Available online at www.sciencedirect.com

SCIENCE DIRECT

Journal of Theoretical Biology 230 (2004) 591–602

Journal of Theoretical Biology

www.elsevier.com/locate/jtbi

Netscan: a procedure for generating reaction networks by size☆

Bertrand Clarke[a], Glenn Fawcett[b], Jay E. Mittenthal[c,*]

[a] *Department of Statistics, University of British Columbia, Vancouver, BC, Canada*
[b] *Caracal Technologies, #106-1670 West 8th Ave, Vancouver, BC, Canada, V6J 1V4*
[c] *Department of Cell and Structural Biology, University of Illinois, 601 South Goodwin Avenue, Urbana, IL 61801-3714, USA*

Received 21 July 2003; received in revised form 24 February 2004; accepted 29 March 2004

Available online 10 July 2004

Abstract

In this paper, we describe an algorithm which can be used to generate the collection of networks, in order of increasing size, that are compatible with a list of chemical reactions and that satisfy a constraint. Our algorithm has been encoded and the software, called Netscan, can be freely downloaded from ftp://ftp.stat.ubc.ca/pub/riffraff/Netscanfiles, along with a manual, for general scientific use. Our algorithm may require pre-processing to ensure that the quantities it acts on are physically relevant and because it outputs sets of reactions, which we call canonical networks, that must be elaborated into full networks.
© 2004 Elsevier Ltd. All rights reserved.

Keywords: Reaction network; Network assembly; Network identification

1. Introduction

A reaction network is a system of coupled reactions, in which outputs of some reactions are inputs to others. Models for reaction networks are pivotal for our understanding of biological systems. Such models facilitate interpretation of data for metabolism, intracellular signaling, and the regulation of gene expression. The models also may predict results of further experiments. Assembling the networks implicit in a list of reactions is challenging because the list and the networks are often large; they may involve hundreds or thousands of kinds of molecules and reactions. Consequently, relating information about proteins and genes to their collective activity requires algorithms to assemble networks from reactions.

The problem we address is finding the networks, in order of increasing number of reactions taken from a specified list, that satisfy a constraint which describes the required function of the networks in terms of input and output molecules. Such a sequential method allows one to choose how extensive the search is. This option is useful if the ensemble of networks to be generated is very large, as may often be the case. Mittenthal et al. (2001) provided a method for generating metabolic networks in order of increasing size. Their method depends on the stoichiometry of reactions and seems to be restricted to metabolic networks, in which proteins are enzymes but not reactants. However, in most intracellular networks macromolecules can be reactants and can catalyse reactions. Our algorithm provides a way to assemble both metabolic and macromolecular networks in order of increasing size; i.e. in order of increasing number of distinct reactions.

1.1. Related work

The problem of assembling networks that meet constraints from a reaction list is one aspect of the problem of identifying networks compatible with data. Network identification typically proceeds by comparison of several data sets. The sets may display the abundance of messenger RNAs and/or proteins from a network with time-varying output, from versions of a

☆We dedicate this paper to the memory of Arthur Winfree. Art and Jay were friends for years, and Jay misses him. Art loved puzzles, and we think he would have enjoyed the puzzle of network assembly. He sought general principles underlying the diversity of phenomena. We hope that Netscan will be useful in the search for such principles.

*Corresponding author. Tel.: +1-217-333-4846; fax: +1-217-244-1648.

E-mail address: mitten@life.uiuc.edu (J.E. Mittenthal).

network with alternative mutations or pharmacological modifications, or from a network under various environmental conditions. From such data, methods of network identification may aim to show the pairwise interactions among molecules, to recognize subnetworks of molecules that are active in particular conditions, or to make dynamical models that can predict the network's operation in new contexts (e.g. Koza et al., 2001; Ideker et al., 2002; Kam, 2002; Kholodenko et al., 2002; Segal et al., 2003; Tegnér et al., 2003). In making dynamical models it is helpful to use a reaction list to find reaction networks that can meet specific constraints. Since this is the aim of our method, in the rest of this section we review other work with this aim.

The problem of finding reaction networks that are consistent with a reaction list and with constraints has been approached in various ways in organic chemistry (Corey and Cheng, 1989; Temkin et al., 1996) and in biochemistry (Seriossis and Bailey, 1988; Happel et al., 1990). Kosorukoff (1993, 1995, in Russian) implemented an algorithm for assembling networks. The motivating problem for that work arises in curriculum selection: given a list of courses available at a university, along with prerequisites and required advanced courses, which courses should a student take, and in what order? A course and its prerequisites correspond to a reaction. This method seems to generate individual networks rather than an ordered list of all networks compatible with the constraints.

Mavrovouniotis et al. (1990) provided a method to assemble all of the metabolic networks that meet specified constraints. Mittenthal (1996) extended this method from metabolic networks to macromolecular networks. This generalized method will be called Mavro. A limitation of Mavro is that it does not output any networks until all of them have been generated and the computation time required to generate all of the networks increases exponentially with the number of reactions. So, this approach is not well suited to the assembly of large networks. For gene regulatory networks Kolpakov et al. (1998) and Kolchanov et al. (2000) developed a program called GeneNet, which performs automated diagram generation using the method of Mittenthal (1996). (See wwwmgs.bionet.nsc.ru/mgs/systems/genenet/ for access to the software.)

1.2. Overview of our approach

Signaling networks provide a context for introducing our method. Suppose we have a collection of responses that we want to get as a function of certain input signals. We call such input–output relations constraints. The simplest constraints are of the form $A \gg B$, A OR $B \gg C$, and A AND $B \gg C$, but more elaborate constraints are readily imaginable. For ease of exposition here, we limit attention to networks that meet a single constraint. In fact, however, Netscan supports multiple constraints.

Our overall strategy for generating networks with specified properties has three stages. In the first stage, preprocessing, we specify the molecules, the allowed reactions, and the constraints. For example, suppose the molecules are proteins represented as sets of domains (e.g. Rzhetsky and Gomez, 2001). The user specifies reactions that occur through interactions of domains, such as phosphorylation of a domain by a protein kinase. The reactions among proteins can then be specified in terms of reactions of their domains.

In the second stage, networks are generated to meet the constraints. This is the most easily automated part of our approach; the main point of this paper is to describe the algorithm we have implemented to accomplish it. The third stage, post-processing, may impose additional performance criteria, such as energetics, on the networks output by the program. Post-processing may also be necessary to generate the actual forms of the molecules, the connectivity of the network, and the multiplicity of reactions.

Our program that implements the second stage is called Netscan. It can be downloaded free from ftp://ftp.stat.ubc.ca/pub/riffraff/Netscanfiles. This website includes the source files and documentation, including a manual, for their use. There are also several example files so one can test the executable obtained from compilation. (A sub-directory there contains the makefiles necessary for certain platform-dependent and development-tool-dependent settings.)

