BioHacker

Nada Amin
MIT
namin@mit.edu

Dan Wheeler
MIT
lowe@mit.edu

Jeremy Zucker
MIT
zucker@broad.mit.edu

1. INTRODUCTION

Despite a plethora of sequenced genomes, a paucity of genome-scale metabolic models have been published thus far. The bottleneck comes from the extensive manual effort is currently required to reconstruct the metabolic network. In order to be effective, a model of metabolism should be *coherent*, in the sense that the model should obey physico-chemical constraints, such as charge and mass balance. It should be *complete*, in the sense that all genes that code for metabolic enzymes should be part of the model. Finally, the model should be *consistent* with conditional gene essentiality experiments that measure growth/no-growth under various different nutrient media and gene knockouts.

Inspired by computer models of skill acquisition[10] and robot scientists that automate the scientific process[5], we have developed `BIOHACKER` as a network debugging tool to detect incoherence in the network, generate reasonable hypotheses for filling in gaps in incomplete networks, and predict sufficient nutrient sets that are consistent with conditional essentiality experiments.

2. NETWORK DEBUGGER

2.1 Overview

The user of the network debugger provides a model of the network and a set of experiments. For each experiment, the network debugger assesses whether the experiment is consistent with the model, explaining why (with a queryable proof) or why not (with queryable minimal fixes).

2.2 Demonstration

For purpose of demonstration, we will use the small network shown in Figure 1. The user code which specifies this network model is shown in Listing 1.

The set of experiments in Listing 2 tests our model. In order to test our system, we purposefully introduced some inconsistencies in our model with respect to the experiments:

- `reaction r1` has its reactants and products swapped.
- `reaction r2` has no enzymes catalyzing it.
- `reaction r4` shouldn't actually be part of our model.

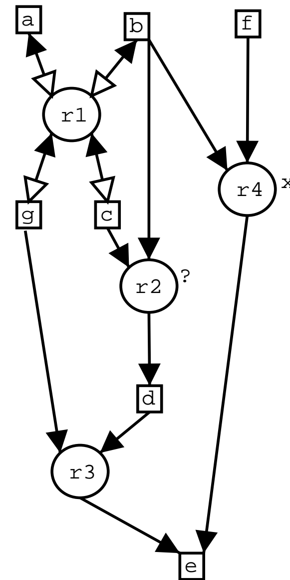


Figure 1: Example Metabolic Network

Listing 1: Network Model

```
(network-debugger demo
  :debugging t
  :abducting t
  :rules :extended-reactions)

(reaction r1
  :reactants (c g)
  :products (a b)
  :enzymes (e1))

(reaction r2
  :reactants (b c)
  :products (d))

(reaction r3
  :reactants (d g)
  :products (e)
  :enzymes (e3))

(reaction r4
```

6.945 2008 Cambridge, MA USA

```
  :reactants (b f)
  :products (e)
  :enzymes (e4))

(enzyme e1 g1)
(enzyme e3 g3 g3p)
(enzyme e4 g4)
```

Listing 2: Experiments

```
(experiment
 positive
 (a b d)
 :growth? t
 :essential-compounds (a e))

(experiment
 negative
 (a)
 :growth? nil
 :essential-compounds (a e))

(experiment
 false-positive
 (a b f)
 :growth? nil
 :essential-compounds (a e)
 :knock-outs (g1))

(experiment
 false-negative
 (a b)
 :growth? t
 :essential-compounds (a e))
```

We immediately get the following feedback when loading the set of experiments:

- Experiments positive and negative are consistent with the model.
- Experiment false-positive is not consistent with the model. Needs:
  - ((NOT (GENE-ON G4)))
- Experiment false-negative is not consistent with the model. Needs:
  - ((NUTRIENT E))
  - ((NUTRIENT D))
  - ((REACTION-ENABLED R2))
  - ((NUTRIENT F))

Notice that the network debugger suggested enabling reaction r2 and disabling reaction r4 to fix our model.

