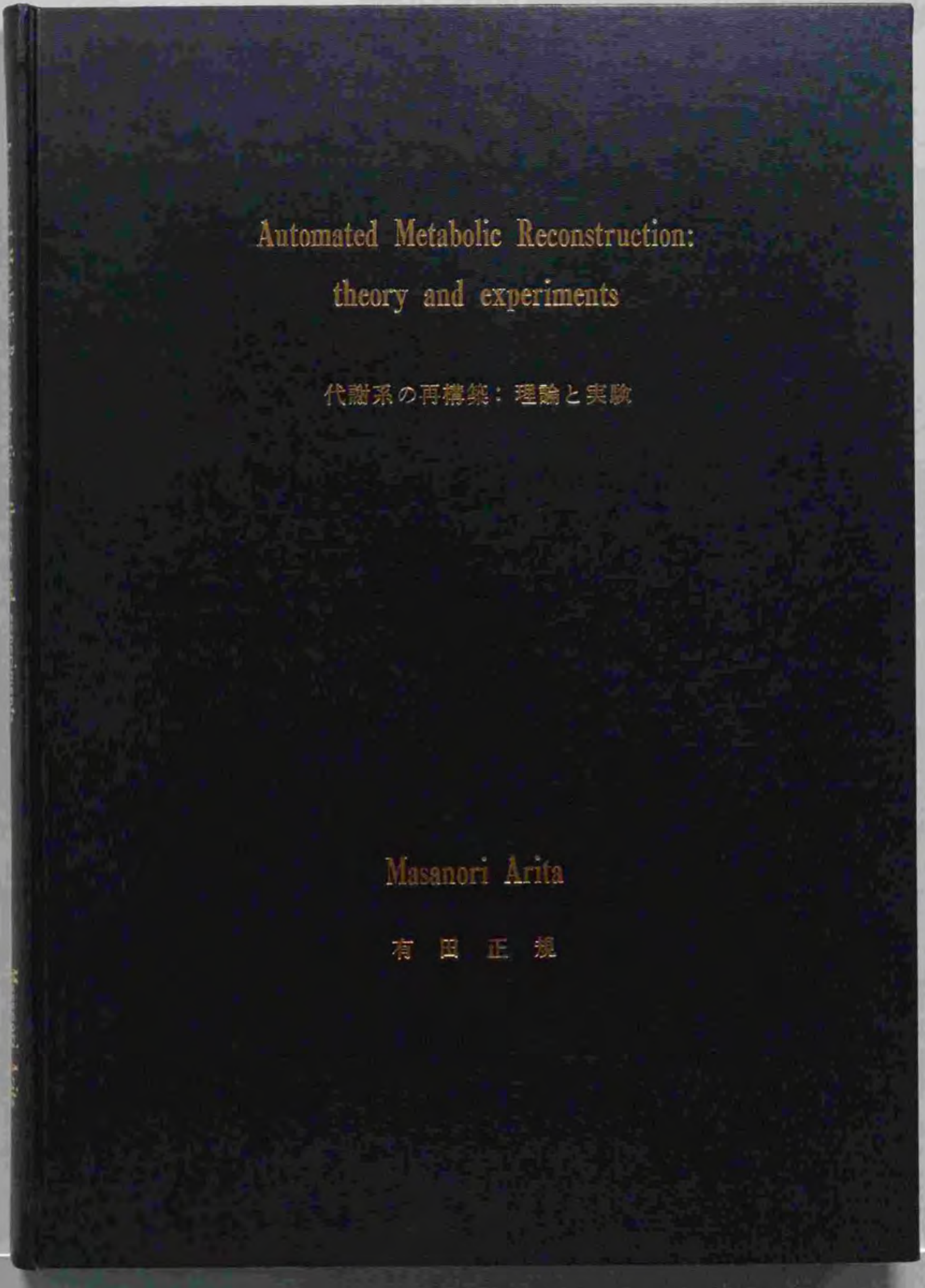


Automated Metabolic Reconstruction: theory and experiments

代謝系の再構築：理論と実験

Masanori Arita

有田正規

Automated Metabolic Reconstruction: theory and experiments
代謝系の再構築: 理論と実験

Masanori Arita
有田正規

ABSTRACT

Systematic understanding of biological phenomena is crucial in biological modeling. The hitherto analysis using differential equations is merely an imitation of observed results in a top-down manner. What is necessary is a bottom-up method to computationally reconstruct biological systems.

The best understood research area in biology is metabolism, i.e. the process of decomposing or synthesizing compounds in a cell. The reconstruction of bacterial metabolism from sequences not only elucidates the mystery of life, but also predicts the function of yet unknown genes, and pioneers the way to bioengineering.

The prerequisites of this reconstruction is an appropriate knowledge representation of metabolism, and the computerization of molecular structures and enzymatic reactions. Moreover, the results of computational prediction must be finally verified in laboratory works. The paper organizes these problems and solves them.

In Chapter 1, we clarify why the systematic analysis of life is important. Also shown is the desirable knowledge representation and the necessary steps for computerizing metabolism, with historical explanation of related works.

In Chapter 2, we introduce a new framework and its notation for the metabolic reconstruction. The framework describes the characteristics of metabolism as a graph, and a user can search any metabolic pathway with a variant of the shortest paths algorithm. It is also possible to search and generate the pathway-maps of an arbitrary shape, by combining multiple of shortest paths. A metabolic version of the framework is implemented, and is used for the analysis of bacterial metabolism. The framework itself is general enough, moreover, to treat problems in computational biology such as signal transduction.

In Chapter 3, three procedures for computerizing chemical molecules and enzymatic reactions are introduced, and we show their solutions: 1. finding aromatic cycles in the input molecular structure; 2. finding an automorphism group for normalizing the structure; and 3. finding the mapping of atoms in an enzymatic reaction.

First we propose a new theoretical definition of aromaticity, and reduces its detection to finding simple cycles in a graph. For finding an automorphism group, we show our improved method with chiral information to be practical for implementation. As for enzymatic reactions, we introduce a new, efficient branch-and-bound method for maximal common subgraph problem, and apply it to the analysis of mapping of atoms within a re-

action. Each algorithm takes an advantage of molecular structures, and we confirmed their practicalness and advantage over traditional methods by the actual implementation.

In the last chapter, we introduce DNA computing, a new paradigm which is useful for the verification of computationally predicted results in laboratory works. For gene knockout, biologists have been using a vector DNA, an artificial gene consisting of various functional units. We propose DNA computing as the effective method for generating vectors in a constructive, bottom-up fashion, and demonstrate its power by basic experiments. These results show that, with the techniques developed in DNA computing, multiple patterns of vectors can be efficiently synthesized in parallel.

Acknowledgment

I would like to thank my supervisor, Prof. Toshihisa Takagi, who allowed me to pursue this independent research on metabolism in his laboratory.

This thesis could not have been possible without help of many people in different fields. Special thanks to Prof. David Morris at U. Washington for his helpful suggestions and discussion on metabolism. It is him who suggested me the modeling of bacterial metabolism. The flexible design of the framework for the metabolism, introduced in this thesis, came into my mind through the discussion with him. Also, the classification of enzymatic functions would not have been possible without his wealth of knowledge and continuous help. Thanks to Dr. Thomas Dandekar at EMBL Heidelberg, for his many useful comments on the practical side of the implemented version of the system. The strategy to find pathways in the framework has improved and will improve, through the discussion with him. Thanks to Prof. Akutsu at U. Tokyo, who helped me a lot through his thoughtful as well as critical comments on my manuscripts, especially on the theoretical aspects of chemical structure. Thanks to Prof. Hagiya, Prof. Suyama, Prof. Yokoyama and the members of Molecular Computing (moco) group for the fruitful discussion and thoughtful comments.

I am grateful to the other supervisor, Prof. Phil Green at U. Washington. I would not have understood genome informatics, especially its strategy in genome sequence analysis and the future direction of the community, if I had not been in his laboratory. The wonderful memory of Seattle will never be forgotten. Lastly, I thank my family and friends, for their warm encouragement and helpful suggestions throughout this research.

Contents

Chapter 1

Approach

It is a capital mistake to theorise before one has data. Insensibly one begins to twist facts to suit theories instead of theories to suit facts. — Sherlock Holmes

1.1 Models in Molecular Biology

From its birth as an academic field, molecular biology has been the study of models. In 1953, James Watson and Francis Crick proposed the double-helical structure of DNA in a single-page paper in *Nature* magazine [103]. Their structure not only explained the X-ray diffraction patterns of half-crystallized DNA, but also suggested the mechanism of DNA replication. The discovery was the breakthrough of the rapidly developing research area, later called molecular biology. The plausibility of the structure, however, rested on the fact that their manually constructed model could explain two experimental aspects of DNA: X-ray diffraction data and Chargaff's rule.[1] There was no proof or discussion on whether their model was the only possibility, as is seen in the unusual beginning of their paper: "We wish to suggest a structure for the salt of deoxyribose nucleic acid (D.N.A.)." Nevertheless, their feat was a remarkable success of model-building over the traditional reasoning style of natural science.

Their discovery was truly a milestone in that it revealed the mechanism of DNA replication at the same time. However, it introduced the myth that the mechanism of all living organisms would be elucidated with the same clarity and simplicity like that of the beautiful DNA structure. In other words, the research style of this new area was molded by the initial success: propose a model, and let people check its credibility. In this manner, many models were proposed only to perish. Thus, the early stage of molecular biology can be called the era of seeking universal models.

After decades of effort with advancing technologies, what researchers found was the tremendous variation of species and haphazard mechanisms. The nature revealed itself to be much more flexible and adaptive than we expected; the world is full of exceptions. For example, the well-known 'central dogma' was refuted by the discovery of reverse transcriptase. Currently, DNA is considered not a permanent repository of genetic information, but an open warehouse exposed to gene shuffling and mutation. Thus, the second stage of molecular biology became the era of reconfirming diversity.

Now we are going into the third stage, **the era of systematic reconstruction**. With advancing computational methods, we begin to understand the relation among species and the mechanism among organelles, in terms of DNA sequences. Based on this universal

[1] Chargaff showed that the amount of adenine (A) equals that of thymine (T), and the amount of cytosine (C) equals that of guanine (G), but he did not notice the base-pairing, implied from the rule.

measure, it is now possible to reconstruct models to understand biological mechanisms as systems. **What is required in this era is a good framework to re-organize our knowledge and to re-construct biological models in terms of system science.** The plural expression, '*models*', must be stressed here. The proposal by Watson and Crick, the structural model which explained the different aspects of DNA, is called a discovery. What is important in this era, however, is not a discovery, but the multiple of discoveries as a result of systematic computation. Thus, the systematic method will and should become the mainstream of model-building in biology.

The systematic computation does not deny the biological models, which have been well simulated so far. Such known models are still considered plausible; they characterize an appropriate number of proteins and genes, and properly describe their relations using differential equations, first-order logic, or if-then rules. However, those traditional methods are characterized as a top-down approach: they are tuned to output observed laboratory data in a goal-oriented manner, and their modeling methods sometimes do not have corresponding biological backing. For example, it is often unknown which cellular components achieve the effect of a differential equation. It may be a single protein, or a result of the interaction of several proteins. In fact, the frailty of existing top-down approaches becomes clear with a single question, 'What other possible models exist for achieving the same result?'

To answer this question, what is more appropriate is the systematic reconstruction in a bottom-up fashion. All logically possible results of the observed laboratory data should be traversed, and it should be checked that the model we have is the only possibility. In this way, systematic computation should cover every model which might be overlooked by human thinking. Thus, the traversal of possible models is a necessary step.

Some might question the importance of the exhaustive coverage of all possible models, especially when a traditional model is well acknowledged. Given this question, we would like to ask the definition of accuracy. For an accurate simulation using a quantitative model, all the components of the model and their interaction must be known. The reductionist approach trying to isolate these underlying parameters by laboratory work is daunting, in that it requires the purification of components and the measurement of their interaction. No existing simulation have considered all such parameters. Even if the measurement of such parameters were completed, it would be still unknown which abstraction scheme, e.g. differential equation, fuzzy logic, or boolean function, could accurately describe the biological mechanism. Each biological component may show a different behavior, and each interaction may fit a different modeling scheme. Without considering different possible models, no simulation can prove its validity.

In summary, computers are better used to search possible models, rather than to output an optimal behavior in a fixed model. The traversal of possible models, or **model search**[2], **is a necessary step in the era of systematic reconstruction.** This paper focuses on how to find possible models out of incomplete knowledge, taking bacterial metabolism as an example. In terms of metabolism, finding new metabolic pathways corresponds to building new metabolic models, on which quantitative simulation can be performed.

1.2 Bacterial Metabolism

The metabolism of a bacterium is a promising theme in computational biology, for (1) basic metabolism is conserved throughout species and is well understood; (2) the number

[2] This term was first introduced to computational biology in the talk of Hagiya and Arita at International Symposium for Human Genome Project and Computer Science (at Sanjo Kaikan, U Tokyo; Mar 1995).

of bacterial genes, therefore the size of their computational problem, is relatively small; (3) the genes for bacterial metabolism have not been identified yet, and their prediction is a main step in post genome science.

Metabolism is a network of chemical reactions, mostly catalyzed by enzymes. For example, the sugar and protein in our food are digested into water and carbon dioxide, and the obtained energy is used to produce adenosine triphosphate (ATP). On the other hand, building blocks of a cellular structure are synthesized from smaller compounds using ATP. Metabolism includes both of these processes, that is, degradation and synthesis.

Metabolism is one of the earliest biological mechanisms being investigated. The main reason is that the efficiency of metabolic enzymes can be measured from the concentration of compounds. Consequently, major pathways such as glycolysis and tricarboxylic acid (TCA) cycle have been investigated in detail for the past few decades, and some pathways are known to be conserved throughout organisms from bacteria to human. For example, most bacteria can digest glucose as their energy source, as human cells do.

Understanding bacterial metabolism by identifying involved enzymes has a variety of academic, as well as commercial applications, including understanding what is life, and accelerating the degradation of toxic compounds. Recently, researchers start re-investigating basic pathways to predict the function of bacterial genes. One well-known bacterium under such metabolic analysis is *Haemophilus influenzae* (*H.inf*), the first completely sequenced organism. Its genome was fully sequenced in 1995, but the function of one fifth its genes still remains unknown [38, 97]. To make the matter worse, its set of known function has been intentionally assigned to comprise glycolysis and other basic metabolism, using a simple-minded strategy: the assignment of gene functions by referencing the pathways of *Escherichia coli* (*E.coli*).

This top-down strategy is called 'reconstruction by reference', and is the mainstream method done by the community of genome informatics. Its obvious pitfall is that the approach implicitly unifies the metabolism of different organisms. Since each bacterium has different set of enzymes, each may have part-by-part different pathways even for basic metabolism. When more organisms are sequenced and compared, the consideration for such variant or alternative pathways will become increasingly important, more important than a quantitative simulation of the so-called consensus pathways.

Considering the historical transition of research-style stated previously, we propose that the future direction of research stream should be **to systematically search possible qualitative models, on each of which quantitative simulation can be later performed.** Note that a simulation always requires a model, which is often shown as a diagram or a textbook cartoon. Any quantitative simulation using differential equations or production rules is premised on the correctness of its qualitative model. The relation between the model-building and the simulation will become clearer in the next section.

1.3 Knowledge Representation

The reconstruction of metabolism requires a new framework, different from the one for traditional simulation. First, a quantitative aspect is unnecessary for the estimation of qualitative models. The modeling scheme of quantitative simulations is of little importance. Second, the size of the modeled metabolism is larger than that of the traditional simulations. In metabolism, the number of enzymes under consideration is well over several hundreds, while in traditional biological simulation, the number of components is up to fifty.

Traditional simulations describe biological or chemical interaction in a procedural manner and in a top-down fashion. In other words, they analytically describe *how* the modeled objects behave. Such analytical investigation proves to be powerful in digital design, but it does not in biological counterpart. In biology, we have to describe *what* we know about the modeled objects. We call it a declarative, bottom-up approach. Before introducing this new framework for metabolic reconstruction, let us explain why the procedural approach do not succeed for a large biological mechanism.

1.3.1 Procedural Approach

Continuous Models

Traditionally, enzymatic reactions are analyzed in Michaelis-Menten model, which analytically derives the relationship between substrate concentration and reaction rate at steady state from differential (rate) equations [72]. With the recent development of computers, it became possible to analyze the dynamic state of enzyme systems. The reaction-diffusion system for spatial interaction [65, 55] and the molecular mechanism of adaptation [18] are such examples.

In particular, the approach to model the time course of metabolite concentrations and fluxes led to the formalized framework, called metabolic control analysis (MCA) [27]. In MCA, the local perturbation of one reaction is related with the global properties of the entire enzyme systems. In other words, it is an approach to measure how sensitive the system is to the variations in the concentration of enzymes.

These approaches with differential equations have two major problems. One is the difficulty in the measurement of metabolite concentrations. The other, which is more fatal, is the difficulty in representing the robustness of biological mechanisms.

First, it is hard to experimentally measure the required parameters such as reaction rates. Although a differential-equation model is mathematically elegant, the realization of the perturbation required for the analysis is impossible. For this reason, the analysis of the large-scale enzymatic systems is still impossible as is stated in the page 9 of the book by Hayashi and Sakamoto [43]: '...the actual situation at present is that a procedure exactly fitted to enzyme dynamics is not yet available. A serious future problem may arise concerning the proper treatment of large-scale systems in the analysis of biochemical networks'.

To overcome the difficulty in measuring the reaction parameters, several researchers including the author tried to computationally predict the parameters of protein interaction, using simulated annealing or genetic algorithm [80, 8]. This approach, however, is faced with additional problems: (1) input data are not sufficient enough for the quantitative modeling; (2) a search for 'good' parameters is computationally too expensive; (3) the model does not represent everything that should be.

The other hardship is the inability to show the robustness of biological mechanisms. In a simple, continuous model, the deviation of the parameters from their fine-tuned values results in the deviation of adapted or optimal results. However, many biological mechanisms show exact adaptation even in the existence of perturbations. The representation of this robustness necessitates a more complex model, like a two-state model by Barkai and Leibler [13], which introduces more conceptual and practical difficulties. Even more fatal is the difficulty of isolating the sub-mechanisms which can be analyzed separately. In a model with differential equations, therefore, the analysis of a local mechanism may have to consider interactions between different parts in the cell.

If robust properties do not depend on exact values of the biochemical parameters, and therefore, if it is possible to decompose the mechanism to sub-mechanisms, then it should be possible to extract qualitative principles underlying the mechanism. It must also be noted that most laboratory data in biology are qualitative.

A qualitative model has a much smaller search space of its biochemical parameters. In order to deal with such qualitative aspects, we once proposed a threshold model, in which the interaction of proteins is described with if-then rules with threshold conditions [9]. It can be called a discrete (but still procedural) model.

Discrete Models

Biological interaction often shows discrete aspects such as genetic switch or protein binding [17]. Such mechanism is better represented with Boolean function rather than with continuous differential equation. In fact, the mixture of Boolean functions and the differential equations was shown to accurately model the fate of λ phage virus by McAdams and Shapiro [62]. In their work, each Boolean function is represented as a if-then rule, conditioned with a certain threshold to suit the overall quantitative framework. We applied this threshold model to the protein interaction in the embryogenesis of fruitfly, and showed that the stripe patterns of segmentation proteins can be reproduced using a discrete model [8].

The discrete approach not only simplifies the model itself, but reduces the search space of required parameters. This makes the computational search of 'good' parameters easier, but still some difficulties remain. First, it must be manually determined whether to use discrete or continuous representation for each protein or gene, and the verification of this judgment ends up in the quantification of metabolite concentration in the continuous model. This problem is unavoidable, as long as the model contains the continuous part. Second, it is possible that the model does not represent everything that should be. It is hard to determine which parameters are good or bad, when the model may obtain additional new components.

Too much reduction of biological models is also problematic. The extreme case is a knowledge-based model, comprised only of if-then rules [66, 93]. In the model, an original mechanism is reduced to a set of discrete switches, and its simulation result is always the reproduction of expert knowledge. Since no expert know the molecular detail of biological mechanisms, no knowledge-based model have succeeded in predicting new biological phenomena, or generating hypotheses.

In summary, the discrete description of a biological mechanism is advantageous over the continuous approach in that it offers a simpler model with less search space for its parameters. However, less search space also indicates less flexibility, and the developer of a discrete model must know more about the behavior of the modeled mechanism, in order to represent it under the constrained description. Inevitably, the developer tends to simulate only the known part of the biological mechanism, and this is why biological simulation has been slighted by laboratory workers as the mere reproduction of textbook contents.

Such hardships in biological modeling originate in the hasty application of procedural knowledge representation. Here, let us turn to the new framework appropriate for metabolic reconstruction, i.e. declarative, bottom-up knowledge representation.[3]

1.3.2 Declarative Approach

Even when it is not known how the modeled objects behave, it is possible to write *what is there*. This is the declarative approach. In this approach, neither unknown parameters nor hypothetical components need to be introduced. It also suits the incremental reorganization of knowledge: more knowledge will automatically produce more accurate

[3] In fact, these two ideas are not disjoint. Horn clauses have both procedural and declarative interpretations, for example.

	continuous *(requiring fine adjustment)*	discrete *(capturing robustness)*
procedural	differential equations *mathematically elegant, but lacking data.*	if-then rules *easy to understand, but no new prediction.*
declarative	real life(?)	graph approach *See the main text.*

Table 1.1: Summary of the Modeling Approach

results.[4]

The crucial issue is what abstraction scheme is suitable for modeling biological interactions. For example, a chemical compound in a cell may be treated either as one molecular unit or as a set of atoms. A protein may be treated either as one molecule or as a sequence of amino acids. One thing is clear, however, in any of these abstraction scheme. The biological mechanism is reduced to a set of molecular units and their interaction (or relation).

This paper proposes the graph representation of a biological mechanism, and shows that metabolism is reduced to this representation. Metabolism is, however, only an example. In general, **computational reconstruction of biological mechanisms can use the declarative knowledge representation, in order to search biologically plausible models.** In other words, the declarative representation is useful in presenting various views of the same biological interaction.

Intuitively, a cell is considered a bag of solution in which floating units interact with one another according to catalytic (enzymatic) rules. Thus, the network of reactions is considered a graph in which units and reactions correspond to nodes and edges, respectively. Careful reader might notice that the number of nodes and edges may be infinite, because of polymerizing reactions. For example, a DNA polymerase may generate an infinitely long chain of DNA. If such details are ignored, then what we usually call metabolic pathways, signaling pathways, or protein interaction is only a subset of the entire graph network. In a reverse way, any subset (or subgraph) becomes a certain model of the mechanism.

In terms of graph theory, a graph G represents the target mechanism S. Its restriction to a subgraph $H \subseteq G$ is a model of S, or its view. Usually H satisfies some requirements, such as H contains certain nodes and edges. The enumeration of such Hs in G, then, corresponds to the exhaustive search of possible models of S. The next section discusses the pros and cons of this graph approach.

1.4 Advantages and Disadvantages

The graph approach is to (1) declaratively describe the biological mechanism as a graph consisting of molecular units and their interaction; (2) specify a pattern graph to extract its images out of the entire graph.

The advantage of this approach is the possibility of traversing models. In the graph representation, one subgraph is one model. For example, glycolysis a linear path from glucose to pyruvate, and TCA cycle is a cycle containing citrate, succinate, and malonate.

[4] To be precise, this statement assumes the knowledge to be monotonic. When the negative knowledge is also integrated, the framework of non-monotonic reasoning is necessary.

Such restriction or constraint can be specified as a pattern graph. The pattern graph extracts only such subgraphs that satisfy the constraint out of the entire graph. Note that the pattern also corresponds to a view of a user, so that she/he can focus on any local subsystem, including a hypothetical one, by providing a pattern graph.

The disadvantage is that the graph representation does not suit the quantitative analyses. As we have explained, however, the qualitative models in a larger scale are more promising than the traditional quantitative simulation for finding new pathways, for predicting functions of bacterial genes, and for understanding life in general.

The graph approach is especially useful for predicting the function of genes. Given a bacterium as a modeling target, the initial graph is constructed from the components known to actually exist in the organism. To these initial data, additional nodes and edges may be supplemented, e.g. if they are found in another related bacterium. These additional components serve as hypotheses for the target bacterium. More precisely, if the additional nodes and edges frequently appear in the images of a certain pattern graph, then such nodes and edges, corresponding to molecular units and enzymatic reactions, are likely to exist in the bacterium. In other words, the appearance of hypothetical graph components corresponds to the prediction of functions for yet unknown genes in the target bacterium.

There exist more advantages:

- The output is logically correct.
  The computational result is not a collection of biased knowledge of experts in biology. A user can freely modify data and query, and can check what is logically derived from the input data. In exchange, the result may be biologically incorrect, if input data are incorrect.

- The scale is large.
  Because of the graph representation, the number of components can be large enough to treat the whole metabolism (the order of thousands). This scale is much larger than a traditional simulation, which usually treat tens or hundreds of components. In exchange, quantitative aspects are sacrificed.

- It can generate hypotheses.
  If the strategy of extracting subgraphs is flexible enough, it is possible to generate various hypotheses which fully utilize the user-input information.

On the other hand, obstacles for modeling metabolism are:

- How do we declaratively describe cell structures?
  Certain compounds prevalently appear in a cell, while others are localized. Certain reactions proceed bidirectionally, while others do one-way, depending on their loci in a cell. Such behavior has not been flexibly described in the traditional simulations.

- How do we achieve an efficient and flexible search in a large graph?
  Finding an embedding of a pattern graph in a large graph is a difficult problem. This problem can be considered a variant of search problem, and there is a trade-off between the accuracy of the search result and its efficiency.