This paper is organized as follows. Section 2 presents the algorithm for Netscan. Section 3 relates our work to other algorithms and issues in computer science, and discusses some possible applications for Netscan. Appendix A presents an example of the use of Netscan.

1.3. Canonical networks

From the reaction list and the constraint, the Netscan algorithm generates sets of reactions that can satisfy the constraint. The reactions in each set are analogous to the pieces of a puzzle. They can be assembled into a unique network, which we call a canonical network, that is a simplification of an actual network formed by combining all occurrences of a reaction into one. Each canonical network must be post-processed into the collection of actual networks it subsumed by reconstituting the connectivity and multiplicity of the reactions in each actual network.

Formally, we say that two networks are equivalent if and only if they are formed from the same set of reactions. Note that this definition permits two networks with different connectivities and different numbers of usages of reactions to be equivalent. Since the relation defined in this way is reflexive, symmetric, and transitive it is a well-defined equivalence relation. Consequently, it leads to a list of equivalence classes. These classes are disjoint, and every network is in exactly one class. It can be seen that what we call canonical networks are in fact canonical representatives of classes of nets that are equivalent under this equivalence relation.

We generated canonical networks rather than actual networks for the sake of economy of computation. Consider the following example: suppose we have the overall constraint $A \gg B + E + G$ to satisfy and the permitted reactions are:

$R1 \;\; A \rightarrow B + C,$

$R2 \;\; C \rightarrow B + D,$

$R3 \;\; D \rightarrow E + F,$

$R4 \;\; F + B \rightarrow G.$

If we start with A, the only molecule in the constraint input, $R1$ is the only possible connection because it is the only reaction using A. Then there is only one possible connection from $R1$ to $R2$ and from $R2$ to $R3$. There are three ways to connect $R4$: F must come from $R3$ but B can come from $R1$ or $R2$, or from both. The output from our program is just the list $R1$, $R2$, $R3$, and $R4$, so the output does not distinguish among these three connectivities. Note that two of the three networks use $R4$ once, while the third uses it twice. As actual networks these three are not isomorphic. However, they correspond to the same canonical net, in which both B's are connected to G and to the overall output B. Note that in practice it is not necessary to construct the canonical network before constructing the actual networks.

This reduction to canonical nets means that for a set of N reactions there are at most $2^N - 1$ candidate canonical nets. This is so because each candidate canonical net either does or does not use each reaction, and the net that uses no reactions should not be counted. Without this reduction, there is no non-trivial upper bound on the number of non-isomorphic ways to connect inputs to outputs, nor is there any non-trivial upper bound on the number of nets if one takes multiplicity into account. Either of these upper bounds would depend delicately on the number of inputs, the number of outputs, and the number of permitted reactions.

Although this example is not complicated enough to show the procedure completely, a canonical net can in general be systematically elaborated into its functional nets. The procedure, for each compound, is to link the various places it is produced as an output to the various places it can be consumed as an input in all possible ways. The resulting nets have their connectivity specified. In some cases this yields isomorphic copies of a single net which should be eliminated. The set of nets remaining can be searched for a fully specified viable network which meets the constraints and is unique up to graph isomorphism.

1.4. Assumptions about network dynamics in Netscan

We make several assumptions to standardize the contexts in which our algorithm will apply. Our four main assumptions about the dynamics of networks of reactions are the following:

1. Any reactions for which substrates and enzymes are currently available can occur at the same time.
2. The competition between reactions for molecules is negligible. That is, the amount of each molecule used by a reaction is insignificant relative to the amount available.
3. Outputs of a reaction are available to all other reactions. This assumption is satisfied if, for instance, all reactions occur in a single well-mixed compartment. Or, a mechanism might assign outputs at random to reactions for which they can be inputs. Note that this assumption is somewhat unrealistic: cells often localize molecules in compartments such as molecular complexes or organelles, so as to reduce the diversity of uses for the output of each reaction.
4. Reactions are specific in that they ignore all molecules except their specified inputs. In particular, enzymes catalyse only those reactions they are allowed to. Consequently, we treat each enzyme as an input to a reaction. However, for convenience we continue to denote the activity of an enzyme C catalysing $A \rightarrow B$ as $A \rightarrow B|C$ even though our program treats it internally as $A + C \rightarrow B$. The amount of C consumed is not important under Assumption 2. Like Assumption 3, this assumption is not fully satisfied: enzyme specificity is high but not perfect.

A constraint, which specifies the required function of the generated networks in terms of input and output molecules, has the syntax *input-expression*$\gg$ *output-molecule-list* where *input-expression* is any combination of AND and OR logical operations on molecules, with parentheses allowed, for example:

$(A \text{ AND } B) \text{ OR } (A \text{ AND } C) \text{ OR } (B \text{ AND } C) \gg D + E.$

For a network to be generated, it must implement the logic of the constraint; i.e. the network must only output all of the molecules in the *output-molecule-list* when the molecules present satisfy the logic of the *input-expression*. When the input expression is not satisfied, not all the output molecules may be output (though some may).

For the above example of a constraint, if Netscan generates a network, the network will only output both molecules D and E from reactions in the network if at least two of the molecules A, B, and C are present. For example, such a network could consist of the following reactions (provided these reactions were contained in the reaction list given to Netscan, of course):

$A + B \to D,$

$A + C \to D,$

$B + C \to D,$

$D \to E.$

It is often necessary to ensure some molecules are available at all times. For a molecule C, this is accomplished by the command *PRESENT C*. Molecules that are always present need not be used by a reaction network; they are only used if required. In particular, this may be useful for molecules such as *ATP* or *NADH* which can be inputs and are freely available. Because the supply of always-present molecules is effectively infinite, within the program they are eliminated by modifying the list of permitted reactions. For instance, the three reactions

$A + B \to C,$

$D \to E|B,$

$B \to G|F,$

PRESENT B

are simplified by the program to,

$A \to C,$

$D \to E,$

$F \to G.$

After the program uses the above assumptions and conventions early in its execution, no molecules will be listed as merely present and all reactions will be of the (non-enzyme) form *input-list* $\to$ *output-list*.

Disinhibiting an inhibitor speeds a reaction and inhibiting a reaction slows it down. This may be done by compounds that are be produced upstream or downstream from where they act. That is, inhibition and disinhibition are just mechanisms by which compounds are fed forward or backward. These mechanisms are implemented by sets of reactions that permit a compound produced in one reaction to affect the rate of another reaction. This is easy to see if one regards a molecule that receives inhibition or disinhibition as already having been produced in a prior process.

