_4b( yes, yes, no ).

0.85 :: rea_2_5_1_19b( yes, yes, yes, yes ).
0.15 :: rea_2_5_1_19b( yes, yes, no, no ).

0.80 :: rea_2_7_1_71b( yes, yes, yes, yes ).
0.20 :: rea_2_7_1_71b( yes, yes,  no,  no ).

0.95 :: rea_4_2_1_1_a( yes, yes ).
0.05 :: rea_4_2_1_1_a( yes, no ).

% 0.60 :: rea_4_2_1_1_b( yes, yes ).
% 0.40 :: rea_4_2_1_1_b( yes, no ).

0.75 :: rea_4_1_2_15( yes, yes, yes, yes ).
0.25 :: rea_4_1_2_15( yes, yes, no, no ).

0.85 :: rea_4_6_1_3( yes, yes, yes ).
0.15 :: rea_4_6_1_3( yes, no, no ).

0.90 :: rea_4_2_1_10( yes, yes ).
0.10 :: rea_4_2_1_10( yes, no ).

0.85 :: rea_x( yes, yes ).
0.15 :: rea_x( yes, no ).

0.80 :: rea_1_1_1_25( yes, yes, yes, yes ).
0.20 :: rea_1_1_1_25( yes, yes, no, no ).

0.95 :: rea_2_7_1_71( yes, yes, yes, yes ).
0.05 :: rea_2_7_1_71( yes, yes, no, no ).

0.85 :: rea_2_5_1_19( yes, yes, yes, yes ).
0.15 :: rea_2_5_1_19( yes, yes, no, no ).
```

```
0.90 :: rea_4_6_1_4( yes, yes, yes ).
0.10 :: rea_4_6_1_4( yes, no, no ).

0.75 :: rea_5_4_99_5( yes, yes ).
0.25 :: rea_5_4_99_5( yes, no ).

0.85 :: rea_1_3_1_13( yes, yes, yes, yes ).
0.15 :: rea_1_3_1_13( yes, yes, no, no ).

0.80 :: rea_2_6_1_7_a( yes, yes, yes, yes ).
0.20 :: rea_2_6_1_7_a( yes, yes, no, no ).

0.85 :: rea_4_2_1_51( yes, yes ).
0.15 :: rea_4_2_1_51( yes, no ).

0.95 :: rea_2_6_1_7_b( yes, yes, yes, yes ).
0.05 :: rea_2_6_1_7_b( yes, yes, no, no ).

0.90 :: rea_4_1_3_27( yes, yes, yes, yes, yes ).
0.10 :: rea_4_1_3_27( yes, yes, no, no, no ).

0.85 :: rea_2_4_2_18( yes, yes, yes, yes ).
0.15 :: rea_2_4_2_18( yes, yes, no, no ).

0.80 :: rea_5_3_1_24( yes, yes ).
0.20 :: rea_5_3_1_24( yes, no ).

0.80 :: rea_4_1_1_48( yes, yes ).
0.20 :: rea_4_1_1_48( yes, no ).

0.75 :: rea_4_2_1_20_a( yes, yes, yes ).
0.25 :: rea_4_2_1_20_a( yes, no, no ).

0.85 :: rea_4_2_1_20_b( yes, yes, yes, yes ).
0.15 :: rea_4_2_1_20_b( yes, yes, no, no ).

0.90 :: rea_4_2_1_20_c( yes, yes, yes ).
0.10 :: rea_4_2_1_20_c( yes, yes, no ).

% scall( can_produce([c00074,c00279], Products), Path,Succ,Prb ).
% sample( can_produce([c00074,c00279], Products), FlPath,Succ,Prb ).

can_produce( Metabolites, Products ) :-
        can_produce( Metabolites, [], Products ).
```

```
can_produce( Metabolites, Stalled, Products ) :-
        ( possible_reaction( Metabolites, Stalled, Reaction, Ins, Outs, RestM ) ->
                reaction_call( Reaction, Ins, Outs, Call ),
                sif( call( s-Call ),
                        can_produce( RestM, [Reaction|Stalled], Products )
                        ,
                        can_produce( Metabolites, [Reaction|Stalled], Products )
                )
                ;
                Products = Metabolites
        ).

possible_reaction( InMets, Stall, React, Ins, Outs, OutMets ) :-
        enzyme( _Enz, React, Ins, Outs ),
        \+ memberchk( React, Stall ),
        ord_subset_difference( Ins, InMets, _Diff ),
        ord_union( Outs, InMets, OutMets ).

reaction_call( Reaction, Ins, Outs, Call ) :-
        enzyme_arg_to_reaction_args( Ins, Yins ),
        enzyme_arg_to_reaction_args( Outs, Youts ),
        append( Yins, Youts, Args ),
        Call =.. [Reaction|Args].

enzyme_arg_to_reaction_args( [], [] ).
enzyme_arg_to_reaction_args( [_H|T], [yes|Targs] ) :-
        enzyme_arg_to_reaction_args( T, Targs ).
```

References

Angelopoulos, N., & Cussens, J. (2001). Pepl.. Available from: http://www.cs.york.ac.uk/~nicos/slps.

Bryant, C., Muggleton, S., Oliver, S., Kell, D., Reiser, P., & King, R. (2001). Combining inductive logic programming, active learning and robotics to discover the function of genes. *Electronic Transactions in Artificial Intelligence*, *6-B1*(012), 1–36.

Cussens, J. (2000). Stochastic logic programs: Sampling, inference and applications. In *Sixteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-2000)*, pp. 115–122 San Francisco, CA. Morgan Kaufmann.

Cussens, J. (2001). Parameter estimation in stochastic logic programs. *Machine Learning*, *44*(3), 245–271.

Hogg, R. V., & Tanis, E. A. (1988). *Probability and Statistical Inference* (3rd edition). Macmillan Publishing Co.

Muggleton, S. (1996). Stochastic logic programs. In de Raedt, L. (Ed.), *Advances in Inductive Logic Programming*, pp. 254–264. IOS Press.