- How do we represent chemical compounds?
  We can browse compounds and reactions in commercial chemical databases, but such data are usually graphical (e.g. GIF) or text-based (e.g. adjacency matrix). The modeling of metabolism by connecting single reactions requires a data structure on molecular structures, specifically on their graph representation. However, there

is no practical algorithm for finding the normal form of molecular graphs, or for computing the common substructures of multiple graphs.

- How do we represent enzymatic reactions?
  A reaction formula in the database tells us only the names of compounds in the reaction. For the modeling of the network, however, we need to know which atom in one compound is transferred to which atom in another compound. The problem of computing this atomic mapping is harder than the subgraph isomorphism problem, and its solution is necessary.

Paper Outline

The key idea throughout this thesis is the systematic reconstruction of biological mechanisms by model search. We show an constructive search strategy, in contrast to the analysis of an already hypothesized model in a traditional way.

The first part covers the computational reconstruction of metabolism. A new framework for describing biological mechanisms is introduced, and questions in metabolism are reduced to problems in graph theory.

The latter part covers the laboratory techniques to achieve the constructive understanding of biological mechanisms. It is the solution to the verification of computational predictions in laboratory works. The following chapters are organized as under.

In Chapter 2, the new framework is introduced. It is called a cookie-cut framework, and represents the whole mechanism qualitatively and declaratively. In the framework, a user can search possible models, each of which is a focused view of the original mechanism, serving as a diagram of quantitative simulation. The model search is realized by a variant of shortest path algorithm.

In Chapter 3, three fundamental algorithms are shown, which are used to digitize chemical compounds and enzymatic reactions at the atomic level. The knowledge representation at the atomic level is necessary for modeling metabolism. In our approach, compounds are represented as graphs, and enzymatic reactions are represented as a mapping between graph nodes.

In Chapter 4, two important topics in molecular biology are introduced as the application of the introduced framework. More precisely, the framework is shown to be effective in functional prediction, and in finding signaling pathways.

In Chapter 5, DNA computing is introduced for the verification of computational predictions. Laboratory experiments require a DNA fragment called a vector. With DNA computing, various vectors can be systematically generated in parallel. Traditionally, biologists design vectors to confirm a certain hypothesis. With DNA computing, this top-down analysis can be replaced with more constructive approach; we can start asking which vectors show the same or different outcome. Thus, DNA computing offers the constructive investigation of biological mechanisms in laboratory works.

Since the thesis covers the broad area of research, each chapter has its overview and summary, and there is no separate chapter for conclusion or discussion. This chapter offers an overview of the main thesis.

Notes

Readers who are interested in the history of molecular biology are referred to the book by Judson [51]. It gives a fair view on the discovery event. Much argument has been discussed on the controversial memoir by Watson [102], whose biased opinion was countered by other researchers and historians [87, 28]. Many on-line resources are also available for the history of molecular biology. For general information, we recommend Access Excellence project by Genentech (`http://www.gene.com/ae/`).

Information on bacteria in this paper is mainly taken from the textbook by Moat and Foster [68]. Throughout this paper, the author tried to avoid details in biology and chemistry, but could not avoid mentioning some basic notions such as chirality. Such terms are explained at the beginning of each chapter and section. For more basic knowledge and the explanation, readers are referred to textbooks on biochemistry and microbiology [31, 77].

The modeling of enzymatic reactions has been pursued by many groups [72, 27, 90, 57, 41, 73]. Followings are some www resources for analyzing metabolism.

```
http://gepasi.dbs.aber.ac.uk/metab/mca_home.htm       http://www.genome.ad.jp/kegg
http://www.labmed.umn.edu/umbbd/index.html             http://wit.mcs.anl.gov/WIT2
```

Chapter 2

Computerizing Metabolism

When you are face to face with a difficulty,
you are up against a discovery.
— Lord Kelvin (1824–1907) Physicist

2.1 Overview

The framework introduced in this chapter is a graph representation of biological mechanisms, which is used to search models for simulation. The overall process consists of two steps: a generation step of the representation graph, and an extraction step of models with a pattern graph.

The two-step process is called the 'cookie-cut framework'. Its name comes from the analogy between the two steps and the steps of making cookies: spread the dough, and then cut out cookies with the cutter of a particular shape. The process has been repeated by biologists in the manual reconstruction of metabolism. An expert first ruminates on the possible explanations for the result of laboratory experiments (generation step), and then looks for an appropriate model by crossing out unlikely situations (extraction step).

In the generation step, all the known interaction in the given biological mechanism is transformed into a graph structure. Any subgraph of the transformed graph corresponds to a subsystem of the original mechanism, and serves as a model for quantitative simulation. When nodes and edges are further supplemented to the transformed graph, they serve as hypotheses in the extracted models.

In the extraction step, a set of subgraphs is computed with a user-specified constraint. The constraint is given as a pattern graph, and each chosen graph is its embedding, i.e. a homeomorphic image of the pattern graph.

The cookie-cut framework comprises five basic components: compartments, molecular units, interaction rules, rule weights, and a pattern graph.

The generation step transforms a biological mechanism into its graph representation using the first three components. They declaratively define the target mechanism, and the notation looks like Datalog enhanced with variable scopes.

- *Compartments* define a hierarchical space. Each compartment is always associated with a name such as 'nucleus' or 'cytosol'. This is the unique difference of the framework from conventional programming languages.

- A *molecular unit* is located inside a compartment, and is always referenced with the compartment name.

- *Interaction rules* define the relation of molecular units. There are two types of interactions: specific rules for experimentally verified relation, and generic rules for hypothetical relation. Their difference will be introduced later in detail. Functionally, rules are not associated with the compartments.

The extraction step uses the remaining two components: rule weights and a pattern graph.

- The homeomorphic images of the *pattern graph* is searched within the transformed graph, and they are output in the smallest order according to the *rule weights*.

Unfortunately, the flexibility in the extraction step and the computational intractability are two sides of a coin; if the extraction step intends to solve the fixed subgraph homeomorphism problem, the cookie-cut framework is not free from the shackle of NP-completeness. In the actual implementation, therefore, instead of finding an theoretically intuitive embedding such as node-disjoint or edge-disjoint, the framework composes the image as a combination of shortest paths between nodes in the transformed graph. The decomposition to shortest paths does not guarantee the restriction of the original pattern graph, but this flaw is superficial in a biological application. It will be shown that biological results need not satisfy the theoretical criteria such as node-disjoint or edge-disjoint.

AMR (Automated Metabolic Reconstruction) system is the implemented version of the framework, designed for searching hypothetical metabolism in bacteria. Note that AMR system does not 'simulate' metabolism. It simply computes a possible pathways among compounds. The major difference from the 'simulation' is that AMR qualitatively models metabolism at the atomic level. In other words, AMR reproduces the tracer experiment in biochemistry.

The reproduction of the tracer experiment is quite important in the post genome research, whose major theme is the functional analysis of already sequenced genes. Such an unassigned function can be predicted by comparing a computational tracing result with an experimentally observed tracing result.

This chapter also introduces an original, structure-oriented classification of major metabolic enzymes. To output hypothetical pathways, AMR requires a list of hypothetical enzymatic functions, which is used when the collection of already known function does not comprise a vital metabolism.

The chapter is organized as follows. In Section 2.2, the cookie-cut framework is introduced. In Section 2.3, the actual transformation of metabolism into the cookie-cut framework is shown. The important point is that molecular structures and enzymatic reactions are modeled at the atomic level. In Section 2.3, we show an original classification of metabolic enzymes.

2.2 Cookie-cut Framework

2.2.1 General Description

Origin of two steps

The cookie-cut framework is a general scheme to describe biological mechanisms. The overall process consists of two steps: a generation step and an extraction step. The generation step is regarded as the construction of all possible interaction of molecular units in the target mechanism, without considering its quantitative aspects. The extraction step outputs a model of the generated interaction, according to the constraint given by a user. It is a step of searching models too.

Since quantitative aspects are ignored, it may seem that the generation step considers too many pathways including biologically trivial ones. This is not correct: it is impossible to judge whether a certain pathway is biologically meaningless or not. Even when a certain interaction proceeds less likely, it does not mean that the interaction may be ignored in any case. Note that the regulation among molecules or the concentration of molecules only *bias* the behavior of the mechanism. In a qualitative framework, therefore, all possible interactions are considered to proceed. It is the most helpful way to understand the overall mechanism too.

When all possible interactions are considered, the resulting interaction network often becomes too complex to grasp in its entirety. For this reason, the extraction step becomes necessary to see what is going on locally. This local, qualitative mechanism can be called a model. Therefore, the extraction step is also a step of searching models.

In another perspective, the step extracts a focused view of some local mechanism for easier understanding. In this sense, the two steps correspond to a complex natural phenomenon and its observation, respectively. Here, an observation is an activity to focus a particular sub-structure of the entire phenomenon.

Generation Step

The generation step is characterized by the three components: the structured *compartments*, *molecular units* inside them, and *interaction rules*.

A biological mechanism is essentially a collection of molecular units (e.g. chemical compounds, proteins, or DNAs) localized in compartments (e.g. nuclei, mitochondria, or cells). Molecular units freely interact with one another inside each compartment, or migrate from one compartment to another. Interaction rules represent such arrangements in the mechanism.

A compartment is denoted $\{p, q, \ldots\}$, in which p and q are molecular units. A compartment is also considered one molecular unit, and the operator $\{\cdot\}$ may be nested. When nested, molecular units in an outside compartment are allowed to enter its inner compartments. Moving out of compartments are not allowed. Compartments are, in a sense, similar to the scope of variables in a programming language.

Interaction rules are of the form

$$m_1,\ m_2, \cdots\ m_k\ \rightarrow\ m'_1,\ m'_2, \cdots\ m'_l$$

, in which m_i and m'_j are molecular units. The molecular units in the left-hand side are called *inputs*, and those on the right are *outputs*. Examples are the phosphorylation of proteins, or the generation of chemical compounds, and so on. There are two types of interactions: natural laws and exceptions.

- A *generic rule* is potentially applicable to an infinite number of molecular units, and corresponds to a natural law.

- A *specific rule* is applicable to a finite set of molecular units, and corresponds to an exception.

Its definition in a programming-language style is that a generic rule contains variables, unifiable with a class of molecular units. A generic rule is more general than a specific rule.[1]

[1] Note that there are very few universal laws in biology, in contrast to physics. In this sense, a generic rule may be better regarded as a hypothesis derived by induction, and not a law. Specific rules then correspond to experimentally verified facts.

In the generation step, given a set of molecular units and a set of interaction rules, the rules are repeatedly applied to molecular units in each compartment. There is no order or control structure in the order of application, and all possible transformation occurs. Moreover, no molecular units are consumed by this application.

Let us give a simple example. Assume two nested compartments with a molecular unit M, and two rules (Fig 2.1): (1) one rule in the 'global' compartment transforms the unit M to N; (2) the other one in the 'local' concatenates the units. Let $M[i]$ denote the concatenation of i copies of M. (Therefore, M is a shorthand notation of $M[1]$.)

```
{ "global"
  M
   { "local"
      M[i], M[j] → M[i + j] /* This rule is generic. */
   }
  M → N /* This rule is specific. */
}
```

Figure 2.1: Sample framework (before the application of rules)

```
{ "global"
  M, N
   { "local"
      M[i], M[j] → M[i + j] /* This rule is generic. */
      N, M[1], M[2], M[3], M[4]...
   }
  M → N /* This rule is specific. */
}
```

Figure 2.2: Sample framework (after the application of rules)

After the unlimited application of the generic rule, the concatenation of an arbitrary number of Ms will be generated in the local compartment. N is generated by the specific rule in the global compartment, and it can freely drift into the local part. Fig 2.2 shows the transformation of compartments by the two rules.

In terms of graph theory, molecular units and interaction rules are viewed as nodes and edges, respectively. The generation step generates a directed hypergraph[2] of possibly infinite nodes (Fig 2.3).

Extraction Step

A user specifies a *pattern graph* to extract a particular set of interactions out of the entire hypergraph. The query is denoted *?(pattern-graph)*, and the given pattern selects its embedded images in the generated hypergraph. In the above example, we can specify a path-like constraint:

[2] A *hypergraph* is a set of nodes and a set of edges, and each edge connects a set of nodes.

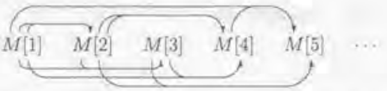
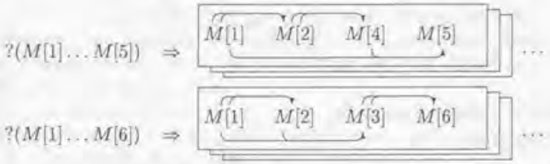


Figure 2.3: The hypergraph generated by $M[i] + M[j] \rightarrow M[i+j]$.



Any subgraph containing the embedding of the pattern graph is a result of the extraction. In general, the result becomes a set of (possibly infinite) subgraphs, and therefore a measure for their prioritization is necessary to output them.

When each selected subgraph is considered a model to explain the given pattern graph, natural criteria for the prioritization are:

Criterion I. Selected subgraphs are minimal.

Criterion II. Selected subgraphs use as many specific rules as possible.

Criterion I reflects Occam's razor: a simpler model is a better model. In order to explain why a particular unit is generated, we choose the explanation with the least number of interactions, i.e. the subgraph of least number of edges. Criterion II reflects another principle: exceptions override laws. In other words, specific rules are more preferred than generic rules, when both can achieve the same result. It must be stressed that the selected results are not necessarily biologically optimal. The purpose of the cookie-cut framework is to provide a flexible environment for listing possible models, or searching models, satisfying the given restriction.

The cookie-cut framework is truly a declarative representation of the target mechanism, and there is no notion of time or control flow. Its difference from a traditional simulation comes from the notion of model search; a user can *search* a model satisfying the given constraint (a pattern graph), by considering which hypotheses are more plausible.

No biological simulation has focused on the importance of hypotheses and model search as this framework; a traditional simulation has been concerned with 'what' can be achieved in the modeled system by describing 'how' each component works. The cookie-cut framework aims its reverse process. **Its motivation is finding 'how' the result comes into effect by describing 'what' is known or experimentally observed.**

2.2.2 Homeomorphism of Pattern Graph

This section discusses the algorithm to extract subgraphs containing the given pattern graph. The realization of the extraction step is difficult, because finding the proper embedding of a pattern graph in a larger graph is formalized as the fixed SHP (subgraph homeomorphism problem), which is NP-complete for most cases.

Before going into SHP issue, let us revisit the two criteria for selecting subgraphs. When a interaction rule may have multiple inputs, Criterion I becomes ambiguous. For example, we want to avoid the following 'anything is available' result, although it certainly seems to be a minimal answer.

$$?(M[1]\dots M[6]) \quad \Rightarrow \quad \boxed{\underbrace{M[1] \quad M[5] \quad M[6]}}$$

One solution to avoid this consequence is to minimize the number of required or necessary molecular units in the selected subgraph. In the above case, $M[6]$ is generated only from $M[1]$. However, this minimization is of theoretical interest only. Rather, we discuss only the case in which the number of rule input is one. As we will show later, metabolism can be described under this restriction.

The restriction reduces the complexity of a hypergraph to that of a graph. In the following, an interaction rule is of the form $m \to m'_1, \dots m'_l$, and is interpreted as l directed graph edges from m to each of $m'_1, \dots m'_l$.

Definitions

Let $H(V_H, E_H)$ and $G(V_G, E_G)$ be two graphs. A *subgraph homeomorphism* between H and G is a pair of one-to-one mapping (f, g) such that $f : V_H \to V_G$ and $g : E_H \to$ { set of simple paths in G}. H is called a *pattern graph*, and G is the *input graph*. If the set of paths $\{f(v) \mid v \in V_H\}$ are disjoint, the homeomorphism is a *node-disjoint homeomorphism*. If the corresponding edge-set is disjoint, the homeomorphism is an *edge-disjoint homeomorphism*. The *minimal* subgraph of G selected by H is a homeomorphic subgraph of G, minimizing its number of edges.

Hardness

In the cookie-cut framework, the generation step constructs the input graph from the user input data. It is not necessarily a simple graph. The extraction step chooses the minimal subgraphs, homeomorphic to a user-specified pattern graph. The problem is called fixed SHP, because the nodes in the pattern graph have specific positions in the input graph.

Unfortunately, the fixed SHP for directed graph is NP-complete for most pattern graphs, if the homeomorphism is assumed to be node-disjoint. The edge-disjoint homeomorphism has the same complexity.[3]

Theorem 2.2.1 (Fortune, Hopcroft, and Wyllie [39]) *SHP for any fixed pattern P is NP-complete if P may contain: (1) two disjoint edges, one or both of which may be a loop, (2) a path of two arcs visiting three distinct nodes, or (3) a cycle of length two.*

Theoretically related work restricts either the input graph to an appropriate special class [15, 24], or the pattern graph to be really plain [48]. Such a restriction is not suitable for our purpose, because both an input and a pattern graph do not confine themselves in a simple class in a biological application.

Heuristics

In a biological application, the embedding of a pattern graph does not need to be node-disjoint or edge-disjoint. For example, Fig 2.4 shows a frequently observed pattern in biological pathways connecting compound s and t. The conjugation of cycles is not useless

[3] The problem is still open for many cases in undirected graph. The problem with the pattern of two disjoint edges, or of a cycle of length three is solved in polynomial time [92, 56].

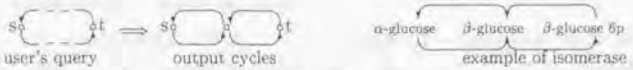


Figure 2.4: Some Patterns of Bio-pathway

in a biological interpretation, because the cycles connecting s and t can have a flow of molecular units.

A typical conjugate cycle is performed by an enzyme called isomerase. Isomerase usually performs reversible catalysis such as interchanging α-glucose and β-glucose, whose equilibrium ratio is approximated by a Michaelis-Menten type equation. When β-glucose is also under equilibrium with β-glucose 6-phosphate, the pathway from α-glucose to β-glucose 6-phosphate is regarded as a conjugate cycle. Thus, in biological pathways, what is important is not a net flow, but the amount of dynamic circulation. Therefore, the embedding of a pattern graph does not need to be node-disjoint.

As a heuristic approach, the embedding of the pattern graph is computed as a combination of shortest paths as under. Let $G(V_G, E_G)$ and $H(V_H, E_H)$ be the input graph and the fixed pattern graph, respectively.

Heuristic Pattern Search

1. For each edge $(s,t) \in E_H$, compute the shortest path $\hat{p}_{st}$ in G.
2. Re-connect all shortest paths $\{\hat{p}_{st} | (s,t) \in E_H\}$ as in H.

The advantage of adopting this heuristics is the efficient enumeration of matching subgraphs. With the k shortest paths algorithm by Eppstein [33], it is possible to generate the k matching subgraphs, minimizing the number of edges. (The embedding may be not node-disjoint or edge-disjoint, because each shortest path is independently computed.)

Rules: specific vs. generic

The rules may be prioritized by setting a weight (or distance) of edges in the shortest-path computation. Criterion II implies that a specific rule, $m \to m'$, should have a strictly smaller weight than any generic rule connecting m with m'. In general, however, the variety of weights depends on each application of the cookie-cut model.

Theoretically, the shortest path algorithm can manage any weight forming a monoid associated with edges, and various conditions can be transformed to this property. For example, the same algorithm can be applied to find a path minimizing the number of generic rules, or minimizing the use of a user-specified group of rules, and so on. The same algorithm can constrain pathways as follows.

Path Patterns:

- A path p_{st} with less than n generic rules.
- A path p_{st} such that no two generic rules appear adjacently.
- A path p_{st} such that $p_{st} \cap S \neq \phi$ for the given $S = \{s_1, s_2, \ldots\}$.

2.2.3 Presenting a graphical view

The k chosen subgraphs for the given pattern graph often share their main structure. Since the subgraphs are arranged in the smallest order only for a theoretical reason, it is understandable to merge the output subgraphs, so that the user can grasp the neighborhood of the embedded images of the pattern graph. That is, the merged graph becomes a graph containing at least k different embeddings of the pattern graph. It is called a *view*. The basic algorithm generating the view owes much to k shortest paths algorithm by Eppstein [33].

As the embedding of a pattern graph was reduced to a set of the shortest path, the k smallest embeddings of the pattern graph is reduced to a set of k shortest paths. We shall discuss on the set of k shortest paths from s to t in a digraph G.

The *length* of an edge e is denoted $\ell(e)$. The endpoints of an edge e are denoted $tail(e)$ and $head(e)$; the edge is directed from $tail(e)$ to $head(e)$. The shortest *distance* from node s to node t is denoted $d(s,t)$. Let T be a single-destination shortest path tree of G with t as its destination. Given an edge e in G, define $\delta(e) = \ell(e) + d(head(e), t) - d(tail(e), t)$. This is the measure of how much distance is lost by sidetracking the edge e.

As Eppstein pointed out [33], any path p from s to t is uniquely determined by a subsequence $sidetracks(p)$ of its edges in $G-T$. Therefore, the total distance of k shortest paths $P = \{p_1,\ p_2, \dots p_k\}$ is determined sorely on their edges in $G - T$:

$$\sum_{p \in P} \ell(p) = k \cdot d(s,t) + \sum_{e \in P} \delta(e).$$

We define ϵ-neighbor view of the shortest path from s to t as a set of shortest paths $P = \{p_1,\ p_2, \dots p_k\}$ such that $\sum_{p\in P} \ell(p) \leq \epsilon$. This view can be found with k shortest paths algorithm with a slight modification.

Note that the lengths of edges in our application are almost uniform: all specific rules have the same distance, and all generic rules may do so too. Therefore, there may be multiple set of edges in $G - T$ which achieve the same ϵ neighbor. In order to prioritize those set of edges, we can consider the connectedness of the paths. However, we do not investigate this problem here, because the biological interpretation of connectedness is still unknown.

2.2.4 Notations

Tools in computational biology often look for several candidate results, and their flexibility is important. Note that biologists' insight often rejects strictly mathematical formalization, and experts usually seek the most plausible result in trial-end-error in the neighborhood of a mathematically optimal answer.

In order to search a biologically plausible result by modifying the original output, two capabilities are required: (1) the output can be incrementally *refined* by a user; (2) the output can be *replaced* part by part. For these operations, graph is an ideal representation in specifying the intended constraint.

The most intuitive and natural way to deal with graphs is to use a graphical user interface. From the perspective of the formal description of biological mechanisms, however, the notation for terminals is also important. This section introduces a notation of the framework, which will be used in the explanation of metabolism later. In BNF, the *slanted* or `typewriter` face denotes terminal symbols.

```
    { "foo"
        molecule_A; molecule_B;
    }
    molecule_A "foo" -> molecule_B "foo";
⟺  /* equivalent description */
    { "foo"
        molecule_A; molecule_B;
        molecule_A -> molecule_B;
    }
```

Figure 2.5: Shorthand for writing rules

Pattern

```
   pattern ::= node rings
             | pattern connection node rings
             | pattern ( pattern )
connection ::= > | :> | :( condition )>
             | < | <: | <( condition ):
      node ::= molecule
 condition ::= node | condition & node | condition | node
     rings ::= empty | integer | connection integer
```

The definition of a pattern graph resembles that of SMILES, so that a pattern can specify an arbitrary graph shape. The connective `>` denotes adjacency. Pattern `s>t` selects the graphs in which molecular units s and t appear with a directed edge (s, t). Pattern `s:>t` selects the graphs with a path p_{st}, and the pattern `:( S )>` denotes a path with condition S. Pattern `s<t`, `s<:t`, and `s <( S ): t` have the same interpretation in the reverse direction. A cycle is described by attaching an integer label to the molecular units to be connected. The pattern-matched output is returned in the same format, without using `:>` or `:( S )>`.

Rules and Compartments

```
       rule ::= hand-side operator hand-side ;
   operator ::= -> | <- | =
  hand-side ::= molecule | molecule string
compartment ::= { string comps }
      comps ::= empty | comps molecule ;
              | comps rule | comps compartment
```

A molecular unit is always associated with some compartment. Each compartment is associated with its name (in string), and can be defined hierarchically. A unit in a global compartment is visible from local compartments as in many programming languages.

A rule can map a molecular unit in one compartment to another. Rule `s -> t` denotes an edge from unit `s` to `t`. Operator `<-` is its reverse, and `s = t` equals both `s -> t` and `s <- t`. Rules are not associated with compartments, but for convenience, they can be written inside a compartment as in Fig 2.5.

2.3 Describing Metabolism

2.3.1 General Description

In the cookie-cut framework, each applied problem is characterized by the five components. We shall describe the application of the framework to bacterial metabolism. It is called the metabolic framework. For clarity of presentation, we call the component of the framework as *molecular units* and chemical substances in metabolism as *compounds*.

Molecular units

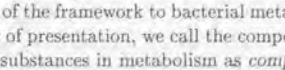
In the internal computation, each atomic symbol in SMILES corresponds to a molecular unit in the metabolic framework. That is, SMILES is a compact representation of the set of molecular units.

EXAMPLES.

name	notation	interpretation
L-alanine	`N[C@H](C(=O)O)C`	Set of 6 atoms C_3NO_2
L-serine	`N[C@H](C(=O)O)CO`	Set of 7 atoms C_3NO_3
hydroxypyruvate	`OCC(=O)C(=O)O`	Set of 7 atoms C_3O_4
pyruvate	`CC(=O)C(=O)O`	Set of 6 atoms C_3O_3

Compounds written in SMILES are stored as molecular graphs. Each input SMILES undergoes (1) the detection of aromatic cycles, and (2) the normalization of the structure.

Some compounds frequently appear in metabolism. They are called *coenzymes*, and include most of vitamins, metal ions, and coenzyme A. They are usually regarded as auxiliary compounds in a reaction, as is seen in the conventional notation:



in which NAD$^+$ and NADH are coenzymes. Atoms within them are usually not considered in the analysis of pathways, because the set of coenzymes in metabolism are common among biological species, and because coenzymes are usually prevalent and recycled in a cell.