In particular, even though a constraint input expression can only consist of AND and OR logical operators and not the NOT operator, Netscan can discover networks that implement inhibition and disinhibition and, more generally, positive and negative feedback. The main way this is done is to include the reactions necessary for constructing the mechanism. Consequently, searching for these kinds of mechanisms may require one to generate more nets than needed, winnow down to reasonable candidates, and post-process them to find the desired canonical nets which can be elaborated into functional nets.

Suppose A disinhibits C by binding to B, an inhibitor of C, causing dissociation of the complex $B{:}C$. (The colon indicates a non-covalent association.) For this to occur in a network generated by Netscan, $A + B{:}C \to C + A{:}B$ must be on the reaction list. Alternatively, A might modify B covalently so that $A : B$ is replaced by a compound AB. Similarly, the inhibition of a molecule requires the interaction of the inhibitor with that molecule in a permitted reaction. Netscan will show a disinhibition reaction as part of a pathway if the disinhibition makes a needed molecule available. In these settings, the complex, $B{:}C$, of an inhibitor B and the protein C it inhibits might need to be freely available before the disinhibitor A is produced.

Suppose the form of feedback under consideration uses a downstream molecule to affect an upstream reaction. An important instance is where a molecule is the output of a sequence of coupled reactions and also affects the activity of an upstream enzyme. For feedback to be incorporated in a generated network, the reaction list must include at least two reactions in which the compound to be fed back occurs. For instance, if an enzyme E catalyses a reaction whose outputs lead to a product that affects the activity of E one must include two reactions, one with E as a catalyst and one with E as an output. Often, in these cases, it is necessary to include E from the beginning by the command *PRESENT E*.

The following example may clarify these points:

$R1 \;\; A + C \to B,$

$R2 \;\; B \to C,$

$R3 \;\; B \to E|D,$

$R4 \;\; A + C \to B|E,$

PRESENT C,

PRESENT D,

$A \gg B.$

Four canonical nets are output: $R1$; $R1, R2$; $R1, R3, R4$; and $R1, R2, R3, R4$. The first is trivial. The second permits B to be fed back to C. In some cases, the third can be post-processed by deleting $R1$ so the feedback is to an enzyme which may upregulate production of B. It seems that if $R1$ is deleted, then no B is ever made because no E is initially available. However, once some E is made, the replenishment of E occurs by $R3$ and if $R4$ is much faster than $R1$, the latter will contribute negligibly to the overall output.

The fourth case combines the second and third, in which post-processing can delete $R1$ again. This fourth case produces a form of positive feedback that can be recognized as disinhibition. Indeed, let $C = C_1 : C_2$, $B = A : C$ and $E = C_1$. It is seen that, D can act on $A : C_1 :$

C_2 by forming $D:A:C_2$ and releasing C_1. In effect, D disinhibits C_1.

Netscan has a command-line option $(-r)$ to delete reactions that only replenish molecules declared *PRESENT*. With that option, the useless reaction $R2$ is deleted and two canonical nets are output: $R1$ and $R1, R3, R4$. The first is trivial and the second can be post-processed by deleting $R1$ by the same reasoning as above.

We comment that negative feedback can be treated in the same way. Here, we have been treating all feedbacks as positive because we are searching for pathways in which they appear. When the possible reactions are listed and the pathways are generated it is seen that feedback of either sort is just the interpretation we put on a sequence of reactions. The reactions themselves are qualitatively no different from any other reactions, and neither are the molecules in them. Thus, correct interpretation can then be included, as appropriate, in post-processing.

2. The algorithm for Netscan

Here we explain how the algorithm for Netscan works. Our problem is to find all subsets, from smallest to largest, of a set of reactions that can form reaction networks meeting an overall constraint.

When the user-specified reaction list and constraints are input to Netscan, the program begins with lexical analysis, parsing, and simplification to eliminate any duplicate reactions. We do not describe this here. (Details can be found in the Netscan manual.)

The first important conceptual stage is connectivity testing. This is followed by a search for candidate networks, from smallest to largest, and the testing of each candidate, as it is found, against the logic of the constraints.

2.1. Connectivity testing

Connectivity testing is used to eliminate useless reactions quickly and early, before they can slow the search for networks. For each constraint, it ensures that there is a path of reactions from each input molecule to at least one output molecule and a path from each output molecule to at least one input molecule. Connectivity testing uses the following rules:

1. A molecule is considered "input-connected to a constraint" if it is declared *PRESENT* or if it is found in the input expression of the constraint or if any of the reactions that generate it have inputs that can be connected to the constraint inputs by a sequence of coupled reactions.
2. A reaction is considered "input-connected to a constraint" if all of its input molecules are input-connected to the constraint.
3. A molecule is considered "output-connected to a constraint" if it is found in the output list of the constraint or if any reaction that uses it has outputs that are connected to the constraint outputs by a sequence of coupled reactions.
4. A reaction is considered "output-connected to a constraint" if any of its output molecules are output-connected to the constraint.

We comment that in item 3, if the "delete reactions that only replenish molecules declared *PRESENT*" option is specified, a molecule that is *PRESENT* is not considered "output-connected" to any constraint even though it may be connected to the overall outputs by a sequence of coupled reactions. Clearly, *PRESENT* molecules are upstream of the constraint. However, they are also downstream from the outputs because they cycle. Our convention is to 'cut' the net immediately after the constraint is met.

For each constraint, Netscan solves the above rules for all reactions and molecules. It starts from each input molecule and works forward through each reaction using the molecule, finding and marking the reactions as input-connected. Each reaction found then has its output molecules marked as input-connected. The process is repeated until no more reactions are found. Circular paths are prevented by not following a path already marked as input-connected.

Then, for each constraint, Netscan starts from each output molecule and works backward through each reaction producing the molecule, finding and marking the reactions as output-connected. Each reaction found then has its input molecules marked as output-connected. The process is repeated until no more reactions are found. Circular paths are prevented by not following a path already marked as output-connected.

A reaction is considered useless if there is no constraint for which it is both input-connected and output-connected. All useless reactions are deleted. Connectivity testing is repeated until no more useless reactions are deleted.

A molecule is considered useless if it is not in the input expression or output list of any constraint and it does not feed any reaction that remains after useless reactions are deleted. All useless molecules are deleted.

2.2. The search algorithm for Netscan

The program searches for networks using structures called "searchsets". A searchset represents a partially completed network. In particular, a searchset contains a reaction list (giving the partially completed canonical network), a disallowed list (containing reactions that will never be added to the network), and a search list (containing required molecules).