Once an experiment is consistent with the model, the network debugger can explain why by providing a proof and allowing the user to query this proof. For example, after loading the experiment positive, the user can type (explain 'experiment-consistent) to get a proof in the syle of Suppes [9]. The user can query the facts that play a role in the proof using all-antecedents. For example, (all-antecedents 'experiment-consistent '((reaction-enabled ?r) (reaction-reversible ?r))) returns ((REACTION-ENABLED R3) (REACTION-ENABLED R1) (REACTION-REVERSIBLE R1)). As a shortcut, it is possible to list reactions that had to explicitly be assumed reversible for the proof: (explicit-reversibility) returns (R1). Similarly, (explicit-gene-expression) returns which genes had to explicitly be assumed on for the proof.

2.3 Specification

We describe the user language to define the network model and experiments.

The network model consists of definitions of pathways, reactions, and enzymes.

An enzyme is defined by (enzyme <name> . <list of genes (conjunctive)>. The list of genes may contain :UNKNOWN to indicate that the enzyme has some unknown gene contributing to its formation.

A pathway or reaction is defined by (pathway/reaction <name> ...) where ... can contain any of the following keys:

:reactants <list of reactants>

:products <list of products>

:reversible? <t / nil / :UNKNOWN> (By default, :UNKNOWN for reactions and nil for pathways.)

:enzymes <list of enzymes (disjunctive for reactions, unspecified logic for pathways)> Note that

- nil for a reaction means that NO enzymes catalyze it, i.e. the reaction will never be fired but might play a role in abduction.
- :SPONTANEOUS, instead of a list, means that no enzymes are needed.
- (:UNKNOWN) means that some unknown enzyme is needed. :UNKNOWN can be part of any enzyme list.

:reactions <list of all reactions in the pathway> (pathways only)

:proper-products <list of the actual end products of a pathway> (pathways only) (This list should be a subset of products, and is the same as products by default.)

An experiment is defined by (experiment <name> <nutrient-list> :growth? <T or nil> :essential-compounds <list of compounds that must be produecd in order to achieve growth (conjunctive)> ...) where ... can contain any of the following:

knock-outs <list of genes asserted to be off>

knock-ins <list of genes asserted to be on>

toxins <list of compounds that if produced cause no-growth experimental outcomes (disjunctive)>

bootstrap-compounds <list of compounds that are asserted to exist, but not nutrients>

2.4 Implementation

We implemented the network debugger on top of a logic-based truth maintenance system (LTMS) [2]. Figure 2 gives an overview of the system.

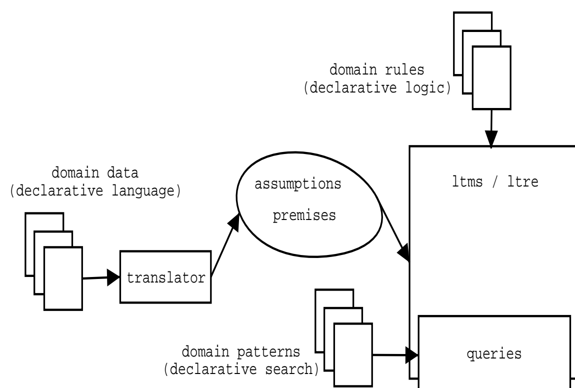


Figure 2: Overview of the Network Debugger System

Crucially, all our domain-specific knowledge is specified declaratively. The domain data is given by the user, in the domain-specific language outlined in 2.3. The domain rules are specified using the rule system of the LTMS engine. Finally, we augmented the the query facility of the LTMS. The query facility is still general. We apply it to our domain by restricting the queries with patterns.

2.4.1 Domain Data

Using a straightforward set of macros, we translate the domain data into assumptions and premises of our LTMS. For example, the macros for `enzyme` and `experiment` are shown in Listing 3 and Listing 4. The network open / closed distinction is explained in 2.4.2.