To reflect this observation, molecular units in global compartments are considered coenzymes in the metabolic framework, i.e. they are ignored in the search of pathways. Typically, compounds like nicotinamide (coenzyme), water, and carbon dioxide (inorganic compounds) are defined in the global compartment. When a user wants to integrate such coenzymes in the search, they are defined in a local compartment. Thus, a user can arbitrarily specify coenzymes in the metabolic framework. For this reason, the structure of compartments does not correspond one-to-one to the actual cellular structure (Fig 2.6).

Specific Rules

Reaction formulas of enzymes correspond to specific rules. Each formula represents a set of edges, each mapping a single atom in the left-hand side to another atom in the right. The mapping is computed by the algorithm in Section 3.6. When the compound is symmetric, the mapping to the symmetric positions are also supplemented.

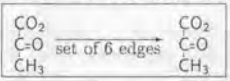
EXAMPLES.

```
{ "cell"
    /* Define H2O, NH3, and other prevalent compounds here. */
    { "cytosol"
        /* Define localized compounds here. */
    }
    { "mitochondrion"
        /* Define localized compounds here. */
    }
}
```

Figure 2.6: Example definition of a eukaryotic cell. Note that prokaryotic cell does not have a mitochondrion.

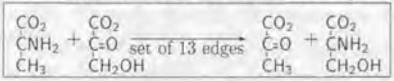
- pyruvate carrier

```
CC(=O)C(=O)O "cytosol" -> CC(=O)C(=O)O "mitochondrion"
```



- alanine-hydroxypyruvate transaminase

```
N[C@H](C(=O)O)C, OCC(=O)C(=O)O "cytosol"
    -> N[C@H](C(=O)O)CO, CC(=O)C(=O)O "cytosol"
```



Generic Rules

Reactions by hypothetical enzymes correspond to generic rules. The rules for metabolic framework are explained in detail in a later section.

Pattern

An arbitrary graph is accepted as a pattern. Since each atom in compounds corresponds to a molecular unit in the metabolic framework, nodes in the pattern graph should be atoms of compounds. That is, each node in a pattern graph should be referred as 'atom i of compound A', and the designation of a certain pathway looks like 'the pathway from atom i of compound A to atom j of compound B'.

On the other hand, a convention in referring a metabolic pathway is 'the pathway from compound A to compound B', whose intended meaning is that the 'major' molecular component (sometimes referred as *moiety*) in A goes to B. For example, the major component in Urea cycle is a nitrogen atom, while in TCA cycle, it is carbon atoms. In most cases, carbon atoms are under focus.

In the metabolic framework, the leftmost carbon atom in SMILES format of each compound is defined to be the default position of moiety. The query `molecule_A :> molecule_B` is interpreted as the paths from the leftmost carbon atom of `molecule_A` to any carbon atom in `molecule_B`. If `molecule_B` is used in the following search of paths, the leftmost atom in `molecule_B` among atoms mapped from `molecule_A` will be used as the source of paths. Thus, the traced atom depends on SMILES by default. It is also possible for a user to explicitly specify the atomic position under focus. [4]

EXAMPLES.

- glycolysis (glucose→pyruvate)

```
?( C(O)[C@@H](O)[C@@H](O)[C@H](O)[C@@H](O)C=O "cytosol"
      :> CC(=O)C(=O)O "cytosol" )
```

- TCA cycle (citrate→ α-ketoglutarate→fumarate →citrate)

```
?( O=C(O)CC(O)(C(=O)O)CC(=O)O 1 :> O=C(O)CCC(=O)C(=O)O
      :> O=C(O)C=CC(=O)O :>1 )
```

Execution of the above query (TCA)

```
O=C(O)CC(O)(C(=O)O)CC(=O)O 1 :> O=C(O)CCC(=O)C(=O)O
  ⇑ This carbon is traced first.  ↑ destinations↑
                                  ⇑ This carbon is traced next.
                                  :> O=C(O)C=CC(=O)O :>1
                                       ↑      ↑ destinations
```

Not all the atomic elements can be traced under catalysis. In particular, the fate of an oxygen and a hydrogen atom is often ambiguous, because of the reaction with water. For example, an oxygenase incorporates a molecular oxygen into some position of the input compound; an isomerase changes the configuration of chiral carbons. Under these transformations, it is ambiguous whether some atoms are exchanged with external water molecules, or phosphates. For this reason, only carbon and nitrogen atoms can be traced in the metabolic framework. They are also the most important atoms whose fixation is discussed in biochemistry.

Rule weight

For a realistic simulation of metabolism, many factors should be considered. The rate of each reaction depends on the concentration of compounds and the activity of the enzyme, and the latter further depends on the temperature, pH, and other physico-chemical factors.

Since the purpose of the framework is the search of logically possible pathways, these quantitative aspects are unnecessary. Also, its output is most useful when it is not biased by some additional criteria other than the number of reactions or number of involved compounds. It is possible, however, to integrate some preference to the order of output results by changing the rule weight.

One acceptable modification is to implement the comparative likelihood of generic rules against specific rules. Criterion II only imply that a specific rule should have a smaller weight than a generic one, but a preference among generic rules can be assigned for each compartment. For example, the reactions consuming NADH is unlikely to occur in cytosol, whereas they may proceed in mitochondrion. Such adjustment can be reflected in the rule weights.

Also the index of anabolism (reactions consuming energy) against catabolism (reactions producing energy), or that of oxidation-reduction (O-R) balance[5] may be reflected in the rule weights.

[4] In the current implementation, each atom of a compound will be assigned an integer label by the system, and the atomic position i of compound A can be specified as '`i in A`', which may be written in place of *molecule* in the BNF definition. With this method, however, a user has to check the integer label of a compound in every query. The best way to avoid this awkward process is to support a graphical user interface.

[5] This measure is used in the evaluation of fermentation process.

```
{ "cell"
    { "cytosol"
        glucose = C(O)[C@@H](O)[C@@H](O)[C@H](O)[C@@H](O)C=O;
        fructose = C(O)[C@@H](O)[C@@H](O)[C@H](O)C(=O)CO;
          :
      /* Bacteria do not have mitochondrion. */
        oxaloacetate = OC(=O)CC(=O)C(=O)O;
        pyruvate     = CC(=O)C(=O)O;
        malate       = OC(=O)CC(O)C(=O)O;
          :
    }
    /* enzymes or transporters */
    malate_dehydrogenase = malate, NAD "mitochondrion"
                         -> oxaloacetate, NADH "mitochondrion";
    pyruvate_carrier     = pyruvate "cytosol"
                         -> pyruvate "mitochondrion";
                              :
}
```

Figure 2.7: Example cell definition for glycolysis

2.3.2 Implementation

The metabolic framework is implemented in C^{++} using LEDA library package [64]. The system is named AMR (Automated Metabolic Reconstruction). A user can freely input compounds, enzymatic reactions, and cellular compartments. Generic rules correspond to basic reaction patterns (See the next section.), and they are fixed in AMR. The typical description of an eukaryotic cell looks like Fig 2.7.

Each compound goes through the detection of aromatic rings, followed by the normalization of its graph structure. The resulting graph is stored using a hash function associated with each cellular compartment. For an enzymatic reaction, its atomic mapping is pre-computed using the graph-matching algorithm, and the computed mapping is stored using a hash function. The search of subgraphs is realized by k shortest paths algorithm.

The reproduction of tracer experiments provides a novel view in metabolism. In this section, views for some basic metabolism are introduced. The following results are the graphical interpretation of AMR output. Phosphate is abbreviated as p.

EXAMPLE. **Glycolysis**
Glycolysis, the best-known metabolic pathway, is usually written separately from pentose phosphate pathway, although they share several compounds (fructose 6p and glyceraldehyde 3p). There are reasons for the separation. One is that glycolysis is the dominant pathway. AMR system can provide the second explanation: the top half of glucose (surrounded by dotted lines) does not join its bottom half through pentose pathway, and therefore pentose pathway cannot be used for gluconeogenesis (generation of glucose from pyruvate).

Fig 2.8 shows several shortest paths from the carbons in the top half of glucose to carbons in pyruvate, found by AMR from among 200 basic reactions in Enzyme Nomenclature. No generic reactions are used in this example. (Each arrow has a specific EC number, but EC numbers and coenzymes are omitted in the figure.) Note that erythrose

4p and xylulose 5p, compounds characteristic to pentose pathway, do not appear in the shortest pathways (shown by dotted arrows; upper left). In glycolysis, the top half of glucose is mapped to dihydroxyacetone p, and then overlaps the bottom half of glucose at glyceraldehyde 3p. This process (fructose bisphosphate aldolase) is crucial in the interchange of carbons between two halves of glucose.

It is also known that serine and glycerate may be involved in glycolysis (Fig 2.8; bottom right). The pathway through oxaloacetate is unlikely, though, because the reaction from phosphoenolpyruvate to oxaloacetate (phosphoenolpyruvate carboxykinase) is usually considered irreversible.

EXAMPLE. **Urea cycle**
The entire cycle in Fig 2.9 is not found by AMR, as long as it traces carbon atoms. Instead, the system finds two components conjugated at arginosuccinate. In each component, carbon backbones are recycled after carrying a nitrogen atom.

By searching longer pathways, AMR can find that the two cycles are not completely disjoint. The carbon backbone of ornithine may become glutamate through glutamate 5-semialdehyde (Fig 2.9; leftmost pathway). Note that its nitrogen atom is transferred to aspartate. Later, the carbon backbone of glutamate will enter TCA cycle and come back to the upper cycle. The backbone of ornithine may become glutamate through proline too, in which case, its nitrogen atom is excised as ammonia (pathway not shown).

EXAMPLE. **TCA cycle**
TCA cycle is referred to as Krebs cycle too. The primary location of the cycle is the mitochondrion, which generates ATP by electron transport-oxidative phosphorylation. Since its overall function is the decomposition of a two-carbon unit into carbon dioxides, the fate of the unit is usually displayed in biochemistry textbooks. AMR can reproduce such tracing results. Usually, a textbook shows (e.g. p.232 [31]) only red carbons as the fate of the second carbon of acetyl CoA (probably for educational purpose). However, in reality, the symmetry of citrate maps the carbon to blue positions too. AMR outputs both positions as the destination. It is also known that a carbon in acetyl CoA will be excised as carbon dioxide first at 2-oxoglutarate dehydrogenase (purple).

2.4 Reaction Patterns

Overview

AMR supports about twenty hypothetical reaction patterns. The classified result is used to predict possible reactions for each input chemical structure. The authoritative EC (Enzyme Commission) classification by IUBMB (International Union of Biochemistry and Molecular Biology) is not suitable for this purpose, because EC classification primarily focuses on the functional category of enzymes.

The whole section is dedicated to the explanation of the patterns of generic rules, corresponding to hypothetical enzymes. Although various types of chemical transformation is performed by enzymes, there is certainly a class of biologically plausible or frequent reactions. For example, an enzymatic transformation from hexane to cyclohexane is considered unlikely, because hexane has no chemical groups for enzymes to work on. We

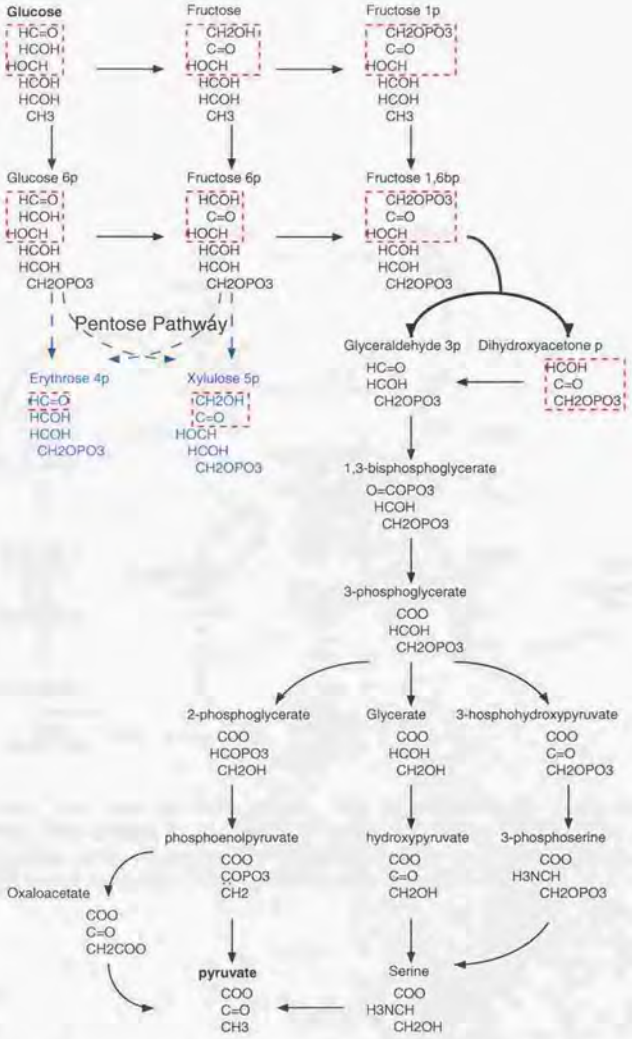


Figure 2.8: Glycolysis by AMR system The figure is manually drawn based on the several AMR outputs for the single query `?( glucose :> pyruvate )`.

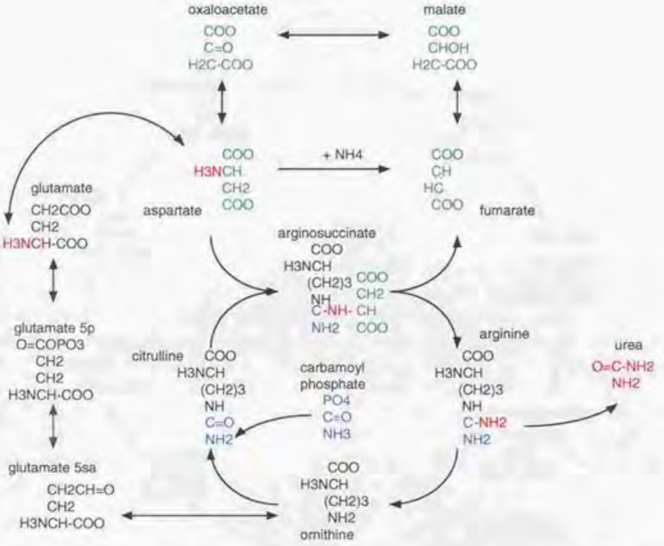


Figure 2.9: Urea cycle by AMR system. The figure is manually drawn based on the several AMR outputs for the queries: `?( arginosuccinate :> ornithine )`, `?( 8 in arginosuccinate :> malate )`, `?( arginosuccinate :> 2-oxoglutarate )`. The number 8 is used to specify a different carbon atom in arginosuccinate.

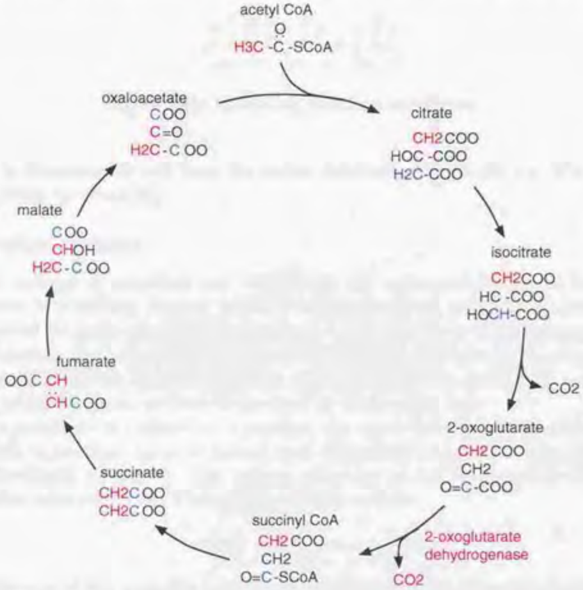


Figure 2.10: TCA cycle by AMR system. The figure is manually drawn based on the AMR outputs for the queries: `?( 2 in acetyl CoA :> oxaloacetate )`, `?( 2 in acetyl CoA :> CO2 )`.

classify biological reactions for basic metabolites, and summarize transformations performed by enzymes. Complex reactions in secondary metabolism are excluded from this original classification, because our concern is the biologically common reactions only.

In the following, we shall introduce pattern by pattern with comments for coenzymes. When the pattern is used in AMR system, additional restrictions written in '*Notes:*' section hold too. When a reaction pattern is applicable to the given compound in multiple ways, we define that atoms at chain terminals have the priority. For example, oxidase is applicable to glucose basically at five positions (Fig 2.11), but position 1 only is considered valid because of its terminal position. Thus, each pattern is applied only to terminals when several possibilities exist. The classification is based on the description from the textbooks in biochemistry and from the online databases [45, 85, 68, 41]. We especially owe to the book by Bugg [21].

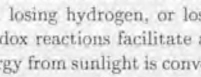


Figure 2.11: Oxidating positions on glucose

2.4.1 Redox reaction

Redox is a coinage of reduction and oxidization. A compound is said to be *oxidized* by either one of obtaining oxygen, losing hydrogen, or losing electron. *Reduction* is its reverse process. In green plants, redox reactions facilitate a series of high-energy electron transfer processes, in which the energy from sunlight is converted into high-energy reducing equivalents. This process ultimately leads to a fixation of carbon dioxide and a production of oxygen, which serves as an electron acceptor in mammalian cells.

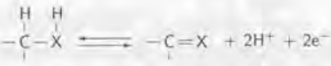
When a substrate is oxidized in a reaction, the other substrate involved is reduced. The direction of reaction can be estimated from the strength of an oxidizing agent, which is electrochemically measured. The voltage measured under the standard condition is known as the redox potential. The reference is the reaction

$$2H^+ + 2e^- \rightarrow H_2 .$$

The difference of this potential between compounds shows the thermodynamic favor of the chemical reaction. The strongest oxidizing agent is oxygen, therefore many enzymes use molecular oxygen as a substrate (oxygenases) or as an electron acceptor (oxidases). Common redox cofactors are nicotinamide adenine dinucleotide (NAD), nicotinamide adenine dinucleotide phosphate (NADP), flavin mononucleotide (FMN), and flavin adenine dinucleotide (FAD). Flavin is a coenzyme form of vitamin B_2 (riboflavin). We skip some enzyme classes such as peroxidase or hydrogenase, because they work on hydrogens which are not considered in our work.

Dehydrogenase, Oxidase

Pattern 1:



X = O, NH, CHR

AMR Notes:

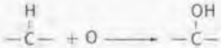
- C–X should not be aromatic.
- C should not be α-carbon of amino acid.

Dehydrogenases transfer two hydrogen atoms from a reduced substrate, usually an alcohol, to an electron acceptor, usually NAD^+. The ratio of $NAD^+/NADH$ and of $NADP^+/NADPH$ in cytosol is about 1200 and 1, respectively. Therefore, NAD^+ tends to be used for catabolic processes (oxidation of substrates) converting itself to NADH, whereas NADP is preferred in biosynthesis (including reduction of substrates) as the counter process.

Other dehydrogenases depend on FMN or FAD. The difference between nicotinamide and flavin is that the latter can exist either as oxidized FAD, or reduced $FADH_2$, or as an intermediate semiquinone radical FADH. Therefore flavin can carry out one-electron transfer reactions such as the one with molecular oxygen, whereas nicotinamide transfers two electrons at a time. Many dehydrogenases in mitochondria are flavin proteins.

Hydroxylase, Mono-oxygenase

Pattern 2:



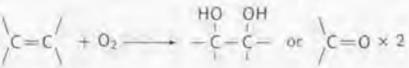
AMR Notes:

- No more than one oxygen is incorporated at each carbon.

The ability of flavin to react with molecular oxygen enables mono-oxygenase reaction, in which one atom from molecular oxygen is incorporated into the product. Tyrosine is generated from phenylalanine with this function. The important role of vitamin C (ascorbic acid) is to hydroxylate proline in the precursor protein of collagen. In mammalian cells, there is a class of enzymes which can catalyze the specific hydroxylation of unactivated alkanes using Haem cofactor, not flavin. There are several other cofactors such as deazaflavins, pterins, and iron-sulphur clusters.

Dioxygenase

Pattern 3:



AMR Notes:

- C=C may be aromatic.

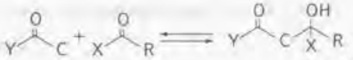
Dioxygenases incorporate both atoms of dioxygen into the product. The difference from the mono-oxygenase is that they use α-ketoglutarate (2-oxoglutarate) as a substrate, which will become succinate and carbon dioxide, and that they can use non-haem iron as a cofactor.

2.4.2 Carbon-Carbon bond formation

Formation of a carbon-carbon bond is central to biosynthesis, in which various metabolites such as amino acids, carbohydrates, and nucleic acids are generated from smaller compounds.

Aldolase and Claisen enzymes

PATTERN 4:



X = C, H; Y = C, OH

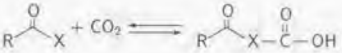
AMR Notes:

- C and X are non-aromatic terminal atoms.
- The attached carbonyl compound is acetyl CoA, i.e. Y = CoA.

The aldol condensation takes two carbonyl compounds to form carbon-carbon bond. Cleavage of carbon-carbon bonds by the reverse reaction is also possible. It is similar to Claisen ester condensation, in which carboxylic esters to form β-keto esters with the presence of alkoxide ions. Biologically, Claisen reaction occurs only with thioesters, which behave more like a ketone than esters. The most typical thioester used for biological condensation is the coenzyme A (CoA) ester, which performs acyl transfer too.

Carboxylase

PATTERN 5:



X = N, C

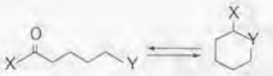
AMR Notes:

- X is a terminal atom (therefore not aromatic).

The fixation of carbon dioxide by green plant is the ultimate carbon source of living organisms. The responsible enzyme is called *r*ib*u*lose b*i*sphosphate *c*arboxylate/*o*xygenase, a carboxylase known as Rubisco. Carboxylases utilize an attack of carbanion equivalents onto carbon dioxide to generate carboxylic acid products. The typical cofactor is biotin (vitamin H). The decarboxylation, often oxidative, is also widely found in biological reactions. β-keto acid is readily decarboxylated, but α-keto acid requires the coenzymes, thiamine pyrophosphate (TPP) and NAD$^+$. TPP is a coenzyme form of thiamine, i.e. vitamin B$_1$.

Cyclase

PATTERN 6:



X = O, COR; Y = N, C

AMR Notes:

- The length of the cycle is five or six.

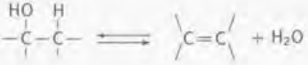
The conversion of allylic pyrophosphate substrates into terpenoid products is an important biological process in plant metabolism. The common feature of terpene products is the five-carbon isoprene unit, which is cyclized by cyclases. On the other hand, certain aromatic products are formed by the radical coupling of phenols. Lignin in plants is such an example, and it's highly heterogenious polymer structure shows that its formation is a chemical reaction, not a uniform enzyme-catalyzed one.

2.4.3 Addition or Elimination

The addition and elimination of water is a common process in biochemical pathways.

Hydratase and Dehydratase

PATTERN 7:



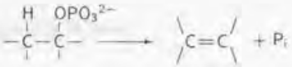
AMR Notes:

- C–C backbone is not aromatic.

The reaction involves the cleavage of a C–H bond and the cleavage of a C–O bond. The reaction is usually reversible. The β-oxidation of fatty acids begins with oxidation by a group of acyl-CoA dehydrogenases, all of which are flavoproteins.

Elimination of phosphate

PATTERN 8:



- C–C backbone is not aromatic.

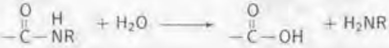
In elimination reactions, the key factor is whether a good leaving group is available. For example, phosphate is a much better leaving group than hydroxide ion.

2.4.4 Hydrolysis or Transfer

All of the hydrolases cleave a chemical group and transfer it to the hydroxyl group of water, while transferases simply transfer this group to an acceptor other than water. Since the acceptor is unspecified in general, both reactions are treated by the same pattern in AMR. That is, the acceptor is always a water molecule.

Protease, Transpeptidase

PATTERN 9:



AMR Notes:

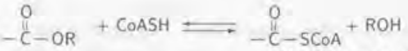
- C–N backbone is not aromatic.

Proteases or peptidases are repsonsible for hydrolyzing the amide bonds in the polypeptide structures of proteins. It plays an important role in the digestive system of all animals. There are four classes of peptidase enzymes, classified according to the groups at their active site: (1) the serine proteases; (2) the cysteine proteases; (3) the metalloproteases; and (4) the aspartyl proteases.

The transferase counterpart of protease is called transpeptidase.

Esterase, Acyltransferase

PATTERN 10:



CoASH may be R'OH or H_2O.

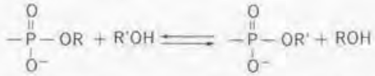
AMR Notes:

- C–O backbone is not aromatic.

Esterases and lipases hydrolyze the ester functional groups of fats and oils, and work in the digestive system of animals. Since these enzymes form a covalent acyl enzyme intermediate, they can transfer the acyl group to an alcohol acceptor. Such enzymes are called acyltransferase. This function is prevalent in fatty acid biosynthesis, polyketide natural product biosynthesis, and the assembly of a variety of amide and ester groups. Its typical coenzyme is CoA, because thiol is a good leaving group. Acetyl CoA can activate not only acyl group but also methyl group.

Phosphoryl Transfers (Phosphatase, Kinase)

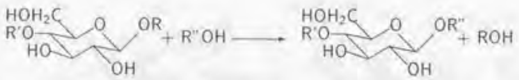
PATTERN 11:



Phosphoryl transfer is of fundamental importance for the biosynthesis and replication of nucleic acids, and in the transfer of phosphate groups. This reaction usually involves the transfer of phosphoryl groups from a 'high-energy' species like ATP. Phosphate monoesters can be readily hydrolyzed under basic conditions, whereas phosphodiesters are resistant to basic hydrolysis. Enzymes that cleave phosphate monoesters is called phosphatases, and those cleaving the phosphodiester backbone of RNA and DNA are called nucleases. Restriction enzyme is one type of nuclease.