Initially, the program creates a single searchset with an empty reaction list, an empty disallowed list, and a search list containing the molecules in the output lists of all constraints. The algorithm then uses the following steps:

1. If there are no searchsets, the program terminates. Otherwise, the searchset with the smallest number of reactions on its reaction list is processed by step 2 (in the case of a tie, an arbitrary searchset is picked).
2. The searchset's reaction list is examined. If the following conditions are met, the reaction list is deemed to be a network candidate and is tested for constraint logic as described in the next section:

   a. The reactions use all of the molecules in all constraint input expressions. b. The reactions output all of the molecules in all constraint output lists. c. All the molecules on the search list are PRESENT, are produced by a reaction on the reaction list, or are in a constraint input expression.

3. If there are no molecules on the searchset's search list, the searchset is deleted and the algorithm returns to step 1. Otherwise, the molecule on the search list with the least number of reactions generating it (looking at the complete list of user-specified reactions) is removed and examined to find the number of reactions generating it that are not on either the searchset's reaction list or disallowed reaction list. If one or more new reactions are found, the algorithm proceeds to step 4. Otherwise, this step repeats.
4. If there are N new reactions producing a molecule, 2N−1 searchsets will replace the searchset being processed, with a new searchset created for each possible combination of reactions. Each new searchset will be identical to the old searchset but with all the new reactions added, either on its reaction list or on its disallowed list. (For example, if the new reactions were R4 and R5, three new searchsets will replace the old one. The first will have R4 added to its reaction list and R5 added to its disallowed list. The second will have R5 added to its reaction list and R4 added to its disallowed list. The third will have R4 and R5 added to its reaction list and nothing added to its disallowed list.) Also, each new searchset will have the input molecules of the reactions added to its reaction list (but not its disallowed list) added to its search list. The algorithm then continues at step 1.

2.3. Constraint logic testing

Before searching begins, Netscan generates a set of test cases from the logic of each constraint. Then, as each candidate network is found, it is tested against all the cases. Networks that pass all the tests are output and networks that do not are discarded.

Each test case is a set of inputs to the network along with an expected output. The inputs consist of values (present or not-present) for every molecule in the input expression of the constraint. The output consists of a value specifying if the molecules on the constraint's output list must all be present or cannot all be present.

For each test case, the molecules specified in the test case are set to the values in the test case, *PRESENT* molecules are set to be present and molecules not in the test case are set to be not-present. The reactions are simulated until the molecule values stabilize and then the constraints' output molecules are examined to determine if they are all being generated. This result must match the test case's output value.

A network can only implement logical OR and AND operations, not logical NOT operations. This is because reactions create the AND of their molecular inputs and molecules create the OR of reactions that can produce the molecule. Because a network cannot implement all possible logical operations, the number of test cases usually can be reduced. In particular, test cases can be eliminated if they have a present output and their inputs are a superset of some other case. Similarly, test cases can be eliminated if they have a non-present output and their inputs are a subset of some other case.

Netscan considers all possible test cases and eliminates unnecessary ones. For example, consider the constraint

$(A \text{ AND } B) \text{ OR } (C \text{ AND } D) \geqslant E.$

There are sixteen possible test cases (where 1 indicates a molecule is present and 0 indicates not-present):

Case 1: $A=0,\ B=0,\ C=0,\ D=0,\ Output=0,$
Case 2: $A=1,\ B=0,\ C=0,\ D=0,\ Output=0,$
Case 3: $A=0,\ B=1,\ C=0,\ D=0,\ Output=0,$
Case 4: $A=1,\ B=1,\ C=0,\ D=0,\ Output=1,$
Case 5: $A=0,\ B=0,\ C=1,\ D=0,\ Output=0,$
Case 6: $A=1,\ B=0,\ C=1,\ D=0,\ Output=0,$
Case 7: $A=0,\ B=1,\ C=1,\ D=0,\ Output=0,$
Case 8: $A=1,\ B=1,\ C=1,\ D=0,\ Output=1,$
Case 9: $A=0,\ B=0,\ C=0,\ D=1,\ Output=0,$
Case 10: $A=1,\ B=0,\ C=0,\ D=1,\ Output=0,$
Case 11: $A=0,\ B=1,\ C=0,\ D=1,\ Output=0,$
Case 12: $A=1,\ B=1,\ C=0,\ D=1,\ Output=1,$
Case 13: $A=0,\ B=0,\ C=1,\ D=1,\ Output=1,$
Case 14: $A=1,\ B=0,\ C=1,\ D=1,\ Output=1,$
Case 15: $A=0,\ B=1,\ C=1,\ D=1,\ Output=1,$
Case 16: $A=1,\ B=1,\ C=1,\ D=1,\ Output=1.$

Because of a present output, case 16 can be eliminated because it is a superset of cases 12–15, case 15 can be eliminated because it is a superset of case 13, case 14 can be eliminated because it is a superset of case 13, case 12 can be eliminated because it is a superset of case 4, and case 8 can be eliminated because it is a superset of case 4. With a not-present output, case 1 can be eliminated because it is a subset of cases 2, 3, 5, 6, 7, 9, 10 and 11, case 2 can be eliminated because it is a subset of cases 6 and 10, case 3 can be eliminated because it is a subset of cases 7 and 11, case 5 can be eliminated because it is a subset of cases 6 and 7, and case 9 can be eliminated because it is a subset of cases 10 and 11. This leaves six test cases: 4, 6, 7, 10, 11, and 13.

3. Discussion

Here we relate our work to other algorithms for network assembly, to issues in computer science, and to potential applications.

3.1. Comparison to other algorithms for network assembly

This paper presents an algorithm that gives a sequence of networks meeting a given constraint. The inputs to the algorithm are the constraint and a list of allowed reactions. The output is a sequence of canonical nets—sets of reactions. In principle our approach can be extended to include more constraints, such as requiring or forbidding the usage of certain molecules. The algorithm produces networks in order of increasing size (number of reactions used counted with multiplicity).

The starting point for development of the Netscan algorithm was the method we call Mavro, which was presented by Mavrovouniotis et al. (1990). In each iteration Mavro augments partial pathways by linking the reactions that produce or consume a chosen molecule, using all possible combinations of reactions. The molecule chosen is the one involved in the fewest partial pathways. Because Mavro may begin with molecules that do not necessarily participate in partial pathways that produce the outputs or consume the inputs given in the constraint, it is more productive to choose at each iteration a molecule that helps to produce the overall outputs. Thus, a natural variant on Mavro is to begin with the overall outputs and work backwards to the overall inputs, somewhat as Netscan does. This tends to give networks in order of increasing size but this is not guaranteed. Also, this "backwards Mavro" does generate networks efficiently, because it processes all molecules a fixed number of reaction steps from the overall outputs. This processing is backward to the overall inputs initially but in later iterations can include output edges from reactions whose products link forward also.