Listing 3: Enzyme Macro

```
(defmacro enzyme (name &rest genes)
  `(ensure-network-open enzyme
    (assert! '(enzyme ,name ,@genes)
             :NETWORK)
    (debugging-nd
     "~%Adding enzyme ~A." ',name)))
```

Listing 4: Experiment Macro

```
(defmacro experiment (name
                      nutrients
                      &key
                      growth?
                      (knock-outs nil)
                      (knock-ins nil)
                      (toxins nil)
                      (bootstrap-compounds nil)
                      essential-compounds)
  `(ensure-network-closed
    experiment
    (assert! '(experiment
               ,name ,growth?
               ,nutrients ,essential-compounds
               ,bootstrap-compounds ,toxins
               ,knock-ins ,knock-outs)
             :EXPERIMENTS)
    (debugging-or-logging-nd
     "~%Adding experiment ~A" ',name)
    (run-rules-logging)
    (investigate-experiment ',name)))
```

2.4.2 Domain Rules

The domain rules describe how to derive more facts from the premises and assumptions. We can easily change the domain rules independently of the rest of the system. For example, we have different sets of rules, depending on whether the network debugger operates on pathways or reactions.

Listing 5 shows an excerpt of the domain rules. It simply expresses the logic that if a reaction is enabled and all the reactants are present, then the reaction is fired and all the products are present.

Listing 5: Excerpt of Domain Rules

```
(rule ((:INTERN (reaction
                 ?reaction
                 ?reactants ?products
                 ?reversible? ?enzymes)))
  ...
  (assert!
   `(:IMPLIES
     (:AND (reaction-enabled ,?reaction)
      ,@(list-of 'compound-present ?reactants))
    (:AND (reaction-fired ,?reaction)
      ,@(list-of 'compound-present ?products)))
   :REACTION-FIRED)
  ...
)
```

In addition to deriving positive facts, e.g. a certain compound is present, we would like to be able to derive negative facts, e.g. a certain compound is not present. However, in order to derive that a compound is not present, we need to assume that the network is closed, i.e. that no additional reactions are going to be declared. Listing 6 shows the domain rule that makes use of the `network-closed` premise.

Listing 6: Domain rule for deriving absence of compound

```
(rule ((:TRUE network-closed))
      (rule ((:INTERN (compound ?compound)))
(let*
 ((product-facts
   (fetch `(product ,?compound ?reaction)))
  (reactant-facts
   (fetch `(reactant ,?compound ?reaction)))
  (reaction-fired-facts
   (mapcar
    #'(lambda (fact)
      `(reaction-fired ,(caddr fact)))
    product-facts))
  (reverse-reaction-fired-facts
   (mapcar
    #'(lambda (fact)
      `(reverse-reaction-fired ,(caddr fact)))
    reactant-facts)))
  (assert!
   `(:IMPLIES
     (:AND
      (:NOT (nutrient ,?compound))
      (:NOT
       (:OR
```

```
        ,@reaction-fired-facts
        ,@reverse-reaction-fired-facts)))
      (:NOT (compound-present ,?compound)))
  :NETWORK-CLOSED))))
```

2.4.3 Queries

We extended the LTMS query facility with two procedures: `needs` and `all-antecedents`.

`all-antecedents` is straightforward. It simply makes a list of all facts that support a fact or recursively support any antecedent of that fact.

`needs` allows our system to perform abduction. It answers the question: which unknown facts, if true, would be sufficient for a given unknown fact to become true?

Both these procedures take an optional list of patterns, which limits the form of facts that can be returned.

3. RESULTS

To test the ability of the `NETWORK DEBUGGER` to debug real data, we exported the EcoCyc[4][8] representation of the *E. coli* metabolic network into the `NETWORK DEBUGGER` domain data format, generating 929 `reaction`s, 836 `enzyme`s, and 251 `pathway` premises.