Glycosidase, Glycosyl transferase

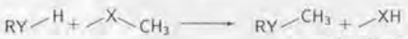
PATTERN 12:



Glycosyl transfer enzymes are highly specific: they can select a particular monosaccharide and its α- or β-glycosidic linkage. They are involved in the assembly of polysaccharides and oligosaccharides.

Methyltransferase

PATTERN 13:



X = N, S; Y = O, N, S

Methyl group transferred in nature is provided by S-adenosyl methionine (SAM). The byproduct of methyltransferase enzymes is S-adenosyl homocysteine, which is re-cycled to SAM via hydrolysis to homocysteine and adenosine.

Another cofactor tetrahydrofolate (THF) can transfer methyl (CH_3), methylene (-CH_2-) or methenyl (- CH=) group. The conversion of THF to methylene-THF is carried out by serine hydroxymethyltransferase, generating glycine as a byproduct. Methyne-THF can be synthesized either by the NADP-dependent oxidation of methylene-THF, or by a synthetase enzyme which uses formyl phosphate. Methyl-THF is synthesized by the NADP-dependent reduction of methylene-THF. This methyl-THF works in methionine salvage pathway, where homocysteine changes to methionine by obtaining a methyl group from methyl-THF. Methionine later couples with ATP by the unusual displacement of its triphosphate to regenerate SAM.

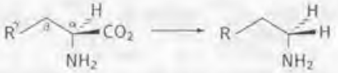
2.4.5 Reactions for Amino Acid

L-amino acids are required for the assembly of proteins and for the biosynthesis of alkaloids. The main cofactor for the common transformation of α-amino acids is pyridoxal 5'-phosphate (PLP).

PLP is the coenzyme form of vitamin B_6 (pyridoxine). The amino acid is bound to its pyridine ring forming an aldimine adduct (Schiff's base), and the ring works as an electron sink. Depending on the type of electron shifts, eight main types of reactions are recognized: transamination, β-decarboxylation, α-decarboxylation, aldol cleavage, γ-elimination, β-elimination, γ-displacement, or β-displacement. In addition, PLP enzymes carry out unique reactions that do not fall into this classification.

Decarboxylase

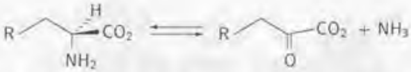
PATTERN 14:



Decarboxylases are basically irreversible. The only exception is diaminopimelic acid (DAP) decarboxylase, for which lysine is the final product. Pyridoxal phosphate is the coenzyme of all the amino acid decarboxylases except for histidine decarboxylase, for which the cofactor is pyruvate. Optimal activity is pH 3–5. The diamine putrescine, the decarboxylation product of ornithine, is an essential growth factor for several organisms and is a biosynthetic precursor of both spermidine and spermine.

Oxidative deaminase

PATTERN 15:



The oxidative deamination of glutamic acid by the glutamate dehydrogenases is the most widely used route for ammonia assimilation. The reaction is reversible, but the operation primarily proceeds in the direction of catabolism when it is NAD-linked. On the other hand, the NADP-linked deaminase works in glutamate synthesis.

The deamination linked with FAD or FMN operates in two stages: (1) reduction of FAD (or FMN) through deamination, (2) nonenzymatic oxidation of FAD (or FMN) by molecular oxygen. These amino acid oxidases are nonspecific; it can work on a variety of amino acids with the different order of activity. It is highly unlikely that they are used in ammonia assimilation (Thus, it is irreversible.).

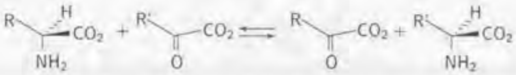
Nonoxidative deaminase

The nonoxidative deaminases are usually specific in their substrate requirements. They are also called ammonia lyase. AMR does not consider their specificity, but we list their category for the completeness of the classification.

- **Aspartase** is present in many organisms. It is reversible, but is primarily involved with catabolic activity.
- **Serine (or threonine) deaminase** catalyzes two-step reaction. First, it transfers hydrogen atoms in serine and removes water to produce an imino acid (β-elimination). Next, imino acid nonenzymatically reacts with water to release ammonia.
- **Cysteine desulfhydrase** conveys a similar reaction as serine deaminase, but it uses hydrogen sulfide instead of water via a β-elimination reaction. It is irreversible.
- **Phenylalanine deaminase** occurs in yeasts, molds, and other bacteria. It catalyzes the nonoxidative deaminiation of L-phenylalanine to trans-cinnamic acid. Dehydroalanine group serves as a cofactor.

Transaminase

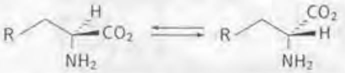
PATTERN 16:



Transaminase activity is ubiquitous. Keto analogs can replace many of the amino acids for the growth of amino acid requiring mutants. The reaction consists of two half-reactions. First, the amino group is transferred to PLP, forming a pyridoxamine 5'-phosphate (PMP). The reaction is completed by carrying out the reverse transamination on the other keto acid substrate. The specificity of the reaction is not completely universal, but one enzyme is usually applicable on several types of amino acids. When focused on a single compound, its activity is the same as deaminase, but it enables the assimilation of amino group from another amino acid.

Racemase

PATTERN 17:



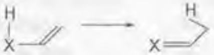
The amino acids found in naturally occurring proteins are usually in L configuration, but the cell walls and polypeptide capsules of many organisms contain D-amino acids. The general reaction converting L-amino acids to D-amino acids is found in many organisms. One important example of a PLP-dependent racemase is alanine racemase, which is used by bacteria to generate D-alanine. There is also a family of cofactor-independent racemases. (In AMR, generic reactions do not consider chirality. We list this category for the completeness of the classification.)

2.4.6 Isomerase

In addition to racemases shown in the previous section, enzymes can perform the interconversion of tautomeric forms of ketones, and the interconversion of positional isomers of allylic compounds. (In AMR, isomerase is not supported. We list this category for the completeness of the classification.)

Keto-enol tautomerase

PATTERN 18:

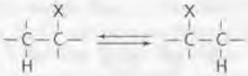


X = O, C (allylic)

In most cases enols and enolate anions are thermodynamically unstable. However in a few cases, enol tautomers of ketones are stable enough to be isolated, and a tautomerase can produce such enol forms. One example is phenylpyruvate tautomerase. The atom X may be carbon in the allylic isomerases.

Allylic isomerase

PATTERN 19:



X = COOH, OH

There are many allylic isomerases which operate in many different areas of metabolism. Vitamin B_{12}-dependent enzymes can perform 1,2-migration of a hydrogen atom, and the corresponding 2,1-migration of another substituent. Together with isomerase,

2.4.7 Appendix

The reaction patterns will be used for generic rules in AMR system. Although the information of cofactors/coenzymes is unnecessary for finding pathways, it is important in the functional assignment of DNA sequences. Some reaction patterns often require a specific type of coenzymes/cofactors, which is summarized in Fig 2.12.

pattern	coenzymes/cofactors
dehydrogenase	nicotinamides, flavins
hydroxylase	flavins
dioxygenase	flavins, irons
carboxylase	nicotinamides, biotin, metal ions
hydratase	metal ions
esterase	CoA, thiol
kinase	adenosine/guanosine phosphate
methyltransferase	folates, adenosyl methionine
for amino acid	pyridoxal phosphate

Figure 2.12: Table of Coenzymes

Vitamin	Chemical name	Biochemical function	Deficiency disease
A	Retinol	visual pigments	Night blindness
B_1	Thiamine	TPP	Beriberi
B_2	Riboflavin	FAD, FMN	Skin lesions
B_6	Pyridoxal	PLP	Convulsions
B_{12}	Cobalamine	radical rearrangement	Pernicious anaemia
C	Ascorbic acid	anti-oxidant	Scurvy
Niacin	Nicotinamide	NAD	Pellagra
D	Calciferols	calcium homeostasis	Rickets
E	Tocopherols	anti-oxidant	Haemolytic anaemia
H	Biotin	carboxylation	Skin lesions
K	Phylloquinone	anti-oxidant	Bleeding disorders

Figure 2.13: Table of Vitamins

Summary and Discussion

This chapter introduced the declarative representation of metabolism in the cookie-cut framework. The crucial point is that metabolism is transformed to a normal graph at the atomic level, not the compound level. In the transformed graph, pathways can be efficiently searched by a variant of k shortest paths algorithm. In order to deal with unknown reactions, the original classification of enzymatic reactions is also introduced.

The metabolic version of the cookie-cut framework is implemented as AMR system. AMR generates a metabolic graph based only on the user-defined compound structures and reaction formulas. The system can extract a model for simulation from the metabolic graph, according to a specified pattern graph. For example, glycolysis is specified as a linear path from glucose to pyruvate, and TCA cycle is specified as a cycle with citrate, succinate, and malonate.

AMR computes the mapping of atoms for each metabolic reaction, and therefore can reproduce the tracer experiment in biochemistry. The tracing at the atomic level enables the system to check whether a certain compound actually comprises from basic cellular compounds. The reproduction of the tracer experiment or the generation of hypothetical pathways could not have been achieved in traditional hypertext-based databases [90, 57, 73].

The system is not knowledge-based; it does not require any pre-determined knowledge, such as 'Acetyl CoA is always cleaved to acetyl group and CoA.' Only the basic enzymatic reactions in Section 2.4 are given as the prior knowledge for hypothetical reactions. Consequently, a user can freely design any metabolism, and there is no assumption such as a certain compound is a coenzyme, or is localized in the nucleus.

Among previous computational tools, Klotho [53] is notable, because it describes the structure of chemical compounds in Prolog. It can also reproduce tracer experiment, by writing a procedure for the structure matching in Prolog. In this respect, AMR can be seen as a graph theoretic counterpart of Klotho. The advantage of AMR is that it can represent the cellular structure, and that the underlining graph algorithm is efficient. Also, SMILES format and reaction formulas are much simpler than a Prolog format.

Although AMR accepts a readable input format, its output result is hard to understand at a glance without a graphical user interface. (For an example session, see Appendix at the end of this thesis.) We are used to see metabolism as a pathway maps, which is a combination of several related pathways such as 'amino acid biosyntheses'. Therefore, graphical interface is of great importance in displaying several search results in combination. This is the main future work to be completed.

Another future work is the more flexible integration of generic rules into the pathway search. Enzymatic reactions are classified into about twenty patterns by structures, but the obtained patterns could not be used in a pattern-matching fashion: in the pattern-match, a larger compound would undergo more variety of transformations, while in reality, it has less applicability of enzymes. For example, the most diversely used metabolic compound is pyruvate (comprising three carbon atoms), while sugars and DNAs interact only with their special enzymes such as glycosidase or ligase. This observation must be reflected to the application of generic rules in AMR. In the current implementation, a restricted number of compounds are subjected to the generic rules for this reason.

Chapter 3

Computerizing Reactions

Everything of importance has been said before by somebody who did not discover it.
— Alfred North Whitehead (1861–1947) Philosopher

3.1 Overview

The good model of a chemical system, compounds as well as reactions, has long been pursued by two giant communities: one in chemistry and the other in biochemistry.

The chemistry group has mainly focused on the construction of database systems, in which the knowledge representation of compounds is of crucial issue. A database for compounds usually contains: (1) unique identification numbers defined by CAS (Chemical Abstract Service); (2) English names standardized by IUPAC (International Union of Pure and Applied Chemistry); (3) molecular structures represented as adjacency matrices or three-dimensional coordinates; (4) other chemical characteristics such as stability or boiling point. The sub-structural search on a database is usually performed using bit vectors. In the database, each compound is associated with a bit vector, in which each bit represents a particular small substructure, or a chemically functional group in the compound. In the search, the query compound is interpreted as a small set of bits, and this set is searched against a large number of bit vectors in the database. With this technique, the chemical information of ten thousands of compounds can be easily retrieved in many commercial database softwares.

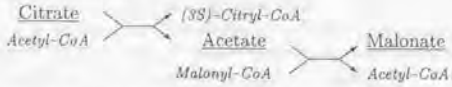
The biochemical group, on the other hand, is concerned with more kinetic aspects of the system. Different reaction patterns under equilibrium as well as non-equilibrium have been modeled with differential equations with experimentally quantified kinetic parameters. In the equations, each variable represents a compound or an intermediate complex, and its structural information is usually not considered. When the number of involved components is small, simulation-softwares can accurately simulate chemical and enzymatic reactions.

The above two modeling strategies show a striking contrast. One group deals with compounds at the atomic level in a qualitative, dictionary-like fashion, while the other tries to accurately describe reactions at the compound level in a quantitative, temporal fashion.

In its size, metabolism occupies the middle position between these extremes, and do not render itself to either of these successful modeling schemes. Historically, metabolism has been modeled by biochemists, whose major interests include biosynthesis and degradation in living organisms. However, as the size of the metabolism under focus increases, the biochemical modeling will end in a rupture for several reasons.

First, reaction formulas or differential equations do not consider molecular structures.

Without the structural information, it is impossible to construct the reaction pathway from each reaction formula.



Let us consider the above short pathway from citrate to malonate. Each $\mathcal{H}$-shaped arrow corresponds to an enzymatic reaction. The above pathway is not correct: the connected reactions only exchanges CoA; citrate, acetate, and malonate share nothing in common. The wrong connection results from ignoring their molecular structures. This flaw may be fixed by adding knowledge on chemical groups, but there is no unique decomposition of molecules into groups. Thus, atomic information is necessary for modeling metabolism.

Second, the biochemical modeling requires the detailed information of reactions, such as kinetic parameters and the concentration of compounds. Even if the amount of kinetic knowledge may rapidly increase in a future, it is impossible to experimentally measure all the necessary parameters. Even worse, the biochemical modeling is sensitive to these parameters. When a new experimental result (e.g. by knockouting a novel gene) comes out after having tuned all the required parameters, the effort having been done for tuning the parameters would have to be repeated. Thus, parameters are better omitted for modeling metabolism.

The above difficulties do not arise, if **we model metabolism qualitatively at the atomic level.** With a monotonic knowledge representation, we can accumulate laboratory results for a better performance, without fussing on parameters. The difference of the metabolic model from a chemical database is that the number of compounds is much smaller, and that the information of enzymatic reactions must be seamlessly combined with that of compounds. The main problem is how to represent and associate the molecular structures in a reaction.

In the cookie-cut framework, molecular structures are represented as graphs. A reaction is interpreted as a mapping between graphs. The sections are organized as follows.

In Section 3.2, SMILES notation is introduced. It is an input format allowing a flexible, easy-to-understand description of molecular structures. SMILES resembles a conventional notation for chemical structures, and can express any compound, including ions, in a linear string format.

When the input SMILES is converted to a graph, two procedures are necessary for the identification of its molecular structure: (1) the detection of aromatic rings, and (2) the computation of its normal form. Both operations are necessitated by the flexibility of SMILES: a user may input the same molecule in different SMILES. This flexibility also exists in IUPAC names or in the input from a graphical interface, and both operations have been a long-standing difficulty in chemical computation. We consider these problems in the following two sections.

In Section 3.3, aromaticity is re-defined so that the detection of aromatic rings can be theoretically discussed. Then the algorithm for finding a minimal cycle basis is shown to be applicable for our purpose.

In Section 3.4, the classical algorithm for automorphism partition is re-introduced, and is shown to be applicable to the computation of normal forms. Both algorithms are practical and easy to implement.

In Section 3.5, the strategy to treat chirality for identifying compounds is discussed. In metabolism, chirality is an important characteristic of carbon atoms, because most enzymes show a specific preference on chirality. For example, it is well known that L-amino acids are more preferred than D-amino acids in a living organism. Nevertheless, the computational treatment of chirality has not been well discussed in chemical computation, probably because its application area will be biochemistry, not chemistry. We show a guideline on the extent of how faithfully the chiral information needs to be considered.

In Section 3.6, an algorithm for finding a maximal common subgraph between a graph and a tree is introduced. This new branch-and-bound method is applied for the modeling of enzymatic reactions. The basic idea is that the enzymatic modification can be detected by finding a maximal common subgraph between substrates and products.

3.2 SMILES

SMILES is a nomenclature for chemical structures, proposed by Weininger [104]. It is one of the most widely used format in chemical computation. This section introduces its basic definition, i.e. the minimum amount necessary to understand our work. The original definition is more detailed and thorough to describe various attributes of chemical molecules, and Backus-Naur form (BNF) of original SMILES is more complex than the one introduced here. [1] Readers interested in SMILES are referred to the on-line tutorial by its developer. [2] This section is essentially a synopsis of SMILES tutorial by Daylight Chemical Information Systems under its permission. In BNF, the *slanted* or `typewriter` face denotes terminal symbols.

3.2.1 Definitions

Atoms

```
       atom ::= B | C | N | O | P | S | F | Cl | Br | I | braced_atom
braced_atom ::= [ label atomic_symbol hydrogens charge ]
      label ::= empty | integer
  hydrogens ::= empty | H | H integer
     charge ::= + integer | - integer
```

An atom is specified by a conventional atomic symbol with brackets. Frequently used elements in organic chemistry (B, C, N, O, P, S, F, Cl, Br, and I) may be written without brackets, if the number of attached hydrogens conforms to the lowest normal valence consistent with explicit bonds. Lowest normal valences are B (3), C (4), N (3,5), O (2), P (3,5), S (2,4,6), and 1 for the halogens. Attached hydrogens are implied in the absence of brackets, otherwise the number of hydrogens must be explicitly specified. An integer value after H may be omitted if it is 1. A label is used to specify isotopes. Atoms in aromatic rings[3] are specified by lower case letters, and chiral carbons are specified by either C@ or C@@.

Examples.

C	CH_4	methane	[C]	C	elemental carbon
[S]	S	elemental sulfur	[OH-1]	OH^{-1}	hydroxide anion
P	PH_3	phosphine	[Fe+2]	Fe^{+2}	iron (II) cation
[235U]	^{235}U	uranium-235			

[1] The BNF in this paper is our original.

[2] http://www.daylight.com/dayhtml/smiles/

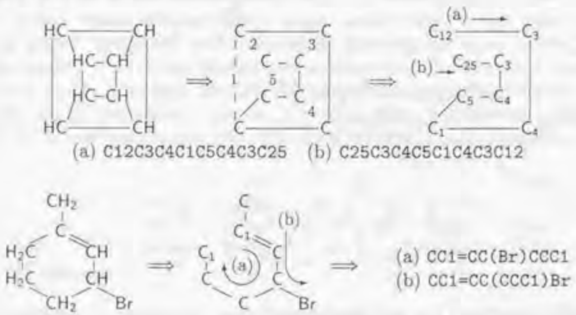
[3] The word *ring* seems preferred in chemistry, but the word *cycle* is used in this paper.

Molecules

```
    bonds ::= empty | - | = | # | :
    rings ::= bonds integer
 molecule ::= bonds atom rings | molecule bonds atom rings
            | molecule ( molecule )
```

Single, double, triple, and aromatic bonds are represented by the symbols -, =, #, and :, respectively. Symbols for a single or aromatic bond may be omitted. Branches are specified by enclosing them in parentheses. Cycles are specified by appending pairs of integer labels instead of bond symbols. Note that linear notation can describe only a tree structure. Therefore, integer labels are attached to the specifications of the atoms connected by non-tree edges. Bond symbols may precede the integers to show the type of non-tree edges.

Fig 3.1 shows some examples from Daylight Chemical Information Systems under its permission. Note that there are many different, but equally valid SMILES descriptions for the same structure.



(a) C12C3C4C1C5C4C3C25 (b) C25C3C4C5C1C4C3C12

Figure 3.1: SMILES examples

From now on, chemical molecules are written in SMILES using `typewriter` face.

3.3 Detection of Aromatic Cycles

3.3.1 Definitions

A *molecular graph* $G(V, E, f_V, f_E)$ is a connected undirected graph of colored nodes and edges. It is without self-loops or multiple edges between the same nodes. Colors correspond to the type of atoms or bonds, and are derived by the functions $f_V : V \to \mathrm{N}$ and $f_E : E \to \mathrm{N}$, respectively. We define the color number of nodes as the atomic number of their corresponding atoms. That is, f_V('carbon node') = 6 and f_V('oxygen node') = 8, for example. Hydrogens are omitted in a molecular graph as in SMILES. [4] The color number of edges is the number of their contributing electrons. That is, f_E('single bond') = 1, f_E('double bond') = 2, for example. An aromatic bond is defined f_E('aromatic bond') = 0. When the type of atoms and bonds are not considered, f_V and f_E are omitted, and

[4] Note that the number of hydrogens attached to each atom can be computed from the valence and the number of connected bonds.

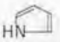
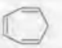
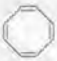
the structure is simply called a *graph*. In considering time complexity, we abuse symbols V and E for $|V|$ and $|E|$.

A *path* in G is a sequence of nodes $p_{uv} = (u = v_1, v_2, \cdots, v_k = v)$ such that $(v_i, v_{i+1}) \in E$ for $1 \leq i < k$. Its *length*, $(k-1)$, or the *distance* between u and v, is denoted $w(p_{uv})$.

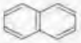
A *cycle* is a path in which the first and the last nodes are identical. A path is *simple* if no node appears twice. Note that shortest cycles are simple. Two cycles, C and D, can be added to form their sum: $C + D = (C \cup D) - (C \cap D)$. The set of all cycles is closed under the addition, forming the *cycle space*. The set of simple cycles which can generate all cycles is a *cycle basis*, and a *minimum cycle basis* $\mathbf{B}$ is a basis minimizing $\sum_{C \in \mathbf{B}} w(C)$. A cycle is *relevant* if it belongs to at least one minimum cycle basis.

3.3.2 Background

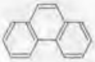
The theoretical basis for aromaticity is easier to state than the experimental criteria. In chemistry, a cycle is aromatic if: (1) its carbon atoms are completely conjugated; (2) it is planar[5]; (3) it contains $(4n+2)$ π electrons, in which n is a positive integer (Hückel rule). Complete conjugation means the alternative appearance of double and single bonds. Inside conjugated bonds, electrons take a special orbit called π-orbit. Each double bond in the conjugation contributes two π electrons, and each nitrogen, oxygen, or sulphur that is not doubly bonded contributes two π electrons. Other type of atoms do not form aromatic cycles. Therefore, benzene (`C1=CC=CC=C1`), pyrrole (`N1C=CC=C1`), and furan (`O1C=CC=C1`) are all considered aromatic. Cyclobutadiene (`C1=CC=C1`), cycloheptatriene (`C1=CC=CC=CC1`), and cyclooctatetreene (`C1=CC=CC=CC=C1`) are not aromatic.

						
	benzene	pyrrole	furan	non-aromatic cycles		
π electrons	6	6	6	4	6	8

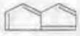
The rule is basically applied to a monocyclic structure, not to a condensed polycyclic structure. The characteristic of polycyclic structures depends on the pattern of condensation. For example, naphthalene (`C1=CC=CC2=C1C=CC=C2`) and phenanthrene (`C1=CC=CC2=C1C=CC3=C2C=CC=C3`) are aromatic, but their outer cycles do not satisfy Hückel rule. No cycles in pentalene (`C1=CC=C2C1=CC=C2`) follow the rule, but it is called nonbenzenoid aromatic compound. [6]



naphthalene

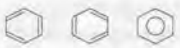


phenanthrene



pentalene

In order to normalize or identify an input SMILES, however, the detection of aromatic cycles is necessary, because different descriptions are valid for the same molecule. For example, `C1=CC=CC=C1` and `c1ccccc1` both denote benzene, and this discrepancy must be automatically resolved by converting a non-aromatic description to an aromatic one. The immediate solution is to apply Hückel rule to 'small' cycles in the structure.



Three descriptions of benzene.

[5] This planarity is a physical planarity of the molecular ring, not the planarity of its corresponding graph structure.

[6] This compound is not stable.

Theoretically the problem is related to the one of finding simple cycles in a graph. However, the rule is applied only to a 'small' cycles, and the computation of all simple cycles is often unnecessary. Let us characterize aromatic cycles before considering an algorithm for their detection.

- An aromatic cycle is constituted of the limited atomic types: carbon, oxygen, nitrogen, or sulphur. Atoms other than carbon do not adjacently appear, because of the stability of cycles. The valences of nitrogen and sulphur are variable in general, but within an aromatic cycle, their valences are always three and two, respectively. Therefore, the degree of a node in an aromatic cycle is less than four, because (1) oxygen, nitrogen, and sulphur all have a valence smaller than four; (2) a carbon must be connected with a double bond to be in an aromatic cycle, and consequently its degree will be less than four.
- An Aromatic cycle is planar. The physical planarity of each cycle does not necessarily imply the graph-theoretic planarity of the entire conjugated structure, but the distribution of π electrons forces the whole structure to be planar. Although this is an empirical observation, the shared edges usually connect carbon atoms. Because one double bond contributes two π electrons, the conjugate cycle of aromatic cycles often satisfies the $4n + 2$ rule of aromaticity.

Observation. An aromatic cycle is planar, and the degree of its nodes is less than four.

Next, we re-define aromaticity so that both mono- and polycyclic structures are treated under the same scheme.[7]

Definition. The function f_π maps a node $v \in V$ in a molecular graph $G(V, E, f_V, f_E)$ to an integer.

$$f_\pi(v) = \begin{cases} 2 \ldots & \text{if } f_V(v) = 7, 8, \text{ or } 16 \quad \text{(nitrogen, oxygen, or sulphur)} \\ & \quad \text{and } (\forall e = (v, w) \in E)\ f_E(e) \neq 2 \text{ (no double bonds)} \\ 1 \ldots & \text{if } (\exists e = (v, w) \in E)\ f_E(e) = 0, \\ & \quad \text{or } f_E(e) = 2 \text{ for only one such } e \\ & \qquad\qquad \text{(one double, or a few aromatic bonds)} \\ 0 \ldots & \quad \text{otherwise} \end{cases}$$

The following procedure finds *aromatic* cycles.