The Netscan algorithm represents each partial pathway as a searchset, as defined in Section 2.2. At each iteration Netscan processes the searchset that has the smallest number of included reactions. Within that node, it processes those molecules that lead to the fewest new nodes. However, Netscan only searches backward to the overall inputs. Consequently, a reaction that produces one molecule needed for the overall outputs and one molecule that is not the input to any other reaction may be included in a network. That is, the networks generated are not in general complete: a constraint need not be satisfied stoichiometrically. Not all molecules produced are necessarily consumed.

There are three consequences of this approach. First is that the networks are produced in order of size because of the search algorithm. Second is that the stoichiometrically complete nets will be a proper subset of the canonical nets Netscan generates. Indeed, if a molecule is produced but not consumed, but can be consumed by the addition of further reactions that bigger network will appear later in the Netscan output. We note that nets with molecules that are consumed but not produced are not included in the output. Third, by searching backwards only, Netscan avoids some of the potential pitfalls of Mavro. That is, Mavro will search forward from molecules that are produced but not yet consumed to track where they go. By contrast, because Netscan assembles networks from the overall outputs to the overall inputs it diversifies partial pathways sparingly in contrast to Mavro.

As Mavrovouniotis et al. (1990) emphasized, their algorithm is complete; it generates all networks that meet the constraint exactly, given a reaction list. However, its algorithm incorporates all reactions that produce each molecule and all reactions that consume it, in all combinations, into partial pathways. Some of these partial pathways may not include overall inputs or outputs. Mavro generates all possible occurrences of each molecule, at any position in any network. Like Mavro, Netscan is complete in that it generates all reaction sets that can satisfy a given set of constraints.

An important generic example is polymerization in which the same reactions are used many times. That is, a reaction can be reused if its inputs recur. In fact, the recurrence may be associated with a futile cycle or with the reoccurrence of the inputs in a novel context, or with the reoccurrence of the inputs in the same context. The synthesis of a protein is an example. At each position where a given amino acid is to be added to the growing polymer, the same reactions get re-used, because the same codon reoccurs in different contexts. In a case like this Netscan would generate the appropriate canonical nets but post-processing would be necessary to count how many times each reaction was used to make the protein.

Thus the above algorithms make different tradeoffs. Mavro generates all networks exactly compatible with the constraint, preserving the connectivity of reactions in each network. To accomplish these aims the algorithms sacrifice speed and use extensive storage. By contrast, Netscan sacrifices connectivity and may overgenerate nets for speed, size ordering, and reduced storage. Consequently, Netscan should deal effectively with much larger reaction lists than Mavro can.

3.2. Relations to issues in computer science

For computational biology and bioinformatics one wants a method of network assembly that works for hundreds or thousands of reactions. Assembly proceeds stepwise. A measure of the difficulty of assembly is the branching factor B, the expected number of ways an output from one step can be connected to inputs for the next step. The examples we have explored suggest that finding all paths that use S steps to get from specified inputs to outputs requires B^S steps—network assembly is exponential in S. That is, it may not be possible to assemble the networks that meet a constraint with an algorithm that is polynomial in S. If it is not possible, network assembly is NP-hard. Below we argue that this is the case, as Mavrovouniotis et al. (1990) concluded for Mavro. We also discuss alternative algorithms and scaling of the algorithm we used.

3.2.1. Network assembly is related to routing and shortest-path problems

The problem of assembling reaction networks is related to the problem of routing transmissions from senders to receivers through a network. There, nodes represent senders and receivers, and edges represent routes of transmission. Many algorithms are available for the routing problem if transmission through a node obeys an OR function, with any input to the node eliciting its output (Bertsekas and Gallager, 1992; Russel and Norvig, 1995). However, in a typical molecular reaction a conjunction of inputs determines the outputs; such a node obeys an AND function. Other Boolean functions are also possible.

Algorithms for finding the shortest path between two nodes might be useful for assembling networks. This seems unlikely. Standard shortest-path algorithms such as the Dijkstra algorithm (Dijkstra, 1959) and the Bellman–Ford algorithm (Bellman, 1958; Ford and Fulkerson, 1962) determine the shortest path between two nodes on a weighted undirected graph. After all nodes are enumerated, these algorithms change variables associated with each node until a weighted path length is calculated.

The impediment to using these algorithms for finding the networks that meet a constraint, in order of increasing size, is that the algorithms require enumeration of all nodes. For a cycle, enumeration of repeated traversals produces an infinite number of nodes and arcs, so the algorithm would never terminate. In the absence of cycles there will be finitely many nodes. Indeed, there is a finite number of molecules and there are finitely many reactions.

Thus, to list the networks that can satisfy a constraint in order of increasing size, there are three problems to overcome if one wants to modify standard shortest path algorithms. First, one needs to enumerate the nodes, but this must not be done by following connections because there may be an infinite number of them. Instead, one must generate the nodes by following all the paths, almost certainly using a tree search. This suggests that a search over paths is unavoidable. Second, a shortest-path algorithm is particularly useful only for a weighted network. (Reductions to unweighted networks achieved by setting all weights to 1 loses the improvement in the search achieved by having substantially different weights.) Thus, using this approach in practice needs more information or stronger assumptions than is typically the case. Third, even if these impediments are overcome, a shortest path algorithm will only find the shortest path. Our task is to generate all paths, in order of increasing size. A shortest-path algorithm would need substantial modification to do this. The result would be an algorithm combining a breadth-first search with a shortest-path algorithm.

3.2.2. Network assembly is a generalization of the minimum cover problem

Garey and Johnson (1979) defined the minimum cover problem as follows. Suppose we have a set S and a finite class of sets $C = \{C_1, \ldots, C_M\}$. Each C_i is a subset of S. Given an integer K, we can ask if there is a subclass of C of size K with the union of those C_i's, from $i = 1, \ldots, K$, containing S. The minimum cover problem is known to be NP-hard.

To represent this problem as a special case of network assembly, let S represent the set of outputs in a constraint, and denote the cardinality of S by u. Choose each element C_i in C to be the outputs of a reaction R_i. Now, let each R_i convert the null element ϕ into a subset of S. There are $2^u - 1$ non-void subsets of S and so there are $2^u - 1$ reactions R_i. An algorithm for network assembly should output all sets of R_i's that produce the outputs in the constraint. This is a solution to the minimum cover problem.

The problem of network assembly is NP-complete: since the minimum cover problem is NP-hard, and it is a special case of the problem of network assembly, the latter problem must be NP-hard. An NP-hard problem is NP-complete if one can test in polynomial time whether a candidate solution solves the problem. This is so for network assembly: given a set of reactions we can represent each by a row containing 1's for molecules produced and -1's for molecules consumed. Each column represents a molecule, so verifying that each column has at least one -1 and one 1 means we can form links to get the overall outputs. This means the set of reactions is a solution. The test can be done in polynomial time—the size of the problem is linear in the number of reactions and in the number of outputs/inputs.