To ensure that our model would grow on at least one nutrient media, we generated an *in silico* rich medium set by taking the union of all proper reactants of each pathway in the model, as shown in Listing 7

Listing 7: *E. coli* rich media

```
(defun make-rich-media (pwy-list filter-p)
  `(experiment
    growth
    (nutrients
     ,@(remove-duplicates
        (loop for pwy in pwy-list
           when (funcall filter-p pwy)
           append (multiple-value-bind
                         (all-reactants
                          proper-reactants
                          all-products
                          proper-products)
                       (substrates-of-pathway pwy)
                     proper-reactants)))) (OFF)))
```

We then selected as our essential compounds the full set of amino acids, nucleic acids, cytoplasmic membrane components, outer membrane components, and cell wall components as shown in Listing 8

Listing 8: *E. coli* essential compounds

```
(setq *amino-acids*
  '(L-ALPHA-ALANINE ARG ASN L-ASPARTATE CYS GLN
    GLT GLY HIS ILE LEU LYS MET PHE PRO SER THR
    TRP TYR VAL))

(setq *dna-and-rna*
  '(DATP TTP DGTP DCTP ATP UTP GTP CTP))

(setq *cytoplasmic-membrane*
  '(L-1-PHOSPHATIDYL-ETHANOLAMINE CARDIOLIPIN
    L-1-PHOSPHATIDYL-GLYCEROL))

(setq *outer-membrane* '(C6))
(setq *cell-wall*
  '(BISOHMYR-GLC ADP-L-GLYCERO-D-MANNO-HEPTOSE
    KDO UDP-GLUCOSE UDP-GALACTOSE DTDP-RHAMNOSE
    GDP-MANNOSE N-ACETYL-D-GLUCOSAMINE))
```

Surprisingly, even with this rich media, we were still unable to produce compound `C6` (lipid intermediate II)

By manual checking, we discovered that the reason for this is that an upstream reaction, `UDPNACETYLMURAMATEDEHYDROG-RXN` , in the Peptidoglycan biosynthesis pathway was reversed, making the entire chain of downstream metabolites unproduceable.

Once this bug was discovered, we added a new rule to our set of strategies so that if a metabolite is unproducable, the system tries to reverse reactions with unknown reversibilities to see if that will solve the problem.

Interestingly, it has been shown that for many metabolic reconstructions, that "the dominant flow restoration mechanism is directionality reversals of existing reactions in the respective models"[6]

Once we debugged this problem, we wished to reduce the rich medium to a nutrient set that was still capable of producing growth. By querying the `explain` mechanism of the underlying TMS, we were able to make the following sufficient nutrient set prediciton as shown in Listing 9.

Listing 9: *E. coli* predicted sufficient nutrient set

```
(GLN ASN THR LEU VAL ILE TRP PHE TYR CYS MET
     LYS GLY ARG HIS PRO UDP-GALACTOSE
     UDP-GLUCOSE RIBULOSE-5P
     ADP-L-GLYCERO-D-MANNO-HEPTOSE GLC-1-P
     TTP FRUCTOSE-6P 3-OHMYRISTOYL-ACP GLT
     NADPH PHOSPHO-ENOL-PYRUVATE
     MESO-DIAMINOPIMELATE L-ALPHA-ALANINE
     D-ALANINE UDP-N-ACETYL-D-GLUCOSAMINE
     UNDECAPRENYL-P GTP GLYCEROL-3P SER
     CDPDIACYLGLYCEROL METHYLENE-THF UTP
     CTP UDP |Red-Glutaredoxins|
     |Red-Thioredoxin| |Reduced-flavodoxins|
     CDP ATP WATER
     DIACETYLCHITOBIOSE-6-PHOSPHATE)
```

This sufficient nutrient set allowed us to focus on reactions and pathways that could produce these compounds starting with experimentally determined nutrient sets, such as Middlebrook growth medium L9, as described in [3].

Finally, we then converted 13750 conditional gene essentiality experiments performed by Covert et al[1] and for each condition, we recorded whether the *E. coli* model's predictions of growth were consistent or inconsistent with the experimental results. as shown in Listing 10.