AROMATICITY DETECTION

1. Find a molecular subgraph of G induced by the set of nodes $\{v \mid v \in V, f_\pi(v) > 0\}$. Let G' be this induced graph.
2. If a simple cycle C in G' satisfies $\sum_{v \in C} f_\pi(v) = 4n + 2$ $(n = 1, 2 \ldots)$, then C is *aromatic*.

□

The check for aromaticity should be applied to all simple cycles, but in practice, only 'small' cycles are of interest. Conventionally, chemists call these cycles as 'Smallest Set of Smallest Rings' of the given molecular graph [35]. In this paper, however, we do not use their definition, and follow the graph-theoretic characterization.

[7] This definition is our original.

Definition. (Vismara [100]) A cycle C is *relevant* if it belongs to at least one minimum cycle basis. □

Observation. A cycle C is relevant if and only if there is no cycle D such that $w(D) < w(C)$ and $C = D + D'$ for some D'.

PROOF. ($\Rightarrow$) If C is relevant, C is in some minimum cycle basis $\mathbf{B}$. If D and D' satisfy $D + D' = C$ and $w(D) < w(C)$, $\mathbf{B} - \{C\} \cup \{D\}$ becomes a smaller cycle basis, which is a contradiction. ($\Leftarrow$) Suppose there is no D such that $w(D) < w(C)$ and $C = D + D'$. In some minimum cycle basis $\mathbf{B}$, C must be generated as a combination of cycles in $\mathbf{B}$ and is written $C = \sum_{i=1}^{k} D_i$ ($D_i \in \mathbf{B}$). From the assumption, $w(C) \leq w(D_i)$. If $w(C) < w(D_i)$ for any D_i, $\mathbf{B}$ is not a minimum cycle basis, which is a contradiction. If $w(C) = w(D_i)$ for any D_i, $\mathbf{B} - \{D_i\} \cup \{C\}$ becomes a minimum cycle basis. □

The observation shows that relevant cycles are in fact those of our interest. Therefore, we apply the '$\sum_{v \in C} f_\pi(v) = 4n + 2$' check of aromaticity only to relevant cycles, not to all simple cycles.

By this procedure, all nodes in naphthalene and phenanthrene will be included in aromatic cycles. Cycles in pentalene remain non-aromatic, but the decision of their aromaticity (in a chemical sense) also relies on experimental data. We shall discuss on this issue later.

3.3.3 Finding Cycles

We have seen that the detection of aromatic cycles is achieved by the detection of relevant cycles.

Problem 1.
Instance: Biconnected (molecular) graph G.
Question: Find all relevant cycles.

Vismara showed a polynomial-time algorithm which computes a compact representation of the set of relevant cycles. A polynomially sized representation is necessary, because the number of relevant cycles may be exponential in general (Fig 3.2).

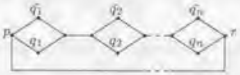


Figure 3.2: There are 2^n patterns from p to r. ($2^{n/4}$ cycles with $3n/4$ nodes)

In a biochemical application, however, such exponential case does not appear. Therefore, it is more appropriate to list all possible relevant cycles, than to use a compressed representation of cycles. The problem is basically solved by the algorithm for computing a minimum cycle basis.

Theorem 3.3.1 (Hubicka and Syslo [47]) *If e is an edge, and C is a shortest cycle through e, then C is in some minimum cycle basis. Moreover, any minimum cycle basis must contain some shortest cycle through e.*

The Theorem 3.3.1 is of fundamental importance in the theory on cycle basis. In general, the set of cycles $S = \{C_e \mid C_e \text{ is a shortest cycle through } e \in G\}$ needs not contain a minimum cycle basis. Horton's algorithm finds a minimum cycle basis in polynomial-time, by generating cycles for each node and edge, and by applying Gaussian elimination [46].

Horton's algorithm

1. Compute the shortest path $\bar{p}_{uv}$ for each pair of nodes u and v in G.
2. For each node w and edge (u, v), create the cycle $C(w, u, v) = \bar{p}_{wu} + \bar{p}_{wv} + (u, v)$ and calculate its length. Degenerate cases in which $\bar{p}_{wu}$ and $\bar{p}_{wv}$ share nodes other than w are omitted.
3. Sort the cycles by length.
4. Consider the cycles as the rows of a binary matrix, whose columns and rows correspond to the edges and the incidence vectors of the cycles, respectively. Perform Gaussian elimination in the order of the length. When enough independent cycles have been found, the process stops.

The last step dominates the computational time of the algorithm, and its order is $O(VE^3)$. Vismara's algorithm for finding (the compact representation of) relevant cycles is similar, and has the same time complexity.

For our purpose, the enumeration of relevant cycles is necessary to check the aromaticity for each cycle, and the computation of cycle bases is unnecessary. The ad-hoc solution of the Problem 1 is to compute all shortest paths to generate all relevant cycles in the first step, and to avoid the costly final step.

Ad-hoc algorithm

1-3. Same as Horton's algorithm except that it generates all shortest cycles.

4. Check aromaticity for each cycle in the order of the length.

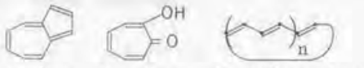
The ad-hoc algorithm applies the check for aromaticity for cycles which are not relevant, but it is harmless for our purpose. Finding all shortest cycles is achieved by a minor modification of Dijkstra's shortest path algorithm. So, the first and the second step takes $O(V^2 \log V + V^2 E)$ time. The time taken by the third step depends on the number of generated cycles. This method takes $O(V^2 \log V + V^2 E + K \log K)$, in which K is the number of shortest cycles.

Another strategy for detecting aromatic cycles is to use the algorithm for finding all simple cycles, instead of relevant cycles. Except for a highly polymeric structure such as nanotube or fullerene, the number of simple cycles in a molecular graph is small. The fast algorithm based on depth-first search with backtrack is proposed by Johnson [50]. For detail, readers are referred to the survey by Mateti and Deo [61].

3.3.4 Discussion

Applicability

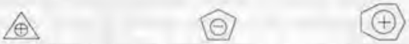
It is important to note which molecules are correctly found aromatic, and which are not, by our definition. Obviously, benzenoid aromatic structures are correctly found aromatic. Among nonbenzenoid aromatic compounds, some are stable, while others are not. Stable ones are found aromatic, because they satisfy Hückel rule. The number of π electrons for non-ionic stable compounds is correctly counted by our definition, even when the cycle is irregular (Fig 3.3). Our definition finds the aromaticity of heme too.



azulene tropolone $[4n+2]$ annulene

Figure 3.3: Stable nonbenzenoid aromatic compounds. They do not appear in metabolism, but our definition of aromaticity works for them too.

On the other hand, non-ionic unstable compounds, such as pentalene, are not found aromatic. Ionic stable compounds, whose number of π electrons depends on the ionic charge, are not found aromatic either (Fig 3.4). However, they do not appear in basic metabolism, and we consider that our definition perfectly works for bacterial metabolism.



cyclopropenium ion cyclopentadienide ion tropylium ion

Figure 3.4: Stable ionic aromatic compounds. They do not appear in metabolism, and our definition of aromaticity does not work for them either.

History

In chemical computation, the ring perception problem has been considered for long [32, 35, 12, 37], and the problem of finding shortest cycles is formalized as SSSR (Smallest Set of Smallest Rings) problem. The tradition of proving by examples and the definition of SSSR, which is not canonical, complicate the problem, but we consider that the related theoretical problems have been solved in many cases [50, 96]. Although the computation time will be exponential in theory, the enumeration of all relevant cycles has an important chemical application, such as predicting a boiling or freezing point of molecules. Therefore, an efficient algorithm like Horton's or Vismara's is quite important in a practical viewpoint.

3.4 Topological Equivalence

3.4.1 Background

In a molecular graph, two nodes u and v are *topologically equivalent* if there is an automorphism mapping from u to v. In chemistry terms, they must share the same surrounding structures: atoms, bonds, and their configuration. The detection of such sites is important for checking whether a given carbon atom is chiral, or for analyzing peaks in spectral analyses. Unfortunately, the problem of automorphism partitioning is known to be equivalent to the graph-isomorphism problem. Since the latter problem is unlikely to be solvable in polynomial-time, the detection of topologically equivalent nodes also seems intractable [54]. Moreover, the problem becomes even more complex when chirality is considered.

In chemical computation, Morgan method [69] has been the mainstream solution for the problem. It is a procedure to partition nodes by iteratively calculating their integer labels, so that topologically different nodes obtain different labels.[8] The original method is simple, but incomplete. Because it does not identify structures which fall outside of its detection power, all positions with the same label must be checked for the correctness, using an additional time-consuming procedure.

In pursuit of a more elegant method, many different approaches were proposed, but were followed by as much disproof [49, 91, 23, 98, 79, 29, 75, 82, 60, 34]. The history of proposal and disproof comes from the tradition of proving by examples; most algorithms were reported to work on a handful of examples, and their actual computational power remained unknown. It is also hard to compare the detection power or the range of completeness of those algorithms.

The detection of topologically equivalent sites is closely related with the normalization, or finding the unique name, of a molecular graph. In practice, the augmented version of Morgan method by Wipke and Dyott (SEMA method) has been the best accepted method [106]. Recently, Faulon showed a proven polynomial-time algorithm for finding the automorphism partitioning of a planar molecule [36]. His algorithm is quite similar to the one for the isomorphism testing for planar graph by Hopcroft and Tarjan [44], and does not consider chirality. Independently, Akutsu published a proven algorithm for normalizing a planar molecule with chiral information [5]. His algorithm is a combination of SEMA method and Hopcroft and Tarjan's algorithm.

Definitions

If all vertices have the same degree k, then the graph is *regular* of degree k or *k-regular*. A graph G is called *strongly regular* if it is regular of degree k, and if any two adjacent (non-adjacent) vertices are adjacent to exactly λ (respectively μ) other vertices. Four integers (n, k, λ, μ) are the parameters of a strongly regular graph.

Given two graphs $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$, a one-to-one mapping σ of V_1 onto V_2 is an isomorphism if and only if $(u, v) \in E_1 \Leftrightarrow (\sigma u, \sigma v) \in E_2$ $(\forall u, v \in V_1)$. If there exists an isomorphism mapping V_1 into V_2, then G_1 and G_2 are *isomorphic*, denoted $G_1 \cong G_2$. An *invariant* is the parameters of a graph preserved under isomorphism. An invariant I is *complete* if $G_1 \cong G_2 \Rightarrow I(G_1) = I(G_2)$, otherwise it is *incomplete*.

A one-to-one mapping σ of V onto V is an automorphism of G if and only if $(u, v) \in E \Leftrightarrow (\sigma u, \sigma v) \in E$ $(\forall u, v \in V)$. If there exists an automorphism mapping u into v, then u and v are *similar*, denoted $u \sim v$. A graph is *transitive* if $u \sim v$ $(\forall x, y \in V)$. The

[8] In chemical computation, it is called a canonical labeling or canonical ordering.

automorphism partition of G is the partition of V induced by the equivalence relation $\sim$. A partitioned set of nodes is called a *cell* of the partition. A *vertex invariant* or *v-invariant* of a graph $G(V, E)$, denoted i_G, is a function which labels vertices with integers: $i_G : V \rightarrow \mathbb{N}$ and $u \sim v \Rightarrow i_G(u) = i_G(v)$ $(\forall u, v \in V)$. The integer value $i_G(u)$ is a *label* of u. The *complement* $\bar{G}(V, F)$ of the graph $G(V, E)$ satisfies $F = \{(u, v) \mid u \neq v \wedge (u, v) \notin E\}$. A v-invariant i_G is *strongly incomplete* if $i_G(u) = i_G(v)$ and $i_{\bar{G}}(u) = i_{\bar{G}}(v)$ $(\forall u, v \in V)$.

3.4.2 Existing Methods

Automorphism Partition

Given a molecular graph G, Morgan method computes its v-invariant, i.e. an integer label for each node, in the following process (Fig 3.5).

MORGAN ALGORITHM

1. For each node, set an integer label considering its atomic number, degree, and type of bond (single, double, and others). The initial label for a node v is denoted l_v. Partition the set $S = \{l_v \mid v \in G\}$ into cells $\{C_1, \ldots C_k\}$, in which $(\forall v, w \in C_i)$ $l_v = l_w$ for each C_i $(1 \leq i \leq k)$.
2. For each node v, update l_v:
   (a) Duplicate labels of all adjacent nodes of v.
   (b) Add those labels to l_v.
3. Partition S into cells. If the number of cells increases by the update, repeat Step 2. Otherwise stop.

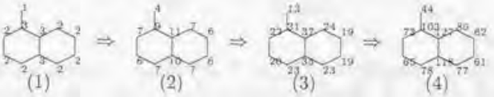


Figure 3.5: Example run of Morgan algorithm. (1) Each node is assigned a label corresponding to its number of connectivity. (2) For each node, add labels of its adjacent nodes. (3) This step is repeated until no classification is achieved. (4) Classification stops. Each node has a unique label.

Step 2 takes E time, and the set S must be sorted for classification in each iteration. The iteration may be repeated V times, and the total time-complexity is bounded by $O(V^2 \log V + VE)$.

The simple addition of labels may produce the coincidental match in the added results for topologically different nodes. The Fig 3.6 is such an example [91]. Two nodes v_2 and v_{14} will not be distinguished by Morgan algorithm, although they are different because of the single nitrogen v_{18}. This result is intuitively understood when the structure is rotated by 120°; the three nodes, v_2, v_{14}, and v_{18} share the same position after the rotation.

The artifact resulting from the addition is avoidable by using a multiset of cells (without integer labels) [25], a one-to-one MAP function [36], or a multiplication of prime numbers [105], instead of the simple addition of labels.

Example of MAP function (Elements $x_1 \cdots x_n$ are sorted.):
$\mathrm{MAP}(x_1, x_2, \cdots, x_n) = x_1 + x_2 M + \cdots + x_n M^{n-1} \quad \forall n (x_n < M)$

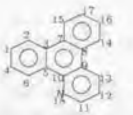


Figure 3.6: Nodes v_2 and v_{14} will not be distinguished by Morgan algorithm.

Even if the addition is replaced with some alternative operation, however, the essence of Morgan algorithm is the same as the terminal connection partitioning by Corneil and Gotlieb [25]. Their partitioning powers are the same, and they do not refine the partition of a large class of graphs including regular non-transitive graphs. The Fig 3.7 shows two negative examples. Left example is regular, but a non-regular graph as in the right also becomes a negative example.

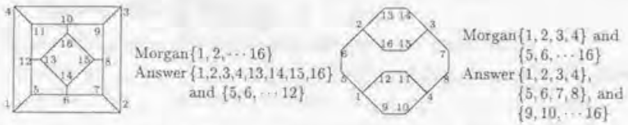


Figure 3.7: Morgan algorithm fails in these graphs.

Observation. Morgan algorithm does not partition regular graphs.

Normalization

The normal form of a molecular graph $G(V, E, f_V, f_E)$ is a string describing the information in V, E, f_V and f_E. Let $P_V = \langle v_1, \ldots v_n \rangle$ be an order among nodes. For node v, let $str(v)$ be a string expression of $f_V(v)$. For edge $e = \{v_i, v_j\}$ $(i > j)$, let $str(e)$ be a string expression of i, j, and $f_E(e)$. The order of edges P_E is defined after P_V: $\{v_1, w_1\} > \{v_2, w_2\} \Leftrightarrow v_1 > v_2 \vee (v_1 = v_2 \wedge w_1 > w_2)$. The brute-force method to generate the normal form of G is to: (1) concatenate $str(v)$ for all $v \in V$ in the order of P_V, and $str(e)$ for all $e \in E$ in the order of P_E, using an appropriate separator; (2) repeat Step 1 for all possible orderings in P_V; (3) select the lexicographically minimum name among all the generated.

The number of possible P_V may be exponential in the brute-force method, but in practice, it will be significantly reduced for a non-symmetric graph, using the result of Morgan algorithm.

NORMALIZING ALGORITHM

1. Apply the breadth-first search in the order of labels from a node with the minimum label. When two or more adjacent nodes have the same label in the search, generate the visiting order for all possible combinations. Let this order be P_V.

2. For each visiting order P_V, output $str(v)$ and $str(e)$ as is explained above.

3. If more than one name is generated, choose the lexicographically minimum one.

It is necessary to generate (tentative) normal forms for all possible orderings among nodes of the same v-invariant, because the partitioning process is not complete; for any pair of nodes v and w, $v \not\sim w$ if $l_v \neq l_w$, but $l_v = l_w$ does not imply $v \sim w$. Because of this incompleteness, the total number of orderings will be the same as the brute-force method for a symmetric structure.

The better idea was suggested by Corneil and Gotlieb already in 1970 [25]. If one node in a certain cell is assumed to be different from the other nodes in the same cell, it will introduce a finer partitioning to the other remaining cells. If two nodes v and w are in fact similar, the assumption that v is unique and the same assumption for w should produce the same partitioning result. This process is called Vertex Quotient (VQ).

VQ PARTITION [25]

1. Apply Morgan algorithm.
2. For each cell of two or more nodes $S = \{v_1, \cdots, v_k\}$:
   - (a) Remove one node v_i from S, and consider it as a unique cell.
   - (b) Apply Morgan algorithm, and obtain a new set of cells P_i on the assumption that v_i is different from nodes in $S \backslash \{v_i\}$.
   - (c) Return v_i to S, and repeat this process for all nodes in S.
   - (d) If $P_i \neq P_j$ for some $1 \leq i < j \leq k$, partition S to sub-cells according to P_i $(1 \leq i \leq k)$, and return to Step 1.

Corneil and Gotlieb showed that the result of VQ process is the automorphism partition of G, if G does not contain strongly regular non-transitive graph. For such graphs, there is no need to consider all possible orderings, if Normalizing algorithm is run after the application of VQ partition. That is, when two or more adjacent nodes have the same label in the breadth-first search in Normalizing algorithm, choose an arbitrary one and proceed the search, then the result will be a unique visiting order among nodes.

Definition. The relation $P \prec Q$ between two algorithms denotes that the partition achieved by algorithm Q is finer than that by algorithm P. □

The above argument tells (Morgan algorithm)$\prec$(Morgan algorithm with VQ partition). Although VQ partition is powerful, it directly calls Morgan algorithm. The partitioning power of Morgan method heavily affects the total computation time in practice, because the VQ partition repeatedly calls Morgan method for each tentatively similar node. In order to minimize the number of calls of Morgan method, we introduce another v-invariant to be replaced with Morgan algorithm.

3.4.3 Partition by Distance

Matrix algorithm

Given a graph G, its *distance matrix* is

$$M = \begin{bmatrix} d_{11} & d_{12} & \ldots & d_{1n} \\ - & d_{22} & \ldots & d_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ - & - & \ldots & d_{nn} \end{bmatrix}$$

in which d_{ij} $(1 \leq i \leq j \leq n)$ is the distance between nodes v_i and v_j, and n is the number of nodes in G. The diagonal line is zeroed: $d_{ii} = 0$. We propose a partitioning using the distance matrix.

MATRIX ALGORITHM

1. same as Morgan algorithm
2. For each node v_i, update l_{v_i}:
    (a) Duplicate labels of all nodes in G.
    (b) For all node v_j $(1 \leq j \leq n)$, add $(v_j \cdot d_{ij})$ to l_{v_i}.
3. same as Morgan algorithm

Addition is used to contrast the algorithm with Morgan's, but a multiset of cells or a MAP function is recommended in practice. The distances from a single node is computed by a breadth-first search, and the matrix is computed in $O(V^2 + VE)$ time. The number of iterations required in the worst case is hard to estimate for Matrix algorithm, but if the iteration is bounded by $\log V$, the time complexity of Matrix algorithm is the same as Morgan algorithm. In practice, a significant acceleration is achieved, because it is used in combination with VQ partition. Note that, in VQ partition, the number of recalls of Matrix (or Morgan) algorithm depends on the partitioning power of Matrix (or Morgan) algorithm.

Observation. (Morgan algorithm) $\prec$ (Matrix algorithm) $\prec$ (Morgan with VQ) $=$ (Matrix with VQ)

The v-invariant computed in Matrix algorithm considers all the nodes in different distances from each node. If only adjacent nodes are considered, Matrix algorithm is the same as Morgan algorithm. Their powers are not equal, because there is a class of graphs which can be correctly partitioned by Matrix algorithm, but not by Morgan algorithm. Fig 3.7 is such an example.

The partition by Matrix algorithm with VQ partition is finer than the one without VQ partition. VQ partition also considers the classification based on the effect of a single node to others as in Matrix algorithm, but it keeps the partition result for each tentatively similar nodes, and later compares them. On the other hand, Matrix algorithm averages the result of partitions based on each node. Therefore, VQ partition produces finer partition than Matrix algorithm alone. Two graphs in Fig 3.8 have the same distance matrices, and therefore look same under Matrix algorithm alone.

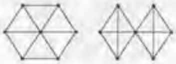


Figure 3.8: Two graphs of the same distance matrix

3.4.4 Discussion

Related Work

Recently, we found a similar work by Schmidt and Druffel [88], which used the distance matrix to compute the initial partition of the automorphism group. Their purpose was to reduce the search space in testing graph isomorphism with backtrack, but they did not refine the partition iteratively with VQ process. In this respect, our work remains useful, because the improvement in the initial partition with VQ process is applicable for their problem too.

Completeness

The invariant based on the distance matrix is also known to be incomplete.

Theorem 3.4.1 (Corneil and Kirkpatrick [26]) *Distance invariant i_G is strongly incomplete.*

In fact, stronger incompleteness was proven for i_G. For any graph, there is a polynomial-time transformation mapping any graph to a regular non-transitive graph, for which i_G fails [26].

In chemical application, the use of Matrix algorithm with VQ partition will be justified, if a strongly regular non-transitive graph does not appear in chemical molecules. It is highly probable, because 1a strongly regular non-transitive graph requires at least twenty-five vertices.[9] The investigation on the type of graph for which the automorphism group is hard to find has been of theoretical interest, but it is almost certain that the VQ process is complete for biochemical molecules.

Other Algorithms

VQ partition has been unnoticed in the community of chemical computation, in which many alternative approaches were proposed but were countered with negative examples. All such negative examples having appeared so far are not strongly regular non-transitive, and in fact, they are correctly partitioned by Matrix algorithm alone. Therefore, those negative examples are correctly partitioned with the algorithm by Corneil and Gotlieb. Note that Matrix algorithm is a practical acceleration of Morgan's algorithm.

Another strategy is to restrict the class of graph. Theoretically efficient algorithms for graph isomorphism problem or automorphism partition exist for a various class of graphs: planar, tree, partial k-tree, almost-tree, bounded valence, or bounded average genus. Some works are far from being practical, but the application of these algorithms is an alternative approach for the problem [36].

[9] There are several. (Personal communication with Dr. E. Spence at U Glasgow)

3.5 Chirality

3.5.1 Background

Conventional Notation

A carbon node is *chiral* or *asymmetric* when it joints four different chemical groups, forming a tetrahedron.[10] Four branches can take only two configurations, denoted using R and S in organic chemistry, or D and L in biochemistry. The definition of D-carbon is that it has the same configuration as the central carbon node of biologically active glyceraldehyde (`C(O)C(O)C=O`). The configuration of L-carbon is its mirror image. D (or L) can be used as a prefix of a molecular name, meaning that the molecule is derived from D-glyceraldehyde (or L resp.). Thus, D-glucose is D, because its fifth carbon node, derived from glyceraldehyde, retains D configuration.

R,S-system has a more formal definition. Given a chiral carbon node, its connected chemical groups are arranged so that the group of the least atomic weight goes backward. When several branches tie, the second next nodes are compared. When the other three groups are arranged in the increasing order clockwise, it is R-carbon, otherwise S. For example, the order of frequently used chemical groups is:

$$\text{-SH} > \text{-OH} > \text{-NH}_2 > \text{-COOH}^{11} > \text{-CHO} > \text{-CH}_3 > \text{-H}.$$

Note that only D,L-system is used as a prefix of molecular names. R,S-system always specifies the configuration of a single carbon node.

It is worth noting that the ambiguous D,L definition is sufficient for biochemistry; molecules in metabolism is simple enough to handle their chirality by human hand with an ad-hoc naming. The natural phenomenon that L configuration dominates in living organisms also helped the acceptance of the ambiguous definition.

Theoretically, the choice of symbols, RS or DL, makes no difference in its treatment. In this paper, two configurations are denoted with + and − signs. Another way to define chirality is, 'a node is chiral when when its two configurations are not superposable'. In this paper, a node is *chiral* when to it has a unique configuration. A node is *non-chiral* when its two configurations are superposable.

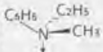
SMILES for Chirality

In SMILES, the configuration of a chiral node's branches is specified using either `@@` or `@`. By default, four branches are given the left to right order in the notation. When they are arranged so that the leftmost group comes to the front, and if the other three are arranged clockwise, `@@` is attached to the notation of the carbon node. Otherwise `@` is attached. These symbols may be intentionally omitted. In such a case, the notation denotes both configurations, i.e. the mixture of both chirality, called *racemic* mixture. Note that carbon nodes which are not chiral also have no `@` symbols.

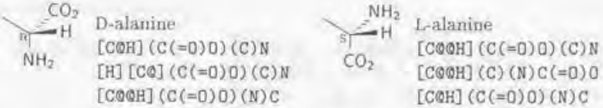
3.5.2 Determining Normal Form

In Normalizing algorithm, the information of node v was output as $str(v)$, which is a string expression of $f_V(v)$. For a carbon node only, we modify its $str(v)$ to additionally

[10] More precisely, this chirality is called *central* chirality. In chemistry, *axial* chirality and *planar* chirality are also important. Moreover, other elements may form a central chirality (right figure). We consider only the central chirality of carbon nodes, because it is most important in biochemistry.