Thus network assembly is NP-complete. Its constraint on the graph-theoretic connectivity of a set of reactions generalizes the minimum cover problem's constraint on a union of sets.

3.2.3. Scaling

Although network assembly is NP complete, the complexity of the algorithm used by Netscan is not exponential with the total number of reactions but rather with the number of reactions producing the same molecule. Because many reactions in biology are quite specific, the program may be useful in many cases, in particular, if less than about 16 reactions produce any one molecule. If so, tens of thousands of reactions can be searched.

It is possible to estimate the complexity (in both memory and execution time) by examining each molecule and looking to see how many potential reactions produce it and calculating a "branching factor" of $2^n - 1$, where n is the number of reactions producing the molecule. The overall complexity is determined by multiplying all the branching factors together and multiplying this product by the overall number of reactions. In fact, before it begins its search, Netscan performs exactly this calculation and prints out the result as estimated execution time and memory required. This lets the user know what to expect.

In addition, as noted, the program also attempts to minimize complexity by performing a connectivity check before the search begins to delete unnecessary (i.e. not fully connected) reactions and molecules. It also combines duplicate reactions that would otherwise needlessly increase the complexity.

3.3. Some possible uses for the algorithm

As noted earlier, the procedure we have presented is intended primarily for assembling regulatory networks, such as signaling pathways and gene regulatory nets. As such it neglects the stoichiometry of reactions, and it is therefore more general than methods that require the use of stoichiometry (e.g. Mavrovouniotis et al., 1990; Mittenthal et al., 1998, 2001).

Contexts in which our program may be helpful include the following. It can be used, in conjunction with post-processing, to give a display of networks corresponding to a database of reactions. EcoCyc (www.ecocyc.org) demonstrated the efficiency of automatically generating diagrams of metabolic pathways from a database of reactions. The Netscan program could also be useful in automating the calculation of global topological properties of networks, such as the number of interactions per molecule.

Our program can be used to help infer routes through which one molecule can influence another, in wild-type organisms or after alteration of a network through mutation or pharmacological intervention. This may have particular use in inferring gene regulatory nets from DNA microarray data. An efficient method for assembling networks from hundreds or thousands of reactions would be a significant step toward a program that generates an exhaustive listing of possible consequences of a given perturbation, such as inhibition of the activity of a protein kinase.

The program could also be used in assembling networks during an evolutionary computation. In such a computation a population of model cells undergoes iterative mutation and selection, with the aim of evolving cells that have higher fitness; fitness is specified according to a selection criterion. Each cell contains a set of reactions. In each iteration the reaction set in one or more cells is mutated. The networks of reactions that determine the fitness of a mutated cell must be assembled from its altered reaction set. If the fitness depends on the kinetic parameters of a network as well as its topology, the automatic generation of nets that can satisfy an input-output constraint is desirable because such a procedure eliminates networks that cannot meet the constraint before optimization of kinetic parameters is attempted. This separation of topological and dynamical evaluation has the potential to accelerate evolutionary computation significantly by reducing the number of networks to be considered.

Acknowledgements

The first author's work was supported by NSERC Operating Grant with RGPIN #138122-00. The authors wish to thank Gary Catt of Caracal Consulting, profusely, for invaluable help navigating the computing labyrinth. The authors thank Ann Condon for pointing out the connection between network assembly and the minimum cover problem.

Appendix A. A Netscan example

Here we examine a signaling network that computes AND from its inputs. Specifically, we use Netscan to assemble networks in which a conjunction of inputs determines the activity of a protein. Schmelzle and Hall (2000) reviewed studies of the target of rapamycin (*TOR*) network of reactions in yeast. They related it to the mammalian *TOR* (*mTOR*) pathway, which stimulates both translation and transcription in response to the presence of amino acids and growth factors. Here, we limit our attention to the part of the *mTOR* network used for translation. When ligand L binds to receptor R

and amino acids are present, translation processes denoted *translation* are activated to *translation**. (An asterisk denotes activation.) Thus our constraint is $L+R+amino\ acdis \gg translation^*$; it is of the form A AND $B \gg C$. If we write down the reactions used in Schmelzle and Hall (2000) and indicate which molecules are always present for use we have the following list.

*Rx*1 $L+R \to L{:}R$,
*Rx*2 $PI3K \to PI3K^* | L{:}R$,
*Rx*3 $PDK1 \to PDK1^* | PI3K^*$,
*Rx*4 $PKB \to PKB^* | PI3K^* + PDK1^*$,
*Rx*5 $X \to X^* | PKB^*$,
*Rx*6 $mTOR \to mTOR^* | amino\ acids$,
*Rx*7 $p70s6k \to p70s6k^* | mTOR^* + PDK1^*$,
*Rx*8 $S6 \to S6^* | p70s6k^*$,
*Rx*9 $4E\text{-}BP1{:}eIF4E \to 4E\text{-}BP1 + eIF4E^* | X^* + mTOR^*$,
*Rx*10 $translation \to translation^* | eIF4E^* + S6^*$.

Here, we assume *PI3K*, *PDK*1, *PKB*, *X*, *amino acids*, *mTOR*, *p*70*s*6*K*, *S*6, *4EBP1:eIF4E*, and *translation* are always present. It can be seen that *Rx*9 is a disinhibition reaction and *X* is a placeholder for an uncharacterized molecule. Using this reaction list as its input, Netscan gives the diagram in Schmelzle and Hall (2000, Fig. 3). This solution is in fact the only one. However, we can make the problem more general and interesting by modifying it to search for alternatives that might be biochemically plausible. This modification might correspond to a search among related networks to see if the existing network is optimal. It might correspond to efforts to uncover the biochemical correlates of *X* and *X**.

Consider the following abstract extension based on the *mTOR* network. It continues to have a constraint requiring three inputs, but it includes a recurrence loop (through reactions 6, 10, 15, and 19 below), and several false paths have been added by adding imaginary reactions and molecules. The network of reactions is summarized graphically in Fig. 1. Here is the reaction list that generated this figure. Note that we include molecules that are freely available by writing them as *PRESENT*.

*R*1 $A+B \to C+II$.

Here, *A* and *B* correspond to the ligand (*L*) and its receptor (*R*); *C* is the complex they form and *II* is imaginary.

*R*2 $II \to JJ$.

This is an imaginary reaction.

*R*3 $N+P1 \to E+EE|C$,
*PRESENT P*1.

This is a combination of several early reactions in the *mTOR* network: *N* corresponds to *mTOR**, *E* corresponds to *PI3K**, *EE* corresponds to *PDK1**, and *P*1 represents a combination of *PI3K* and *PDK*1.

*R*4 $QQ \to N+NN$.