Listing 10: *E. coli* conditional essentiality experiments

```
(experiment  in_vivo_EG11074_ala-D_nh4
            ( CARBON-DIOXIDE PROTON WATER
                             AMMONIUM
                             SULFATE
                             D-ALANINE |Pi|
                             OXYGEN-MOLECULE )
   :growth?  T
   :essential-compounds
   ( L-ALPHA-ALANINE ARG ASN L-ASPARTATE CYS
     GLN GLT GLY HIS ILE LEU LYS MET PHE PRO
     SER THR TRP TYR VAL DATP TTP DGTP DCTP
     ATP UTP GTP CTP
     L-1-PHOSPHATIDYL-ETHANOLAMINE CARDIOLIPIN
     L-1-PHOSPHATIDYL-GLYCEROL C6 BISOHMYR-GLC
     ADP-L-GLYCERO-D-MANNO-HEPTOSE KDO
     UDP-GLUCOSE  UDP-GALACTOSE DTDP-RHAMNOSE
     GDP-MANNOSE N-ACETYL-D-GLUCOSAMINE )

   :knock-outs  ( EG11074 )
   :knock-ins Nil
   :toxins Nil
   :bootstrap-compounds Nil)
```

We found that most of the model predictions were false negatives due to the existence of uninstantiated generic reactions. Inspired by the symbolic computational approach to infer novel biochemical knowledge described in [7], we decided to implement the `METABOLIZER` portion of our project.

4. METABOLIZER

Consider alcohol dehydrogenase, whereby an -OH group, connected to a chemical substructure R of arbitrary complexity, loses its Hydrogen and becomes an =O group. The first line of the database shows how this general reaction is currently stored in the pathway tools database:

Pattern Name:	alcohol	→	aldehyde
Graph Pattern:	R−OH	→	R=O
Instance Pattern:	C−C−OH	→	C−C=O
Instance Name:	ethenol	→	acetaldehyde

The second line is the same reaction at a structural level. An instance of this general reaction – ethanol -> acetaldehyde – is shown on the third and fourth lines.

Clearly, to be able to recognize the pattern/instance relationship between these two reactions, knowledge of the chemicals' structural representations is necessary; there is no deducible relationship between the atomic symbols 'alcohol and 'ethanol. In our current approach we therefore have no means of representing this general-level chemical knowledge. Our database is often bloated with repetitive instances of a small set of patterns. Worse, it is incomplete: in the case of alcohol dehydrogenase, for example, there is no way that the database has populated a corresponding aldehyde for every single chemical entry that has an -OH group.

An extension to our project, one that we made a minor dent on, would be to move our database from a reaction-of-names to a reaction-of-structures representation. Such a move would naturally allow such generic chemical knowledge to be encoded. The idea is to create a dynamic database that expands as reactions are explored throughout a search, through the application of appropriate generic reactions to the current chemicals at hand. The generic reactions, applied to specific reactants, produce specific products that can be added to the database.

The set-up we propose is to use graph pattern matching to figure out when a generic reaction applies, and to use graph rewriting to derive the products that result. Two procedures, MATCH and REWRITE, would provide this behavior.

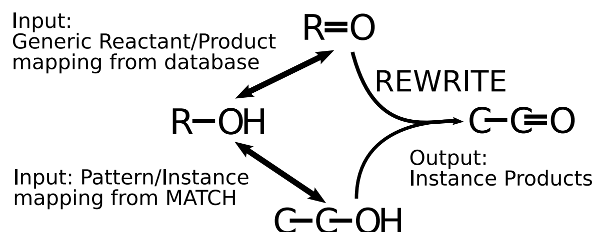
We started attempting to write MATCH using combinator-style syntactic pattern matching. We chose to represent chemical structure graphs using adjacency lists. CO_2, for example, looks like:

((C (2 2) (3 2)) (O (1 2)) (O (1 2)))

Where the C has index 1, and the Os have index 2 and 3 respectively. The C entry indicates that it is bonded to atom 2, that it is a double-bond, and likewise for atom 3. The remaining entries redundantly reflect the same connectivity. We thought about using canonicalization – alphabetical atom ordering, with increasing-index order for the bond lists of each entry. Unfortunately, a single chemical can still have several different adjacency lists under this canonicalization (for example C=0=C=0=C), so syntactic pattern matching would require combinatorially many patterns to work correctly.