[11] Carboxyl group is denoted -CO_2 hereafter.



D-alanine
```
[C@H](C(=O)O)(C)N
[H][C@](C(=O)O)(C)N
[C@@H](C(=O)O)(N)C
```

L-alanine
```
[C@@H](C(=O)O)(C)N
[C@@H](C)(N)C(=O)O
[C@H](C(=O)O)(N)C
```

Figure 3.9: SMILES examples for a chiral node

contain the string expression of either one of three configurations: [normal], [+], or [−]. The string expression of a chiral node contains either [+] or [−], and that of a non-chiral node contains [normal]. The main topic of this section is how to choose one configuration for each carbon node according to the user-specified SMILES symbols.

In this section, we assume that a user makes no input error and that a complete v-invariant is available. For each chiral node in a molecular graph, a user specifies @@, @, or no symbol (racemic mixture) in SMILES. Since there is no input error, its interpretation is as follows.

- If a user put neither @@ nor @ symbol to a carbon node, its configuration is [normal].
- Otherwise, its configuration is either [+] or [−].

Theoretical Aspects

With a complete v-invariant, it is possible to find the normal form of any molecular graph with chirality, at least in theory, because there is a simple transformation of the chiral configuration into a normal graph [4]. If Normalizing algorithm is applied after this transformation, it will generate the unique normal form of the given molecular graph.

This transformation, however, introduces 40 nodes for each chiral carbon atom. For example, the total number of nodes for glucose, originally 12 nodes, will become 168 after the transformation. Considering the size of the expansion, it is not a good strategy to compute the normal form of the transformed graph with Normalizing algorithm in Section 3.4. Also, the representation of racemic mixture is not straightforward in the transformation method; the simplest way is to prepare several different graphs for each configuration. In reality, almost all structures appearing in basic metabolism are not highly symmetric. Therefore, we will show an ad-hoc but simpler method.

Our ad-hoc strategy uses adjacency lists as in SEMA method. It is not complete, but works efficiently and sufficiently enough for biochemistry. Its advantage is that it treats chirality as an additional property of the graph structure. This patched-up method is suitable for biochemical application, because chirality should be considered in some enzymatic reactions (chiral specificity), while others proceed regardless of chiral information.

Let i_G be a complete v-invariant for molecular graph $G(V, E, f_V, f_E)$, computed without considering @@ or @ symbol. Assume that carbon node $v \in V$ has four adjacent nodes $\{w_1, w_2, w_3, w_4\}$. If $i_G(w_i) \neq i_G(w_j)$ $(\forall i, j)(1 \leq i < j \leq 4)$, then the chirality of v is implied by the graph structure, because its four adjacent nodes are not automorphic. On the other hand, $i_G(w_i) = i_G(w_j)$ for some i and j means that the chirality of v is implied by the chirality of other nodes.

We first discuss the former case, and explain how [+] or [−] is chosen according to the input @@ and @ symbols. Then we discuss the latter case, in which the chiral information must be integrated in the automorphism partitioning process. In the following argument, we assume that node v is a chiral node with @@ or @ symbol in SMILES.

Chirality based on Topology

Consider four adjacent nodes $\{w_1, w_2, w_3, w_4\}$ of node v, and assume $i_G(w_i) \neq i_G(w_j)$ $(\forall i, j)$ $(1 \leq i < j \leq 4)$. In SMILES notation, `@@` or `@` symbol is attached to a chiral node according to the written order of its branches. Note that the `@@` (or `@`) symbol switches to the other, when the order two adjacent branches are exchanged in the input (Fig 3.9). Without loss of generality, we assume that the adjacent nodes are written in order $\{w_1, w_2, w_3, w_4\}$ in SMILES. We write $i_G(w_i)$ as l_i, and then the v-invariants for adjacent nodes form a list $\{l_1, l_2, l_3, l_4\}$. The exchange between adjacent branches in the input, say, between the first and the second, produces the list $\{l_2, l_1, l_3, l_4\}$, and this list signifies the other configuration. That is, an even number of permutations in the list retains the configuration of chirality, while odd permutations change it. Therefore, the following twelve patterns share the same configuration.

$$\begin{array}{cccccc} l_1,l_2,l_3,l_4 & l_1,l_3,l_4,l_2 & l_1,l_4,l_2,l_3 & l_2,l_3,l_1,l_4 & l_2,l_1,l_4,l_3 & l_2,l_4,l_3,l_1 \\ l_3,l_1,l_2,l_4 & l_3,l_2,l_4,l_1 & l_3,l_4,l_1,l_2 & l_4,l_1,l_3,l_2 & l_4,l_3,l_2,l_1 & l_4,l_2,l_1,l_3 \end{array}$$

Definition. For each chiral node v, let l_1, l_2, l_3, l_4 be the four labels, computed by a complete v-invariant, of its adjacent nodes written in SMILES in order $\{w_1, w_2, w_3, w_4\}$. We assume none of them are equal. If the above twelve patterns includes the sorted order of the labels, the configuration of v is [+]. Otherwise it is [−]. □

This method using an adjacency list is first introduced in SEMA method [106].

Chirality based on Other Chiral Nodes

If any two adjacent nodes of node v share the same label, i.e. $l_i = l_j$ $(1 \leq i < j \leq 4)$, then the chirality of v depends on the chirality of other nodes. Note that the `@` (or `@@`) symbol indicates v is in fact chiral, because there is no input error. One such example in basic metabolism is arabitol (`C(O)C(O)C(O)C(O)C(O)` ; Fig 3.10). The chirality of its central carbon node (numbered 3 in the figure) depends on the chirality of two siding carbon nodes (numbered 2 and 4).[12]

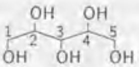


Figure 3.10: Arabitol

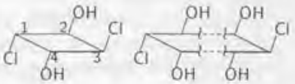


Figure 3.11: Symmetric structures with p-chiral nodes

Since the configuration of the node 2 and 4 become known during the automorphism partitioning process, it is possible, in the following partitioning process, to assign new

[12] Arabitol is the only one structure having this type of symmetry among about 1000 basic compounds in metabolism. In theory, arabitol can take two patterns in its configuration, but in nature, only D-arabitol exists.

labels to the carbon nodes of [+] and [−] configuration. If [+] and [−] nodes are given different labels from the one for nodes of unknown or [normal] configuration. That is, if they are regarded as different atomic elements from carbon, then the configuration of other chiral node will be determined by the same procedure, which is the list of v-invariant labels. This observation leads to the following operation.

Definition. *Chiral conversion* is a procedure to assign new initial labels to carbon nodes whose configuration is newly detected [+] or [−]. The labels are different from those for carbon nodes of unknown or [normal] configuration. □

Note that the chiral conversion reveals the chirality of nodes at the symmetric center of the molecular graph. Let us consider the case of arabitol again. The first chiral conversion provides new labels to node 2 and 4 simultaneously. The configuration of node 3 is then determined, and it will obtain a new label in the second chiral conversion. The conversion is repeated until no further partitioning of nodes is achieved.

There may be chiral nodes whose configuration cannot be determined through the chiral conversion. We call these nodes *pseudo-chiral* or *p-chiral*.

Definition. A *p-chiral* node is a chiral node whose configuration has not been determined by the automorphism partitioning with chiral conversion. □

The four numbered nodes in Fig 3.11 (left) are all p-chiral.[13] Note that p-chiral nodes are chiral nodes which are not yet assigned their configuration. It's chirality will be revealed when the chirality of other nodes is determined, which means, there must be at least two other p-chiral nodes in the structure. If there are less than two p-chiral nodes, their automorphic branches will not be differentiated by assigning the chiral configuration to p-chiral nodes.

One way to determine their configuration is to tentatively assign either one configuration to some p-chiral node. In Fig 3.11, if the node 1 is assumed to be [+], three other nodes will obtain either [+] or [−] configuration, accordingly. On the other hand, if the node 1 is [−], the other nodes obtain different configuration, which is not necessarily the reverse of the previously assigned configuration. Since the pattern of configuration is not predictable [14], a number of configuration patterns must be tested to find the normal form of the structure. The similar observation is also found in the work by Agarwal [3, 2]. In the case of Fig 3.11 (left), suppose node 1 and 3 share the minimum label in Normalizing algorithm. We need to test two cases: in one case, the node 1 becomes the root of the breadth-first search and is assigned [+], and in the other, the node 3 becomes the root and is [+].

From a practical viewpoint, we do not pursue the problem of efficiently finding the configuration producing the normal form, firstly because p-chiral nodes do not appear in basic compounds in metabolism, and secondly because users frequently make input errors. When p-chiral nodes exist in input data, it is highly probable that the chiral specification is an error. Moreover, the procedure to correctly treat p-chiral nodes with errors is time-consuming. The next subsection explains this problem in detail.

3.5.3 Input Error

So far we have assumed that there is no input error, but in reality, users often make mistakes. The typical error is to:

[13] We assume that a user specifies @ (or @@) symbol in its SMILES notation.

[14] Note that the decision of [+] or [−] depends on v-invariant labels, and the labels depend on atoms, bonds, and other parameters.

- forget a chiral specification;
- attach a spurious chiral specification.

First of all, the former error cannot be detected, because the omission of the input configuration is interpreted as racemic mixture. The detection of the latter error is possible, if there is no p-chiral node.

Observation. For structures without p-chiral nodes, spurious chiral configuration can be detected.

PROOF. Let v be a node with `@@` (or `@`) symbol whose adjacent nodes w_1 and w_2 satisfy $i_G(w_1) = i_G(w_2)$, when `@@` and `@` symbols are not considered (i.e. without chiral conversion). If v is not p-chiral, both $i_G(w_1)$ and $i_G(w_2)$ must change and differ after (possibly multiple execution of) the chiral conversion. Otherwise, the branch structure does not contain chiral nodes, and the chiral symbol at v is an input error. □

In general, the detection of spurious chiral signs is time-consuming, because all the possible configurations of carbon nodes must be checked to achieve the complete detection.

Let us assume that p-chiral nodes may exist in the structure. If $i_G(w_1)$ and $i_G(w_2)$ do not differ after the chiral conversion, v is a p-chiral node, or the chiral symbol of v is an input error.

Assume that v is in fact a p-chiral node. In this case, when v is tentatively assigned [+] or [−], the branches of other p-chiral nodes will be differentiated, i.e. the previously equal v-invariant labels of their adjacent nodes become different by the assignment. The previously equal labels of v's adjacent nodes should become different too. On the other hand, assume that the chiral symbol is an input error. If v is tentatively assigned [+] or [−], the branches of other p-chiral nodes is differentiated, but the previously equal labels of v's adjacent nodes do not become different. Thus, it is possible to detect input errors for the nodes with chiral signs.

The difficulty arises when their tentative assignment differentiates the adjacent labels of a carbon node without any chiral symbol. The differentiation indicates an input error in some cases, while in others, it does not lead to an error. In the left structure of Fig 3.11, for any of the four carbon atoms to be chiral, the chiral sign for all of them must be specified. If any of chiral sign is omitted, then the remaining signs become input errors. This argument does not hold in the right structure of Fig 3.11.

In order to find input errors for the nodes without chiral signs, all their possible configurations need to be checked. Since the omission of chiral signs is a shorthand of writing both configurations (racemic mixture), the number of possible configurations can be big even for a simple compound in metabolism.

3.5.4 Discussion

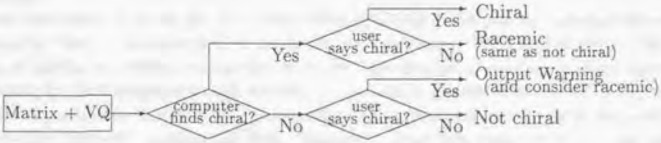
History

Although the treatment of chirality is an important issue in chemical computation, especially in biochemical application, the general scheme to handle chiral information has not been established yet. SEMA method is widely accepted for the generation of normal forms, but it does not support the description of racemic mixture.

Implementation

In contrast to the high frequency of input errors, p-chiral nodes very rarely appear in basic metabolism. A rarity of complex chiral structures is also understood from the widespread use of the naïvely defined D,L-system.

In practice, it is more helpful to support an error detection than to support the efficient algorithm for finding normal forms with p-chiral nodes. Therefore, in the actual implementation, we suggest the following process: (1) Apply Matrix and VQ process with chiral conversion; (2) output warnings for p-chiral nodes, and consider them as [normal] carbon. Thus, different configurations of p-chiral nodes will be ignored. This strategy is not appropriate for general chemical database, because there are many structures with p-chiral nodes. However, it is sufficient for biochemical application.



3.6 Graph Matching

3.6.1 Background

Finding a maximal common substructure between compounds is one of the oldest problems in chemical computation.[15] The problem drew attention not only of chemists but of computer scientists, because it can be formulated as finding a maximal common subgraph (MCS), a problem which is harder than the subgraph isomorphism problem. As can be supposed, the cost for its optimal solution is prohibitively expensive; the problem is NP-hard [40] in general, and its approximation is also one of the hardest problems to achieve [52].

There are some ways to get over this difficulty. One is to use suboptimal solutions computed by Monte-Carlo method or the linear programming. Another is to use a different measure of similarity. When we say two molecules are similar, the measure of similarity is not necessarily their graph-oriented structures. In fact, a quantum-mechanical similarity, such as comparing the electrostatic potentials, is the most commonly used measure for the similarity between compounds [76]. However, there are cases in which the exact solution of the graph-theoretic similarity is necessary. One such example is our problem, the analysis of enzymatic activity; in catalysis, an enzyme changes a small part of the substrate structure, and leaves its carbon backbone untouched (Fig. 3.12). In most cases, this unmodified part equals to the MCS between the substrate and the product.

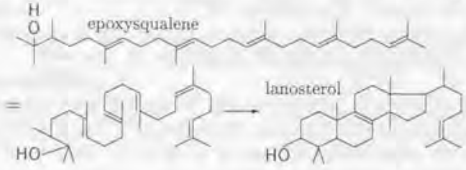


Figure 3.12: lanosterol synthase: Epoxysqualene becomes lanosterol.

In computer science, the practical algorithm for the subgraph isomorphism problem has been Ullmann's method [99]. The backtracking counterpart was introduced by McGregor [63]. Given two graphs, Ullmann's method orderly generates all possible matrices mapping nodes in the smaller graph to those in the larger one, to test their isomorphism. Some other theoretical works have been also reported [6, 95], but they are too slow for actual implementation. Recently, GMA algorithm is presented by Xu [107] for the subgraph isomorphism problem. GMA tries to place the smaller graph onto the larger one node by node using backtrack. It is reported to work better for the chemical application than the commercially implemented substructure search system, which is often Ullmann's method.

Finding MCS is even harder than this subgraph isomorphism problem. One practical application software is EMCSS by Wang and Zhou [101], which uses the genetic algorithm. In contrast to this heuristic approach, the exact solution should be computed brute-force in our case, because finding a mapping in enzymatic reactions is considered a special case of general MCS problem. Specifically, we can use the following characteristics of enzymatic reactions.

[15] For the chemists' perspective on the similarity of compounds, see the review by Stobaugh [94] or by Rouvray [81].

- Compounds in basic metabolism usually consist of up to fifty atoms.
  This boundary also seems to be the frequently required range for the substructure search of molecular structures.

- Enzymes usually preserve cyclic structures.
  An overall rearrangement of atoms never occur in catalysis; only a partial structure is modified in most cases. For example, Fig 3.12 is one of the most complex reactions in metabolism.

- The number of maximal common substructures is not exponential.
  Input and output compounds share the structural similarity, and the number of their MCS is not exponential.

Considering these properties, we propose a new algorithm for MCS. This section is devoted to the analysis of metabolic reactions, and does not consider theoretical aspects of the algorithm.

3.6.2 Finding MCS

Definitions

The *size* of a graph $G(V, E)$ is $|V|$, and is denoted $size(G)$. Given two molecular graphs $G(V, E, f_V, f_E)$ and $G'(V', E', f'_V, f'_E)$, G' is a *subgraph* of G (denoted $G' \subseteq G$), if there is a one-to-one mapping σ such that $(u, v) \in E' \Leftrightarrow (\sigma u, \sigma v) \in E \wedge f'_V(u) = f_V(\sigma u) \wedge f'_V(v) = f_V(\sigma v)$ $(\forall u, v \in V')$. Note that the equality of edge-colors, $f'_E((u, v)) = f_E((\sigma u, \sigma v))$, is not required in this definition.

Given two molecular graphs G and G' whose nodes are linearly ordered, the maximal common subgraph H of G and G' is a directed molecular graph maximizing $size(H)$ and satisfying $H \subseteq G \wedge H \subseteq G'$. A node in H corresponds to a pair of nodes, one node u from G and the other v from G', and is called a *node-pair*, denoted $\langle u, v \rangle$. An edge $e = (\langle x, y \rangle, \langle w, z \rangle)$ in H is directed, and satisfies $(x < w \vee (x = w \wedge y < z))$. Given a common subgraph H of G and G', the set of nodes from G is denoted $L(H)$, i.e. left-side nodes in H, and the set of nodes from G' is denoted $R(H)$. The *difference* between G and H is a subgraph of G induced by the nodes $\{v \mid v \in G \wedge v \notin R(H)$ or $L(H)\}$, and is denoted $G \setminus H$.

The order of nodes in G and G' introduces a total order among node-pairs. The minimum node-pair in H is denoted $min(H)$. A set of *extension-pairs* of H is a set of node-pairs which can be incorporated to H for its enlargement, i.e. $\{\langle u, v \rangle \mid u \notin L(H) \wedge v \notin R(H) \wedge (x, u) \in G \wedge (y, v) \in G' \ (\exists y \in R(H), \exists x \in L(H))\}$. It is denoted $ext(H)$.

Brute-force method

Problem 2. Maximal Common Subgraph (MCS)
Instance: Two graphs G and G'.
Question: Find one of their MCS.

The basic idea to find MCS is to expand the common subgraph stepwise from a single node-pair. We can restrict the order of the newly added node-pair to be always increasing, i.e. the order of the added node-pair must be larger than that of any other node-pair in the common subgraph (Fig 3.13). If necessary, multiples of common subgraphs merge at their lastly expanded node-pair (Fig 3.14). These operations in generating all the possible common subgraphs are written as functions:

- *add* (C, p) enlarges a subgraph C by adding $p \in ext(C)$;
- $merge(C, C')$ merges two common subgraphs C and C', if they share only the lastly added pair $\langle u, v\rangle$, and share no other nodes. In other words, $(\{u\} = L(C) \cap L(C')) \wedge (\{v\} = R(C) \cap R(C'))$.

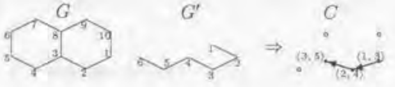


Figure 3.13: Function *add*. Thick arrows show common subgraph C between G and G'. Their direction is determined by the order of node-pairs. White points are $ext(C) = \{\langle 4, 6\rangle\langle 8, 6\rangle\langle 10, 2\rangle\}$, or the set of candidate nodes. Any of them can be *add*ed to enlarge C.

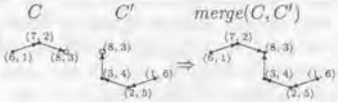


Figure 3.14: Function *merge*. C and C' are common subgraphs between G and G' in Fig 3.13. They can *merge* at $\langle 8, 3\rangle$ to form the common subgraph isomorphic to G'.

Brute-force algorithm

1. Set $M = \{\}$.
2. Set $Q = \{$ all node-pairs of size 1 between G and $G'\}$.
3. If $Q = \phi$, halt; otherwise choose $C = \{c_1, c_2, \cdots, c_n\}$ such that $min(c_i) = min(c_j)$ $(1 \leq i < j \leq n)$ and that no subgraph in Q contains smaller node than $min(c_i)$.
4. Set $Q = Q - C$.
5. For all pair of subgraphs $c_i, c_j \in C$ which can be *merged*, set $C = C \cup \{merge(c_i, c_j)\}$.
6. For each $c \in C$:
    (a) If $size(c) > size(m)$ for any $m \in M$, set $M = \{c\}$.
    (b) If $size(c) = size(m)$, set $M = M \cup \{c\}$.
    (c) For other cases, set $Q = Q \cup \{add(c, p)\}$ for all $p \in ext(c)$.
7. Goto Step 3.

Observation. Brute-force algorithm generates all common subgraphs.

Proof. Let T be a spanning tree of a common subgraph C between G and G'. T is generated in Brute-force algorithm with *add* function at the nodes of in-degree one, and with *merge* function at nodes of in-degree more than one. □

The number of structures stored in Q can grow factorially. The key to the efficiency is to trim as much redundant subgraphs as possible.

Matching a Graph with a Tree

We shall show the condition for the bounding subgraphs stored in Q.

Theorem 3.6.1 *Let C_i and C_j be two common subgraphs between a graph G and a tree G'. If $R(C_i) \subseteq R(C_j) \wedge L(C_i) \subseteq L(C_j) \wedge ext(C_i) \subseteq ext(C_j)$, then C_i is redundant.*

The theorem is not so trivial as it seems, because it holds regardless of the graph structures inside C_i and C_j.

Lemma 3.6.2 *If $\langle s,t\rangle \in ext(C_i)$, then $s \notin L(C_j) \wedge t \notin R(C_j)$.*

Proof. Directly follows from $ext(C_i) \subseteq ext(C_j)$. □

Proof. (Theorem 3.6.1)
Let C_I and C_J be the MCS containing C_i and C_j, respectively. If $size(C_I) > size(C_J)$, C_I must contain a node-pair $\langle u,v\rangle$ such that $v \in (R(C_j) - R(C_i))$. Otherwise, C_i-component of C_I can be replaced with C_j, and the resulting structure becomes a larger MCS than C_I. Note that this replacement is always possible because $ext(C_i) \subseteq ext(C_j)$. Let $\langle s,t\rangle$ be an arbitrary node-pair in C_i. There exists an undirected path from t to v in C_I, and another undirected path from v to t in C_j. Because G' is a tree, two paths in G' must coincide. However, there must be a node-pair $p \in ext(C_i)$ such that $R(p)$ is on a path from t to v. From the lemma, $R(p) \notin C_j$. Therefore, two paths do not coincide, which is a contradiction (Fig 3.15). □

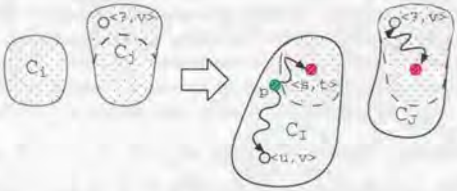


Figure 3.15: There must be a node-pair $p \in ext(C_i)$ (green) such that $R(p)$ is on a path from t to v.

Corollary 3.6.3 *Let C_i and C_j be two common subgraphs between a graph G and a tree G'. If $R(C_i) \subseteq R(C_j) \wedge L(C_i) \subseteq L(C_j) \wedge ext(C_j) \subset ext(C_i)$, and if $size(C_I) > size(C_J)$, then C_I contains a node-pair $p \in (ext(C_i) - ext(C_j))$.*

Proof. If $p \notin C_I$ for all $p \in (ext(C_i) - ext(C_j))$, C_i-component of the C_I can be replaced with C_j. By the same reasoning as in Theorem 3.6.1, C_I must contain a node-pair $\langle u,v\rangle$ such that $v \in (R(C_j) - R(C_i))$, which leads to a contradiction. □

The corollary tells, if $R(C_i) \subseteq R(C_j) \wedge L(C_i) \subseteq L(C_j)$, the C_i-oriented common subgraphs can be discarded if they cannot include any node-pair from $ext(C_i) - ext(C_j)$.

Matching two Graphs

The correctness of the above algorithm does not hold when both graphs contain cycles. In such cases, one approach is to change one graph into a tree by deleting some edges. Another method is to match cycles with priority.

Edge deletion: This strategy resembles the method by Akutsu [6], which uses the dynamic programming in finding MCS for graphs with small number of cycles. Each cycle is opened by deleting one of its edges, and the matching procedure is applied to each resulting tree. (The MCS between two trees can be solved in polynomial time. See [40] for detail.) Although the patterns of the deletion grows as the number and the size of cycles increase, the total computational time becomes polynomial, if the number of cycles is bounded.

Priority for cycles: Aromatic cycles have a different characteristic from allylic cycles. Therefore, it is of little importance in chemistry to match an aromatic cycle with a linear aliphatic chain or with a non-aromatic cycle. Moreover, the generation and the decomposition of cycles do not simultaneously occur in the same reaction. Therefore, the practical way to match chemical structures is to: (1) locate and match cycles with priority; and (2) extend branches using Brute-force algorithm.

The edge-deletion method becomes quite slow for structures with multiple cycles, although it guarantees the correctness of the resulting MCS. Therefore, we adopt the priority for cycles in computing the mapping of metabolic reactions. In order to achieve this priority, we only need to re-label nodes of the input graphs. Let G and G' be two graphs with cycles. Without loss of generality, we assume the number of cycles in G is equal to or less than that of G'. From the property of enzymatic reaction, cycles in G *must* match cycles in G'. The problem of finding cycles of G within G' is the subgraph isomorphism problem, which can be solved by Brute-force algorithm without *merge* function. Note that this simplification reduces the number of common subgraphs considered.[16] Thus, the matching algorithm for two graphs with cycles is summarized as under.

1. Locate cycles in G and G'. If G has more cycles than G', then exchange them.
2. Re-label nodes in cycles of G as $\{1 \ldots n\}$ in a breadth-first way. This labeling prioritizes matching of the cycles in Brute-force algorithm.
3. Apply Brute-force algorithm without *merge* function up to label n. Delete all the common subgraphs which do not contain any of re-labeled n nodes in G.
4. Continue Brute-force algorithm with *merge* function.

Step 3 deletes all subgraphs which do not contain the cyclic component in G. Note that the bounding strategy using Theorem 3.6.1 is applicable in the subgraph isomorphism problem too.