With *R*24 below, this is an artificial inflation of $mTOR \to mTOR^* | amino\ acids$; *N* corresponds to *mTOR** and *NN* is imaginary.

*R*5 $P2 \to I+PP|E$,
*PRESENT P*2.

Here, *E* corresponds to *PI3K**, *P*2 corresponds to *PKB*, and *I* corresponds to *PKB**, while *PP* is imaginary and *PDK1** is ignored on the left-hand side.

*R*6 $EE+X \to G+GG$.

Here, *EE* corresponds to *PDK1**, while *X*, *G* and *GG* are imaginary. With *R*10 this gives *P* corresponding to *p70s6k**.

*R*7 $PP \to Z$,
*R*8 $P3 \to OO|PP$,
*PRESENT P*3.

These two reactions, *R*7 and *R*8, are imaginary.

*R*9 $I \to K$.

Here, *I* corresponds to *PKB** and *K* corresponds to *X**, with *X* in the *mTOR* network not represented.

*R*10 $G+GG+N \to P$.

Here, *G* and *GG* are imaginary, and *N* corresponds to *mTOR**. With *R*4 we have *P* corresponding to *p70s6k**.

*R*11 $P \to HH+LL$,
*R*12 $Z \to BB|AA$,
*R*13 $AA+OO \to CC$,
*R*14 $P4 \to FF|OO$,
*PRESENT P*4.

These four reactions are imaginary.

*R*15 $P5 \to R+MM|P$,
*PRESENT P*5.

Here, *P* corresponds to *p70s6k**, *P*5 corresponds to *S*6, and *R* corresponds to *S6**, with *MM* imaginary.

*R*16 $BB \to DD$,
*R*17 $CC \to DD$,
*R*18 $P6 \to DD|KK$,
*PRESENT P*6,
*R*19 $P7 \to X|MM$,
*PRESENT P*7,
*R*20 $DD \to U$.

Reactions *R*16–*R*20 are imaginary.

*R*21 $K+N \to T+U$.

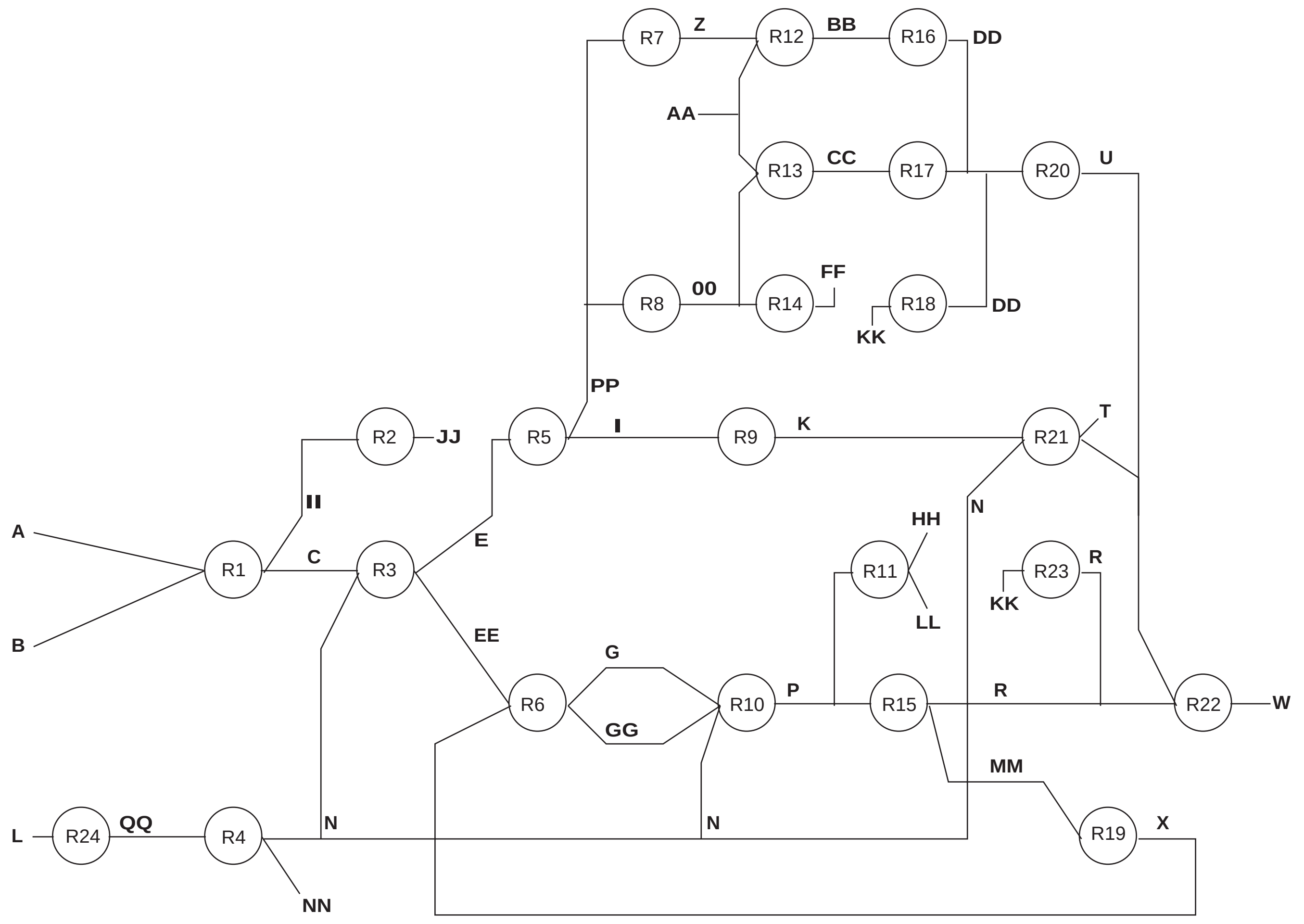

Fig. 1. The connectivity of the artificially extended *mTOR* network from Appendix A is shown. Note that some molecules, such as *P*1–*P*7, that are specified as always *PRESENT* are not shown. Also, molecules used as catalysts are shown as inputs even though they are not consumed.

Here, K corresponds to X^*, N corresponds to $mTOR^*$, T corresponds to *4EBP1*, and U corresponds to $eIF4E^*$.

$R22 \quad R + U \rightarrow W.$

Here, R corresponds to $S6^*$, U corresponds to $eIF4E^*$, and W corresponds to *translation*.

$R23 \quad KK \rightarrow R.$

This reaction is imaginary.

$R24 \quad L \rightarrow QQ.$

With $R4$, this is an imaginary extension of $mTOR \rightarrow mTOR^* | amino\ acids$.