It turns out several hours in the lab saved us 30 minutes in the library. Looking through Pathway Tools, we found an ancient subgraph matcher that could do precisely what we wanted. It works by aligning two atoms in the two input patterns, seeing if their bonds are consistent, and recursing on the atoms at the end of each bond, backtracking when necessary. We wrote a quick wrapper to provide the interface we desired and to hide hairiness of the underlying code.

The next step, one we may pursue in the future, is the REWRITE procedure. The design we had in mind is as follows:



In addition to the specific reactants, we have two pieces of information to work with as input to REWRITE: our generic reaction, represented as an atom mapping from generic reactants to generic products, and our MATCH to the specific reactants, also represented as a mapping. The REWRITE

procedure would then combine these two mappings to figure out how to rewrite the specific reactants into specific products. We may be better served thinking about how to create a better language for describing the mappings, rather than using atom number mappings. We haven't yet attempted an implementation.

5. CONCLUSION

By representing our domain knowledge declaratively, the underlying Truth Maintenance System can effectively reason about the behavior of the metabolic network. Furthermore, as additional knowledge about how to debug metabolic networks is gained, we expect that only small changes in the code will be necessary to accommodate those changes. We plan to continue developing this system with the eventual goal of releasing the software to the Pathway-tools community to aid in the rapid reconstruction of metabolic networks.

6. AVAILABILITY

`BIOHACKER` is released under the GNU General Public License v3. The latest source code can be found in the SVN repository at `http://biohacker.googlecode.com`.

7. REFERENCES

[1] M. W. Covert, E. M. Knight, J. L. Reed, M. J. Herrgard, and B. O. Palsson. Integrating high-throughput and computational data elucidates bacterial networks. *Nature*, 429(6987):92–96, May 2004.

[2] K. D. Forbus and J. D. Kleer. *Building Problem Solvers*. MIT Press, Cambridge, MA, USA, 1993.

[3] A. R. Joyce, J. L. Reed, A. White, R. Edwards, A. Osterman, T. Baba, H. Mori, S. A. Lesely, B. O. Palsson, and S. Agarwalla. Experimental and computational assessment of conditionally essential genes in escherichia coli. *J. Bacteriol.*, 188(23):8259–8271, December 2006.

[4] I. M. Keseler, J. Collado-Vides, S. Gama-Castro, J. Ingraham, S. Paley, I. T. Paulsen, M. Peralta-Gil, and P. D. Karp. Ecocyc: a comprehensive database resource for escherichia coli. *Nucleic Acids Res*, 33(Database issue), January 2005.

[5] R. D. King, K. E. Whelan, F. M. Jones, P. G. Reiser, C. H. Bryant, S. H. Muggleton, D. B. Kell, and S. G. Oliver. Functional genomic hypothesis generation and experimentation by a robot scientist. *Nature*, 427(6971):247–252, January 2004.

[6] V. S. Kumar, M. S. Dasika, and C. D. Maranas. Optimization based automated curation of metabolic reconstructions. *BMC Bioinformatics*, 8:212+, June 2007.

[7] D. C. McShan, M. Updadhayaya, and I. Shah. Symbolic inference of xenobiotic metabolism. *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing*, pages 545–556, 2004.

[8] P. R. Romero and P. Karp. Nutrient-related analysis of pathway/genome databases. *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing*, pages 471–482, 2001.

[9] P. Suppes. *Introduction to Logic*. D. Van Nostrand Co., Princeton, NJ, USA, 1957.

[10] G. J. Sussman. *A Computer Model of Skill Acquisition*. Elsevier Science Inc., New York, NY, USA, 1975.