3.6.3 Performance

The algorithm is implemented in C^{++} using LEDA library package [64]. It is difficult to theoretically evaluate the efficiency of Theorem 3.6.1, because the degree of bounding depends on the labeling of nodes. Also, the algorithm cannot cope with large or highly regular graphs, because of the inherent hardness of the problem.

[16] The subgraph isomorphism problem is still NP-hard, as we mentioned earlier.

The question is whether the efficiency of Theorem 3.6.1 is sufficient for chemical compounds appearing in metabolism. The answer is positive. Figure 3.16 shows the maximum number of possible subgraphs in the set Q of Brute-force algorithm for about 70 major compounds in metabolism [17], each of which is matched against itself. (Therefore, the resulting MCS is identical to the input.) Some structures contain cycles, but the matching algorithm is performed without the priority for cycles. Otherwise, the problem becomes much simpler.

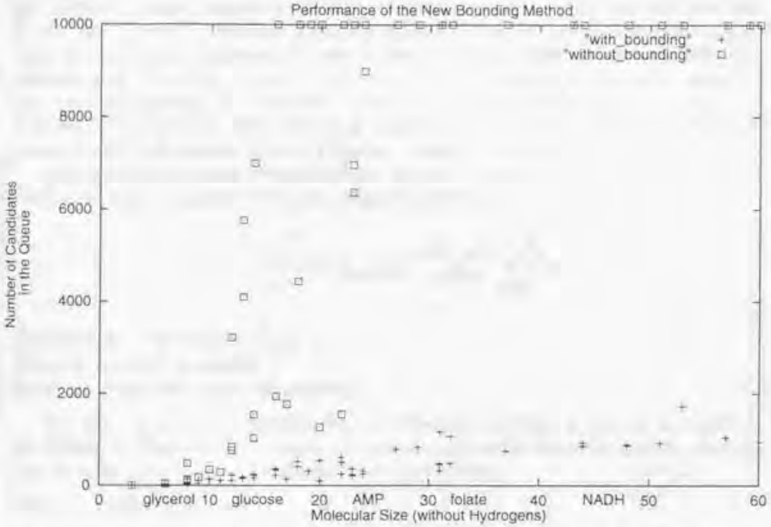


Figure 3.16: Performance of the new branch and bound method. Boxes show the maximum number of candidate subgraphs in the queue without the new bounding method. Crosses show the same parameter with the bounding method. Without the bounding, many structures of size 20 consider more than 10000 common substructures for finding their MCS, while with the bounding, few compounds around size 50 reach this level. Note that most compounds in metabolism have sizes less than 50.

[17] (In size order) ethanol, 3-oxo propanoate, glycerol, lactate, acetyl phosphate, succinate, fumarate, erythrose, oxaloacetate, malate, ribose, ribulose, phenoxy acetate, cis-aconitate, erythrose 4p, glucose, fructose, citrate, isocitrate, glutamate 5p, ribose 1p, ribulose 5p, xylulose 5p, galactose 1p, glucose 1p, fructose 6p, biotin, sedoheptulose 7p, ribulose 15bp, deoxy guanosine, glutathione, fructose 16bp, AIR, PRPP, cyclic AMP, FGAM, GAR, AMP, FGAR, CAIR, orotidine 5p, ADP, dUTP, lanosterol,adenylo succinate, SCAAIR, DHF, ATP, FMN, folate, UDP glucose, heme, NAD+, NADH, NADP+, FAD, CoA, acetyl CoA, citramalyl CoA, ferro cytochrome, citryl CoA. Among them, only lanosterol, heme, and ferro cytochrome reached 10000 when the bounding method is applied.

3.7 Finding Enzymatic Mapping

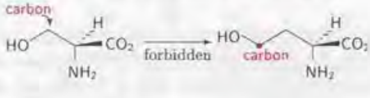
MCS algorithm is used for the analysis of an enzymatic reaction with multiple inputs and outputs. In a chemical database, a reaction is usually described as:

$$s_1,\ s_2 \ldots\ s_m = s'_1,\ s'_2 \ldots\ s'_n.$$

s_i $(1 \leq i \leq m)$: an input molecule, or *substrate*
s'_j $(1 \leq j \leq n)$: an output molecule, *product*

The number of input/outputs is usually very small; m and n are two for most reactions. In our case, s_i and s'_j represent molecular graphs. From the law of conservation, there must be a one-to-one mapping of nodes between both hand-sides. Moreover, nodes from the same graph are mapped always in a cluster. More precisely, the condition is as follows. Let σ be the mappping from the set of graphs $\{G_1 \ldots G_m\}$ to the set $\{G'_1 \ldots G'_n\}$. For each pair of compounds, define the set of nodes $S_{ij} = \{u \mid u \in G_i,\ \sigma u \in G'_j\}$. Then, the subgraph of G'_j induced by the set of nodes S_{ij} must be always connected.

The condition of connectedness forbids the reaction as in the following example, in which a carbon is inserted before the terminal hydroxyl group.



Problem 3. (Connected Mapping)
Instance: Two sets of graphs
Question: Find their connected mapping.

The problem is hard in general, but the following algorithm is used in the actual implementation. This strategy is empirically known to work for almost all existing reactions, and we show a few negative examples in the next section.

Greedy Algorithm

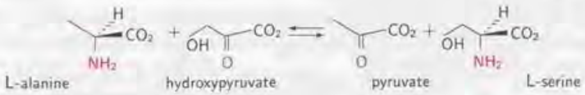
1. Choose two molecular graphs G_i and G'_j $(1 \leq i \leq m, 1 \leq j \leq n)$, which have the largest MCS C_{ij} in the reaction. Replace G_i and G'_j with $(G_i \setminus C_{ij})$ and $(G'_j \setminus C_{ij})$, respectively.

2. Repeat Step 1 until no graph structures are left.

3.7.1 Negative Examples

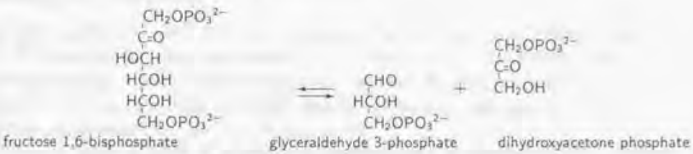
The mapping computed in a greedy way is sometimes not equal to the real mapping in catalysis.

Example. alanine-hydroxypyruvate transaminase



The correct mapping is the transfer of amino group from L-alanine to hydroxypyruvate. However, it can be regarded as the transfer of hydroxyl group too, because both transfers have the MCS of the same size.[18] Without the information that the enzyme is transaminase, it is impossible to choose the correct mapping, because some monooxigenase might convert L-alanine into L-serine, albeit this particular monooxigenase has not been experimentally found.[19] In this case, the program needs a 'teacher' explicitly telling that L-alanine be matched with pyruvate first.

EXAMPLE. fructose 1,6-bisphosphate aldolase



The correct mapping matches the upper part of fructose 1,6-bp with dihydroxyacetone p. Note that the chiral configuration of the central carbon of glyceraldehyde 3-p does not change in the reaction. To cope with this type of reactions, the matching process should consider the change in the attached hydrogens, and the type of bonds. Weighted matching can resolve this reaction.

3.7.2 Discussion

Weighted Matching

The modification for the weighted matching is quite easy, except for the treatment of chiral carbon.

Basically, chiral carbon labeled as either [+] or [−] in a normal form may match the other one. When the configuration of carbons is determined as in Section 3.5, [+] and [−] configurations cannot be used in the weighted matching. The configuration is determined by the entire molecular structure, not by the local arrangement of atoms. For example, suppose a matching process between two graphs G and G'. It is possible that [+] carbon in G matches (including chirality) [−] carbon in G' at one position, while another [+] carbon in G matches [+] carbon in G'. When the structural matching considers chirality, therefore, R,S-system is a more rational definition for the chiral configuration.

In the previous section, we concluded that p-chirality need not be considered in metabolism. That is, [+] or [−] configuration will be assigned to only those carbons which can obtain R,S configuration too. For this reason, R,S configuration is employed in the actual implementation of chirality. ([+] or [−] configuration is used to detect topologically symmetric sites.)

In the implementation, weights for nodes are assigned as in Table 3.1. Also, phosphate is counted as a single node in its entirety, because OPO_3 is always transferred in a cluster.

[18] Note that hydrogens are omitted in a molecular graph.

[19] The function of various enzymes will be discussed later.

matching element	weight
carbon (including chirality and attached hydrogens)	25
carbon (including attached hydrogens)	22
carbon (without attached hydrogens)	10
other elements (including attached hydrogens)	10
other elements (without attached hydrogens)	5

Table 3.1: Weights for Different Atoms

Matching for Cofactors

Enzymes performing oxidation and reduction usually use large cofactors like NAD or ATP.[20] Their pairing products are always NADH and ADP (or AMP), respectively. The knowledge about donors and acceptors in a reaction like this case can be exploited in the generation of the mapping: i.e. these pairing compounds can be removed with priority from the given reaction.

When a large number of reactions is processed, for example, once the program computes the difference between ATP and ADP, it finds that a single phosphate is transferred in the reaction. This relation is preserved for the later matching process. After this computation, whenever the program finds ATP and ADP in another reaction each for either hand-side, it can replace them with a single phosphate on the side of ATP.

Note that such cofactors have their synthesis and degradation pathways too. Therefore, the cofactors must be treated in the same way as other main metabolic compounds, even though they are named as if they are trivial.

[20] The details for these compounds will appear in Section 2.4.

Summary

The main results of this chapter are as under.

- The theoretical definition of aromaticity was shown, and the algorithm for detecting aromatic (under our definition) rings was introduced.
- The classical work by Corneil and Gotlieb for the automorphism partitioning was re-introduced and was applied to the normalization of chemical molecules. We also introduced the method to accelerate their original partitioning process using a distance matrix.
- The strategy to assign the chiral configuration of carbon atoms was explained. The assignment will be time-consuming, when the configuration depends on the configuration of other carbon atoms. We concluded that such carbons are to be ignored in metabolism from a practical viewpoint (in the presence of input error).
- The algorithm for finding a maximal common substructure was introduced. With this algorithm, we showed how to find a mapping of atoms in a reaction.

All results will serve as a foundation for computerizing metabolism.

Chapter 4

Applications

The cookie-cut framework is applicable to biological systems other than metabolism. The essence of the framework is that (1) it provides an environment of pathway search, flexible in refining and modifying results; (2) it is declarative and provides unbiased logical view to users.

4.1 Functional Prediction

As we noted in the first chapter, one application area of metabolic reconstruction is the prediction of functionally unknown genes. In bacteria, many genes show a marginal sequence similarity with several other genes in different species, and the information only from the similarity is insufficient to determine a particular function for those genes.

With AMR system, it is possible to suggest the required function in the target biological system. This enables us the functional prediction in a reverse way; given a function, we pick up one sequence, from among yet unassigned genes which is most likely to code that function.

Note that this reverse assignment is impossible in the 'reconstruction by reference' approach, which considers the consensus pathways only. It cannot provide the set of candidate function which will be required in the modeled metabolism. Moreover, it is difficult to determine the consensus pathway among bacteria, because each may have quite different mechanism, not to say of mammalian cells.

BASIC STRATEGY:

1. Connect fractional pathways to form a larger, meaningful pathway.
   Suppose that already known function of genes comprises a partial pathway, or consecutive reactions. This fractional pathway is conserved, because it is useful in some larger pathway. AMR system can search such larger pathways. The source and the destination of a query (a pattern graph) are determined by the nutritional condition of the growth medium. The search result indicates a set of required gene function, any of which can glue the existing fractional pathways to form a larger one.

2. For each candidate function, find its sequence from among unassigned genes in the operon related to the searched pathway.
   In bacteria, genes used in the same metabolic purpose are likely to be coded in the same operon. It is more likely, therefore, that the candidate function will be found in the operon where other genes from the same pathway exist.

4.1.1 Haemophilus influenzae

We shall explain the strategy more in detail taking *Haemophilus influenzae* (*H.inf*) as an example, the first fully sequenced bacterium whose metabolism is relatively known. It has about 1700 genes in 1.83 Megabase sequence, and a general functional prediction has been done for about 80% of its genes [38, 97].

The validity of the assigned function is verified by knock-outing a particular gene. We shall focus on the pathway from glutamate to proline, and consider a hypothetical case in which the gene converting glutamate to glutamyl phosphate is knocked out. If *H.inf* is still alive after the knockout, there must be a salvage pathway generating proline.

Nutritional consideration

Each bacterium has its own lifestyle. Even if two bacteria belong to the same branch of a phylogenetic tree, they may require different nutritional media of growth. For example, *E.coli* is an organotroph which can grow on

$$\{\ \text{glucose},\ NH_4^+,\ Mn^{2+},\ Mg^{2+},\ Fe^{2+},\ K^+,\ Cl^-,\ SO_4^{2-},\ PO_4^{3-}\ \},$$

while *H.inf* is an obligate parasite, additionally requiring [67]

$$\{\ \text{Arg, Cys, Asp, Glu, NAD, thiamin, pantothenate, uracil, hemin}\ \}.$$

The set of required compounds is called a growth medium. Both bacteria belong to the same branch within the same subdivision of purple bacteria, but their requiring metabolism differ considerably. In *H.inf*, only amino acids other than those listed above need to be generated *in vivo*. That is, its assigned metabolic function constitutes the anabolic (synthetic) pathway of amino acids which are not contained in the growth medium. For example, proline must be synthesized from the above growth medium. Fig 4.1 shows the consensus pathway of the anabolism of twenty amino acids in *E.coli*. From the diagram, it is known that all amino acids are generated from the growth medium in *H.inf*. When glutamate kinase (EC2.7.2.11; glutamate → glutamyl phosphate; Fig 4.1 right below) is knocked out, however, proline must be generated using an alternative pathway.

In *E.coli* and many other bacteria, proline is generated from glutamate. Shortest pathways from proline to glutamate suggests salvation by new functions (See also Fig 4.1.):

1. ornithine → proline + NH_3;

2. glutamate ↔ pyrroline 5-carboxylate;

3. ornithine ↔ glutamate semialdehyde.

Our purpose is to specify a gene of one of the above three types.

Sequence assignment

The problem here is, given a function, how to locate a gene of the specified function. If there is a known sequence of the same function, homology search is one strategy; when the similarity of two amino acid sequences is more than 30%, they are considered to share the same function. When the similarity is around 20% (twilight zone), further consideration is required.

The homology search is powerless if the function under investigation is totally new and has no known instance, or if the similarity is too low to conclude the homology. In such cases, there are several useful aspects for the sequence assignment. Among them, we shall discuss operons and motifs.

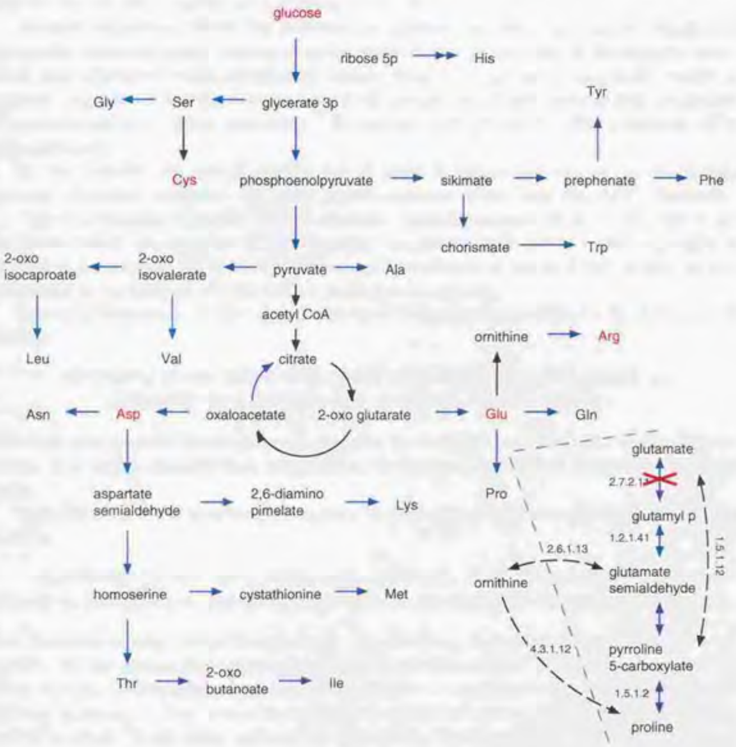


Figure 4.1: Amino acid biosynthesis in *E.coli* and *H.inf.* Amino acids in red are contained in the growth medium. Blue arrows are presumably existing pathways in both bacteria, while black arrows exist only in *E.coli.* Coenzymes are omitted, and arrows are in anabolic direction.

In bacteria, functionally related genes are often coded in the downstream of the same regulatory unit. Such transcription units are called operons. When the sequence we are looking for belongs to a pathway for a characteristic situation, say, starvation, its related function are likely to be coded under the same regulatory switch for starvation. This idea using operons can be relaxed to 'gene cluster', or physical neighborhood. Like in operons, genes in the same cluster tend to belong to the same or related pathway. This observation well fits the physical organization of genes in *E.coli*.

Another important factor for sequence assignment is functional motif. Motif is a commonly conserved short pattern of amino acids shared by proteins of the same function. Motif may represent characteristics of protein structure (e.g. helix-turn-helix motif) or function (e.g. kinase motif). Our interest is the latter one. When motif is well conserved, it is represented as a regular expression.[1] Otherwise, a weight matrix offers a more sensitive representation.

Let us consider the case of finding one of three functions introduced for the salvage pathway of proline synthesis. All motif quoted here are taken from PROSITE database.

Type (1) function is NAD$^+$ linked ornithine cyclodeaminase (EC4.3.1.12), which has no known motif. An enzyme of this function was found in Ti plasmid and is similar to carbamoyl transferase (EC2.1.3.3). Carbamoyl transferase exists in *H.inf* (arcB), so it is interesting to investigate its relation to ornithine deaminase.

Type (2) function is NAD$^+$ linked aldehyde dehydrogenase (EC1.2.1.3), whose motif is either

```
[FYLVA]-x(3)-G-[QE]-x-C-[LIVMGSTANC]-[AGCN]-x-[GSTADNEKR] or
[LIVMFGA]-E-[LIMSTAC]-[GS]-G-[KNLM]-[SADN]-[TAPFV].
```

Although this general function seems essential in metabolism, *H.inf* has none of its instance. It is highly possible that dehydration of aldehyde is achieved using some different route.

Type (3) function is pyridoxal-phosphate linked aminotransferase (EC2.6.1.13), whose motif is

```
[LIVMFYWC](2)-x-D-E-[LIVMA]-x(2)-[GP]-x(0,1)-[LIVMFYWAG]-x(0,1)-
[SACR]-x-[GSAD]-x(12,16)-D-[LIVMFYWC]-x(2,3)-[GSA]-K-x(3)-[GSTADN]-[GSA].
```

This function usually serves for ornithine degradation, though the reaction itself is reversible. If this aminotransferase exists, the overall direction of proline synthesis will be either arginine → ornithine → proline, or glutamate → ornithine → proline. If the former pathway is active, carbon source for ornithine and proline comes only from arginine in the growth medium. If the latter pathway is active, then the degradation pathway of arginine is also likely to exist, in order to remove excess arginine generated through ornithine. In fact, although arginine deaminase activity is reported to exist in *E.coli* [68], its gene (*argA*) is assigned a different function (N-acetylglutamate synthase) in SWISS-PROT. The investigation on the existence of arginine deaminase is important.This analysis also depends on whether arginine is necessary in the growth medium. In other words, the determination of salvage pathway is impossible without considering these aspects.

If none of above motif patterns are detected, motif for substrates may be further taken into consideration. There are motif patterns specific to certain amino acids or coenzymes, and their combination may reveal the function of unassigned genes. As can be seen from the above example, it is difficult to perform the functional assignment automatically, or

[1] The regular pattern is normally described in a PROSITE format. [] denotes a grouping, and x(n) denotes n consecution of arbitrary residues.

in a fully computational fashion, because the lifestyle of bacteria must be taken into consideration. In case of *H.inf.* characteristics to be considered include: (1) its respiration is almost anaerobic; (2) it requires nitrogen and heme rich environment, and so on. All these characteristics should affect the assignment of sequences.

4.2 Signal Transduction

The application area most similar to metabolism is *signal transduction*. It is a cascade of proteins transmitting an extracellular stimulus to its responding mechanism within a cell.

'Signal' is a quite abstract concept. It originates in an external stimulation, which is percepted by a *receptor* protein on the cell surface. This initial signal is sometimes a change in the concentration of metal ions. The receptor usually modifies the subunits of guanine nucleotide binding protein, or *G-protein*, and the signal is transmitted to other proteins by binding with a phosphate in the downstream, until it reaches the target mechanism within a nucleus. Thus, the major component of signal transduction is proteins, not chemical compounds. In addition to the main phosphorylation route, many auxiliary proteins are known to inhibit or forcefully resume the signal of the cascade. Such proteins may perform feedback loops or concerted inhibition or activation. The route is considered a fuzzy network of message passing, which resembles the graph we have discussed in the cookie-cut framework.

We shall briefly describe how to treat signal pathways in the cookie-cut framework. We again stress that the framework does not simulate pathways; it only searches the possible routes of the signal. The signal version of the framework is called signal framework, as we called the metabolic version as metabolic framework.

Molecular Unit

One protein sequence corresponds to one molecular unit in the framework. Although protein does not have a clear structure as a chemical compound does, its amino-acid sequence is regarded as a set of components called *domains* (Fig 4.2).

An amino-acid sequence involved in signal pathways usually contains several domains, characterized by a short consensus sequence having a specific binding target. Typical domains in signaling pathways are:

- SH2 ⋯ Src homolog 2 domain binds to a short ($5 \sim 10$) pattern with a phosphorylated tyrosine;

- SH3 ⋯ Src homolog 3 domain binds to a short (~ 10) proline-rich pattern;

- PI ⋯ Phosphorylated tyrosine interactive domain binds to a short pattern with a phosphorylated tyrosine;

- PH ⋯ Protein kinase C homolog domain is presumed to bind to some short pattern.

Each domain class is further classified into sub-categories, each of which requires a specific short pattern for its binding site. For example, Src family binds to

phosphorylated tyrosine(`pTyr`)-glutamate(`Glu`)-glutamate(`Glu`)- isoleucine(`Ile`),

whereas PLC-γ family binds to

phosphorylated tyrosine(`pTyr`)-hydrophobic residues.

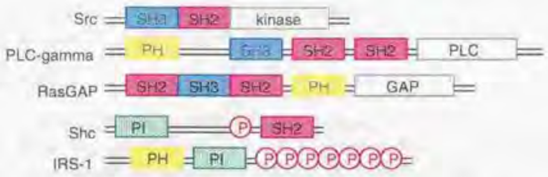


Figure 4.2: Structures of some signaling proteins. P in a circle represents a binding site with phosphorylated tyrosine, to which SH2 domain binds.

In summary, each protein is considered a set of domains, and each domain is characterized by a specific amino-acid pattern.

We introduced protein-protein interaction only, but in general, three types of interaction are identified: (1) binding with a small compound (e.g. metal ions), (2) binding with another protein, (3) binding with DNA. Metal ions and specific DNA sites are therefore treated as molecules too.

Rules and Compartments

Graph edges in the framework consist of specific as well as generic rules. As in the metabolic framework, specific rules correspond to the experimentally confirmed interaction between proteins, while generic rules correspond to the hypothetical interaction between domains.

For example, G-protein consists of three subunits, α, β, and γ. In human, each subunit has many homologs,[2] but if a particular G-protein G_1 always uses only one type of subunits,[3] say, α_1, β_1, and γ_1, this unique specificity is represented by specific rules regardless of their motif. On the other hand, generic rules represent the motif-based relationship. For example, any protein in Src family has a possibility to bind to any `pTyr-Glu-Glu-Ile` domain of another protein.

Note that the function of protein interaction, e.g. activation or inhibition, or the direction of signal transmission is ignored. In the signal framework, all edges are basically bidirectional, and a feedback loop is indistinguishable from an undirected cycle.

In the metabolic framework, we restricted the number of rule inputs to one ($m \to m'_1, \dots m'_l$). This restriction brought us the algorithmic advantage, but in exchange, we lost the potential to describe *AND* condition, such as 'both m_1 *AND* m_2 are required to produce m''. Signal framework introduces the same restriction too, and the interaction requiring several proteins $m_1, \dots m_i \to m'_1, \dots m'_j$ is reduced to i different edges: $m_k \to m'_1, \dots m'_j$ $(1 \leq k \leq i)$. The problem of ignoring *AND* condition will be discussed later.

Cellular compartment is also important in signaling pathways. Since signal pathways are discussed on eukaryotes, many proteins are known to localize at membrane, organelles, and a nucleus. A target pathway is almost always linear, leading from membrane surface to the nucleus. Thus, the overall direction is pre-determined in signal framework in contrast to metabolism, in which compounds can freely migrate from one place to another. The

[2] There are at least 20 α, 5 β, and 10 γ subunits in human [58]. In biology, subunits are refereed only as α, β, and γ. The notation with a subscript number, such as α_1, is our fictional definition.

[3] Experimentally, each G-protein is known to use a unique α subunit. The specificity of β and γ subunits are lower, and the same $\beta\gamma$ complex can bind to different α subunits.

pre-determined signal direction can be imposed as follows: (1) color reactions depending on the loci within a cell; (2) restrict the number of usable reactions for each color.

Pattern and Distance

The assignment of biologically plausible distance is again a difficult problem, because edge distances should not be biased by some expert knowledge. One acceptable assignment of edge distances is to reflect the sequence similarity of domains. Domains with less conservation of the consensus pattern imply less binding specificity, which can be interpreted as 'long' edges. However, the relative length, or 'how long', remains unclear, and this is the reason the same distance is assigned to all edges.