In our more abstract notation the constraint is

$A + B + L \gg W.$

Netscan finds no solution because the recurrence prevents the required outputs from being produced. If the recurrence is satisfied by adding the line *PRESENT MM*, Netscan gives the single solution

$R1, R3, R4, R5, R6, R9, R10, R15, R19, R21, R22, R24.$

If *PRESENT AA* is added, Netscan gives seven solutions:

$R1,R3,R4,R5,R6,R9,R10,R15,R19,R21,R22,R24,$
$R1,R3,R4,R5,R6,R7,R10,R12,R15,R16,R19,$
$R20,R22,R24,$
$R1,R3,R4,R5,R6,R8,R10,R13,R15,R17,R19,$
$R20,R22,R24,$
$R1,R3,R4,R5,R6,R7,R9,R10,R12,R15,R16,$
$R19,R20,R21,R22,R24,$
$R1,R3,R4,R5,R6,R8,R9,R10,R13,R15,R17,$
$R19,R20,R21,R22,R24,$
$R1,R3,R4,R5,R6,R7,R8,R10,R12,R13,R15,$
$R16,R17,R19,R20,R22,R24,$
$R1,R3,R4,R5,R6,R7,R8,R9,R10,R12,R13,$
$R15,R16,R17,R19,R20,R21,R22,R24.$

If *PRESENT AA* is deleted and the constraint is changed to

$$A + B + L + AA \geqslant W.$$

Netscan reports three solutions:

$R1, R3, R4, R5, R6, R7, R10, R12, R15, R16, R19, R20, R22, R24,$
$R1, R3, R4, R5, R6, R8, R10, R13, R15, R17, R19, R20, R22, R24,$
$R1, R3, R4, R5, R6, R7, R8, R10, R12, R13, R15, R16, R17, R19, R20, R22, R24.$

In this way one can generate a class of networks within which one can evaluate the comparative performance of a specified network.

References

Bellman, R., 1958. On a routing problem. Q. Appl. Math. 16, 87–90.

Bertsekas, D., Gallager, R., 1992. Data Networks, 2nd Edition. Prentice-Hall, New Jersey.

Corey, E.J., Cheng, X.-M., 1989. The Logic of Chemical Synthesis. Wiley, New York.

Dijkstra, E.W., 1959. A note on two problems in connection with graphs. Numer. Math. 1, 269–271.

Ford, L., Fulkerson, D.R., 1962. Flows in Networks. Princeton University Press, Princeton, NJ.

Garey, M.R., Johnson, D.S., 1979. Computers and Intractability: A Guide to the Theory of NP-Completeness. W H Freeman & Co., New York.

Happel, J., Sellers, P.H., Otarod, M., 1990. Mechanistic study of chemical reaction systems. Ind. Eng. Chem. Res. 29, 1057–1064.

Ideker, T., Ozier, O., Schwikowski, B., Siegel, A.F., 2002. Discovering regulatory and signalling circuits in molecular interaction networks. Bioinformatics 18 (Suppl. 1), S233–S240.

Kam, Z., 2002. Generalized analysis of experimental data for interrelated biological measurements. Bull. Math. Biol. 64, 133–145.

Kholodenko, B.N., Kiyatkin, A., Bruggeman, F.J., Sontag, E., Westerhoff, H.V., Hoek, J.B., 2002. Untangling the wires: a strategy to trace functional interactions in signaling and gene networks. Proc. Natl Acad. Sci. USA 99, 12841–12846.

Kolchanov, N.A., Ananko, E.A., Kolpakov, F.A., Podkolodnaya, O.A., Ignat'eva, E.V., Goryachkovskaya, T.N., Stepanenko, I.L., 2000. Gene networks. Mol. Biol. (Msk.) 34, 449–460.

Kolpakov, F.A., Ananko, E.A., Kolesov, G.B., Kolchanov, N.A., 1998. GeneNet: a database for gene networks and its automated visualization. Bioinformatics 14, 529–537.

Kosorukoff, A., 1993. Synthesis of static situative structures. Mathematical model. Unpublished manuscript available at http://www.geocities.com/alex + kosorukoff/papers.html.

Kosorukoff, A., 1995. Synthesis of resource exchange structures (in Russian). Unpublished M.Sc. Dissertation from the Computer Aided Design Department, Ivanovo State Power University. Available at http://alkos.chat.ru/dissert/disstcfm.html.

Koza, J.R., Mydlowec, W., Lanza, G., Yu, J., Keane, M.A., 2001. Reverse engineering of metabolic pathways from observed data using genetic programming. In: Altman, R.B., Dunker, A.K., Hunter, L., Klein, T.E. (Eds.), Pacific Symposium on Biocomputing 6. World Scientific, Singapore, pp. 434–445.

Mavrovouniotis, M.L., Stephanopoulos, G., Stephanopoulos, G., 1990. Computer-aided synthesis of biochemical pathways. Biotechnol. Bioeng. 36, 1119–1132.

Mittenthal, J.E., 1996. An algorithm to assemble pathways from processes. In: Altman, R.B., Dunker, A.K., Hunter, L., Klein, T.E. (Eds.), Pacific Symposium on Biocomputing '97. World Scientific, Singapore, pp. 292–303.

Mittenthal, J.E., Yuan, A., Clarke, B., Scheeline, A., 1998. Designing metabolism: alternative connectivities for the pentose phosphate pathway. Bull. Math. Biol. 60, 815–856.

Mittenthal, J.E., Clarke, B., Waddell, T.G., Fawcett, G., 2001. A new method for assembling metabolic networks, with application to the Krebs citric acid cycle. J. Theor. Biol. 208, 361–382.

Russel, S.J., Norvig, P., 1995. Artificial Intelligence: A Modern Approach. Prentice-Hall, Englewood Cliffs, NJ.

Rzhetsky, A., Gomez, S.M., 2001. Birth of scale-free molecular networks and the number of distinct DNA and protein domains per genome. Bioinformatics 17, 988–996.

Schmelzle, T., Hall, M.N., 2000. TOR, a central controller of cell growth. Cell 103, 253–262.

Segal, E., Shapira, M., Regev, A., Pe'er, D., Botstein, D., Koller, D., Friedman, N., 2003. Module networks: identifying regulatory modules and their condition-specific regulators from gene expression data. Nat. Genet. 34, 166–176.

Seriossis, A., Bailey, J.E., 1988. MPS: an artificially intelligent software system for the analysis and synthesis of metabolic pathways. Biotechnol. Bioeng. 31, 587–602.

Tegnér, J., Yeung, M.K.S., Hasty, J., Collins, J.J., 2003. Reverse engineering gene networks: integrating genetic perturbations with dynamical modeling. Proc. Natl Acad. Sci. USA 100, 5944–5949.

Temkin, O.N., Zeigarnik, A.V., Bonchev, D., 1996. Chemical Reaction Networks. CRC Press, Boca Raton, FL.