Example: G-protein

Different kinds of G-protein participate in the signal transduction, and Fig 4.3 is one such example. The signaling process proceeds as follows. A hormone called epinephrine binds to a receptor on a liver cell (Step 1). The receptor aggravates dormant G-protein, and it exchanges its bound GDP with GTP (Step 2). The GTP-bound, active G-protein releases $\beta\gamma$ subunit, and moves along the cell surface to an effector protein (Step 3). The G-protein activates the effector (Step 4), and the effector cyclizes ATP into cAMP (Step 5). The right cartoon of Fig 4.3 shows the model of the signal mechanism. We assume that there are several homologs for each G-protein subunit, and they all have a potential to bind to each other. Note that the query `?( epinephrine :> cAMP )` returns the path shown by red arrows, `( epinephrine > receptor >` $\alpha 1$ `> effector > cAMP )`, which corresponds to the physical binding process. What is missing in this shortest pathway is the information that GTP and β, γ subunits are 'required' for the normal signal transduction. The fact

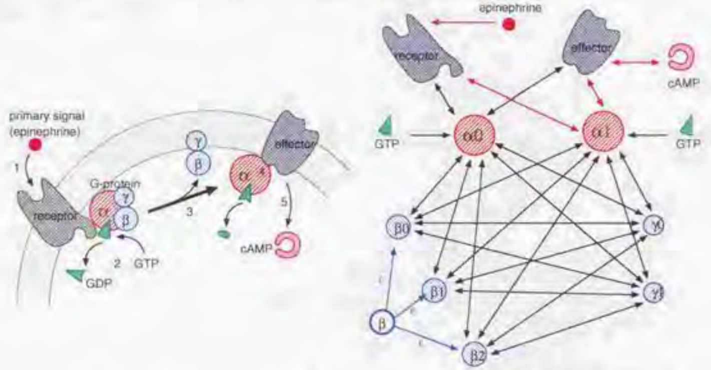


Figure 4.3: Signal pathway of G-protein in human. The widely accepted signaling mechanism (left cartoon) is interpreted as a graph in the signal framework (right cartoon). In the graph, several homologs are shown for each of the three subunits. Red arrows show a shortest pathway from epinephrine to cAMP. The unshaded β subunit is a conceptual β, representing all β subunits. Blue arrows from the conceptual node have no edge length, denoted by ϵ.

that signal framework cannot describe *AND* condition sounds fatal at this point, but in reality, it is not.

When either one of three β subunits in the cartoon is experimentally shown to be necessary, for example, there must be a pathway from one of β_0, β_1, and β_2 to the shortest pathway shown by red arrows. Considering this, we can supplement a conceptual node β (white β) to the graph, and issue the query to `?( epinephrine :> cAMP <:` β `)`. All edges from the conceptual node (blue arrows) have length zero, and any of three actual β subunits may be equally reached. Note that the information that 'β is necessary' says nothing about the meeting point of the pathway from β with the main red pathway; it is unknown whether β_i $(i = 0, 1, 2)$ independently binds to α_1, or affects one of γs before the binding, or it may even interacts with α_0, which binds to the effector.

On the other hand, if we know that β_1 and γ_1 together binds to α_1 from laboratory results, then we can supplement a node representing the complex of α_1, β_1, and γ_1.

Thus, signal framework can generate such logically possible routes of required components, as long as laboratory results do not include many *AND* functions.

One possible problem is that shortest paths may be too short to be realistic, because molecular units are linked sorely on protein motif. Many proteins contain phosphorylated tyrosine sites or proline-rich sites, but not all of them would serve as SH2 or SH3 binding domains. In order to reduce the number of edges, results from the yeast two-hybrid experiment is of great importance. However, the introduction of two-hybrid system is beyond our scope here.

Chapter 5

DNA computing

Man is still the fastest and most efficient computer that can be mass-produced with unskilled labor.
— Wernher von Braun (1912-1977) Father of Rocket

5.1 Overview

Even when the computational analysis of a biological mechanism seems reliable, the predicted result should finally be confirmed in lab experiments. This final step of verification should be systematic too, when the systematic reconstruction of the mechanism is feasible. Since the main thesis is the computational reconstruction of metabolic pathways, the corresponding laboratory work will be the gene knockout and the expression analysis, which are the necessary step for verifying pathways in metabolism.

Knockouting a particular gene has become a routine process in molecular biology, but its procedure is far from being systematic. After choosing a gene to be disrupted, biologists normally design a DNA sequence to replace the gene, synthesize the sequence, and ligate it into a ready-to-use DNA fragment for the gene knockout. The expression analysis also follows the similar procedure.

In short, experimentalists are allowed to modify the gene only in a top-down manner; they first expect a certain result, design a DNA sequence to achieve the result, and apply the sequence to confirm the expectation. What is necessary for a systematic reconstruction is, however, not the confirmation of an expected result. There should be a bottom-up method which can provide possible alternatives. For example, experimentalists are encouraged to generate all possible vectors, and to check which vectors can produce the same results. For this purpose, a free design of various DNA sequences is important.

In this chapter, we introduce basic laboratory techniques for the systematic generation of DNA sequences in laboratory works. Specifically, two types of experiments and one new model of computation are introduced: (1) the relational operations on DNA vectors, (2) the parallel generation of DNA vectors on a solid-phase, and (3) the parallel and programmed generation of DNA vectors as a new model of computation.

The later sections are organized as follows. In Section 5.2, the necessity of a new strategy for knockouting genes is explained in contrast with the traditional knockout method.

In Section 5.3, DNA computing is introduced as the new paradigm for the systematic understanding of life. Although its original aim is to perform logical computation using real DNAs, its ideas and techniques will be shown to fit for the free design and synthesis of DNA sequences.

Remaining sections are the laboratory result of DNA computing with which the author has been involved.

In Section 5.4, laboratory results indicating the applicability of relational operations on vectors is shown [10]. In Section 5.5, the feasibility of efficient synthesis of the vectors is demonstrated on a solid-phase [71]. In Section 5.6, a new model of computation is introduced. In this model, the parallel generation of vectors can be *programmed*. Each DNA fragment represents one program and its input, and multiple programs with different inputs can be executed in a single tube within a short time.

In Section 5.7, the result of the above experiments are summarized and the future direction is discussed. The detail of the experiments are attached in the last section.

5.2 Gene Knockout

The rapid accumulation of knowledge in molecular biology owes much to an laboratory technique called homologous recombination, a widely accepted procedure for knockouting genes. In homologous recombination, an extrinsic DNA fragment is introduced into a cell using a circular DNA, which replaces the cellular DNA fragment by recognizing its common subsequences (Fig 5.1).

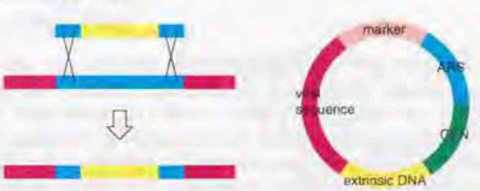


Figure 5.1: Left: homologous recombination. The artificially modified part (yellow) will replace the original gene at a certain probability. Flanking sequences (blue) must be identical. Right: the structure of an example vector. For detail, see the main text.

The tiny circular DNA used to introduce the DNA fragment is called a *vector*. A vector is an artificially designed virus, containing at least two important units: an extrinsic gene to be exchanged with the original gene, and a marker sequence against which recombinant cells can be selected with a chemical. In addition, a vector may contain various additional units. Most common are the restriction sites to easily ligate the extrinsic gene into the vector, an autonomously replicating sequence (ARS) to freely replicate vectors within an introduced cell, and a centromere (CEN) sequence to stabilize the copy number of the vector within a cell.

Currently, the arrangement of these units are custom-made; users can exchange the extrinsic gene-part only, and the rest remains untouched. For the systematic knockout experiments, however, it is better for these units to be flexibly assorted and rearranged to suit the different purpose of the different experiments. For example, a vector is introduced to a cell in order to knockout a particular gene. After knockouting the gene, it is likely to introduce several different vectors to the cell. For the expression analysis, their difference may be the copy number of an extrinsic gene (dosage), the arranged order of extrinsic genes, or the existence of ARS (CEN) unit.

Traditionally, a vector has been generated by a careful recombination, restriction, and ligation of units by enzymes. This process is essentially a top-down analysis; the sequence is designed only to confirm an expected result, or to verify an already hypothesized model. However, what is truly systematic is a bottom-up, constructive approach, in which different vector sequences are tested in a cell, in order to search or elucidate a model. For this purpose, we need a method to systematically generate and rearrange different vectors, by an efficient assortment or a randomization of functional units.

To deal with this topic in a formal way, the structure of a vector is regarded as a list of functional units such as ARS and CEN. The function and the length of each unit are abstracted, and the discussion concentrates on how to assort or rearrange the short units to generate longer sequences. In reality, each unit may be as long as hundreds of base pairs, but each may consist of smaller subunits [83].

The crucial point for the systematic design of vectors is the feasibility of relational algebra on DNA fragments, when DNA fragments are regarded as tuples in relational database. Realization of the relational operation on DNA molecules indicates the possibility of parallel assortment and rearrangement of various vectors for experiments. Moreover, the randomization of functional units may be a novel method for understanding a biological mechanism. The protocols which the author has been investigated in DNA computing are ideal for this purpose.

5.3 DNA computing

DNA computing, first demonstrated by Adleman in 1994, is an attempt to perform logical computation using DNA molecules in laboratory experiments. After Adleman showed that a tiny instance of Hamiltonian path problem (HPP) is solvable in a week of experiments, many researchers started to investigate on the computational power of DNA computing. Its theoretical advantage is the inherent massive parallelism of molecules: more than 10^{20} molecules can be easily treated in a tiny plastic tube for little energy cost [1]. Using this advantage, Lipton proposed a model of molecular computation, which consisted of *Extract*, *Merge*, and *Detect* operations [59]. Assuming these primitives to be perfect, he showed that the satisfiability of any boolean formula (3-SAT) can be tested using the number of operations proportional to the size of the formula.

There has been many following papers on molecular computation, showing how various classes of decision and optimization problems can be solved in this new paradigm [14, 16, 7]. Despite the initial optimism, however, researchers came to know pros and cons of molecular operations [89]. Many theoretical works are premised on unrealistic operations, and ignore the most unfavorable factors in experiments, i.e. the instability of lab operations susceptible to temperature, substrate concentration, and enzyme activity. Another hurdle is the difficulty in realizing complex procedures and data structures in terms of primitive operations on molecules.

Consequently, in contrast to the fanatic enthusiasm in its theoretical analysis, laboratory achievements in DNA computing have been rarely reported. From the experimentalists' viewpoint, DNA computing faces serious difficulties.

- **Reliability**. Laboratory operations are unreliable. They sometimes rely on uncertain biological reactions susceptible to factors like temperature, concentration, and sequence length. To make DNA computing successful, a reliable data encoding method and the careful adjustment of above factors are necessary. Also, the use of enzymes should be avoided as much as possible, because they deteriorate with time, and because their efficiency heavily relies on the above factors in experiments.

- **Cost**. Molecular data are expensive to synthesize both in time and money, because a number of data used in DNA-based computation must be physically synthesized one by one. About 100 bp or shorter DNA can be obtained by mail order, but its purity exponentially deteriorates as its length gets longer.[1]

Thus, generating a very long DNA sequence by a synthesizer is currently impossible. It is important to realize operations such as Cartesian product or Join in relational database theory, for the systematic generation of DNA data.

Many theoretical works have not considered the feasibility of their computation. To clearly know what can be and cannot be done with DNA molecules, more basic experiments are necessary. From this standpoint, the author has been working on DNA computing to clarify what is really possible in laboratory works [10, 71, 70, 11, 42].

A vector is called a *DNA tuple* hereafter, because it corresponds to a tuple in database theory. A DNA tuple is in the form of `(tag data tag)+` , in which + means the repetition of one or more. Tags represent the kind of data (attribute name), and can be used as target sites of PCR primers.

5.4 Designing Tuples: Relational Operations

This section shows the laboratory results of efficient and reliable concatenation and rotation of DNA tuples. The key idea is to elongate or shorten DNA sequences using PCR. This method safely treats various kinds of data in parallel, and suggests the possibility of achieving relational operations with DNA molecules.

5.4.1 Results

Concatenation and rotation of DNA tuples were performed using PCR.

PCR-CONCATENATION: Different DNA strands were concatenated by PCR when their termini overlap at least for 15 bp. Sequences overlapping for 30 bp produced more reliable results.

PCR-ROTATION: DNA tuples were rotated by **1.** integrating biotinylated uracil to the tuples; **2.** collecting tuples by streptavidin; **3.** circularizing tuples by ligase; **4.** applying PCR to obtain rotated tuples (Fig 5.8).

The successful results in PCR-concatenation and PCR-rotation suggest the realization of relationally complete operators: *selection, projection, union, set difference,* and *Cartesian product.* **1.** For *selection*, primers with biotin are annealed to target attributes and are collected altogether with streptavidin. **2.** For *projection*, tuples are rotated and amplified by inverse PCR so as to delete unnecessary attributes. **3.** For *union*, solutions are simply mixed. **4.** For *difference*, an excess amount of subtracting tuples with biotin are annealed and collected with streptavidin. **5.** *Cartesian product* is achieved by PCR-concatenation. Note that there is no procedural difference between Cartesian product and Join operation. We tested PCR-concatenation with overlaps only of tags. This concatenation corresponds to Cartesian product. In the same way, PCR-concatenation with overlaps of data and tags altogether corresponds to Join operation.

5.4.2 Experiments

Data representation

Five tuples designed for the experiments are in Fig 5.2. Each numbered unit indicates a corresponding 15 bp sequence. A striped block represents data attribute, and has tags

[1] The fidelity of DNA synthesizer is said to be 99% per base. When the synthesized sequence is 100 bp long, the ratio of the correct sequence is $(0.99)^{100} \times 100 = 36.6\%$.

(shaded blocks) on both sides. Each tag shows what kind of data is stored after or before it, and each corresponds to a unit of elongation in concatenation using PCR. In Fig 5.2, tuple A and B (or C and D) are the same type of tuples, in which only data attributes are different. Therefore there are three types of tuples.

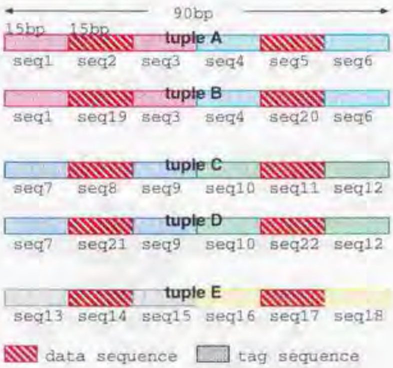


Figure 5.2: five tuples

Let 1-2-3 be a concatenated sequence of seq1, seq2, and seq3. Tuples from A to E in Fig 5.2 are written A, B, C, D, and E, respectively. Single-stranded DNA has a header fwd or rev before a sequence of numbers and alphabets; fwd indicates the 5'→3' direction, and rev indicates the 3'←5' direction. A sequence of numbers without any header indicates double-stranded DNA.

Simple polymerization

Two single-stranded DNAs with an overlap anneal at the overlapping region at low temperature. Theoretically, if a polymerase exists in the same buffer, the enzyme extends annealed DNA terminals to form double-stranded DNA. However, this polymerization was shown inefficient although a large amount of *Taq* polymerase existed. With *Taq* polymerase, each tuple (double-stranded DNA of 90 bp) was made from two overlapping single-stranded DNAs of 60 bp and 45 bp each, and the concentration of the product was less than 20 RFU (Fig 5.3). The amplification by PCR was necessary to obtain the sufficient amount of tuples after the simple polymerization.

PCR from single-stranded templates

Amplification by PCR after the simple polymerization is a time-consuming process, and it is preferred to directly amplify double-stranded DNA from the single-stranded templates. By adjusting the concentration of templates, amplification by PCR from single-stranded templates was shown possible (Fig 5.4). Table 5.1 shows the amount of templates which was required for the amplification.

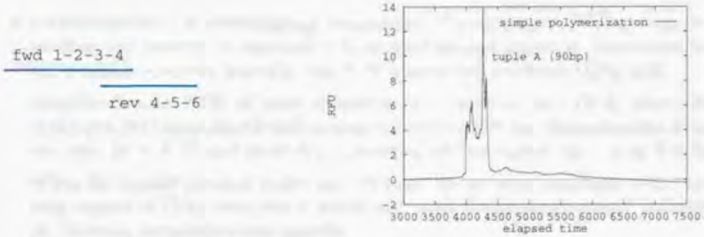


Figure 5.3: simple polymerization

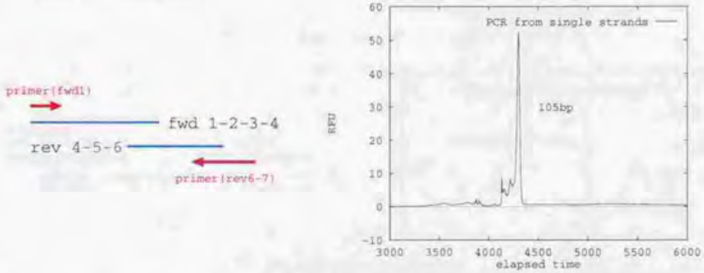


Figure 5.4: PCR amplification from single strands

concentration (*fmol* each)	0.1	1	10	100
amplification	×	×	×	○

Table 5.1: amount of template (per 100μl)

Concatenation

The conventional way to concatenate DNA is ligation of blunt or sticky ends. We propose more efficient procedure called 'PCR-concatenation'.

- **Concatenation of overlapping sequences**: If templates in PCR overlap to one another, the product is expected to be a concatenated sequence. Concatenation of three double-stranded tuples in one PCR process was performed (Fig 5.5).

  Amplification by PCR of three templates (`A-7`, `6-C-13`, and `12-E`) with primers (`fwd1` and `rev18`) produced double-stranded DNA of 270 bp. Concatenation between two sets, {`A-7`, `B-7`} and {`6-C`, `6-D`}, produced all four tuples: {`A-C`, `A-D`, `B-C`, `B-D`}.

  When the concentration of tuples was 100 fmol/100 μl, three templates with overlapping regions of 15 bp were concatenated too. With the concentration of 10 fmol/100 μl, however, no product was amplified.

- **Concatenation with bridges**: In the previous method, each tuple should be elongated for 15 bp before the PCR-concatenation. This process can be omitted if the concatenation without overlap is possible. For this purpose, disjoint tuples were amplified by PCR in the presence of bridge primers which overlap both DNAs to be concatenated (Fig 5.6).

  A double-stranded tuple was amplified when the concentration of both templates and bridges was 50 fmol/100 μl (Table 5.2). The concentration of templates was always 50 fmol/100 μl.

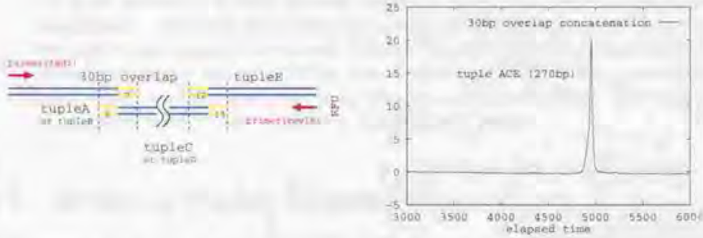


Figure 5.5: concatenation with 30bp overlaps

concentration ($fmol$ each)	50	500	5,000
concatenation	○	×	×

Table 5.2: amount of bridge (per 100μl)

Rotation

Linear DNA data can be rotated by cutting self-circularized data at a new position.

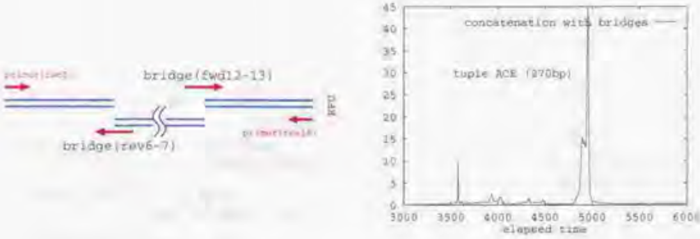


Figure 5.6: concatenation with bridges

- **Rotation in diluted solution**: The conventional way to circularize DNA is to dilute solution until the probabilistic condition favors the circularization rather than polymerization [86]. Then, a subsequent inverse PCR on circular DNA can produce rotated data. In our experiments, however, the application of ligase (more than 2 h) to the diluted mixture of two tuples `A-C-E` and `A-D-E` produced 540 bp dimers, even though the concentration of two tuples is sufficiently low. This result shows diluting solution cannot prevent ligation between two tuples (Fig 5.7).

- **Rotation with biotin**: A procedure called 'PCR-rotation' with biotin was performed to prevent each DNA fragment from touching one another (Fig 5.8).

  Using an inverse PCR with primers `fwd10-11` and `rev8-9`, we confirmed correct amplification of rotated data. Two shorter extra bands, however, were observed too, and they always appeared even with different ligation times (Fig 5.8). With primers `fwd1-2` and `rev17-18`, only 270 bp sequence was amplified. There were no 540 bp band. With a primer `fwd10-11` only, no sequence was amplified. This fact shows there was no head-to-head dimers on the surface of beads.

5.5 Designing Tuples: Stepwise Generation

This section shows the laboratory results of stepwise generation of DNA tuples with a solid-phase chemistry. The key idea is to fix one termini of DNA on the surface, and to stepwise extend the other termini by selectively hybridizing and ligating additional units. This method treats data in parallel, and enables the efficient generation of different arrangements of tuples within a short time.

Solid phase DNA synthesis is an automated technique for synthesis of polynucleotides, in which DNA chains are extended while they are attached to the solid support. In the standard synthesis, all nucleotides in a reaction mixture are coupled to the end of growing chains. Our approach is different from the standard method in that only the selected units, not all of them, are coupled to the termini of tuples. For this purpose, several basic operations were established, including the technique to avoid duplication of the same units in tuples. With these operations, it is possible to efficiently solve HPP, the same instance by which Adleman showed the effectiveness of DNA computing in his seminal paper. For the solution of HPP, readers are referred to our laboratory works [71, 70, 11].

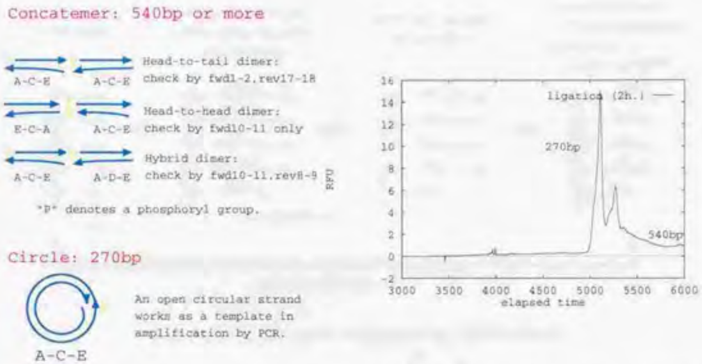


Figure 5.7: inverse PCR of ligated sequences (left); ligation of tuple ACE and ADE (2 h)

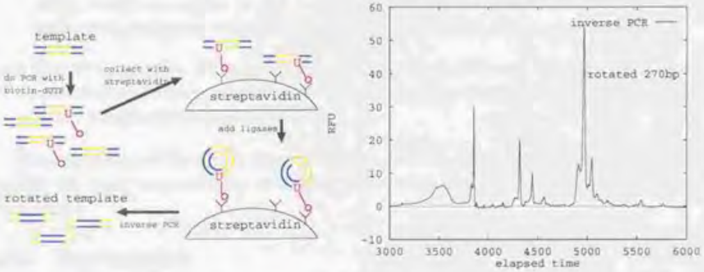


Figure 5.8: procedure of PCR-rotation

5.5.1 Results

Four basic operations were performed. These operations form a cycle to extend DNA tuples on a solid phase (Fig 5.9).

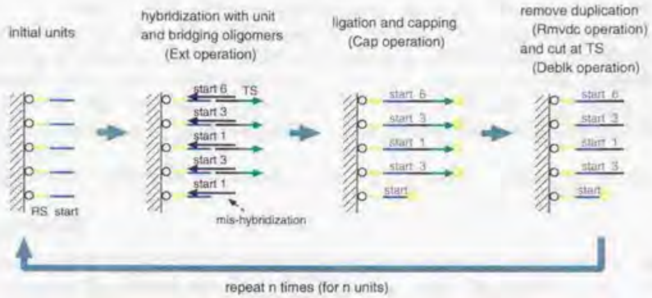


Figure 5.9: The cycle of generating DNA tuples.

- EXT operation: This operation extends DNA tuples by one unit in parallel. Single-stranded DNA fragments were hybridized to the termini of DNA tuples on the solid support. The following ligation process by *Taq* DNA ligase secured the hybridized fragments.

- CAP operation: This operation prohibits the extension of tuples. Termini of the DNA tuples were capped with an unreactive chemical (dideoxyribosyl thymidine triphosphate, or ddTTP).

- RMVDC operation: This operation removes only those tuples that contain duplicates of the same unit. The tuples with more than one copy of a certain unit were detected using a tandem repeat of the complementary sequence of the unit. The detected tuples were removed from the solid support by a restriction enzyme (*Eco*RI).

- DEBLK operation: This operation restores the inactivated tuples. The capped termini of the DNA tuples were trimmed by a restriction enzyme (*Bam*HI) to advance to the next extension cycle.

The repetition of the above four operations, with an occasional application of PCR to amplify the total copy number of tuples, will realize the parallel generation of different DNA tuples.

5.5.2 Experiments

Generation of DNA tuples

Each unit is represented by a single-stranded unique oligomer (15 bp). Each oligomer is combined with the common terminal sequence (`TS`) at its 3' end. `TS` begins with a restriction site for *Bam*HI, so that only the terminal part can be cut and removed. In order to connect multiple units, bridging oligomers complementary to the two adjacent units (30 bp) are also prepared. Starting from an initial unit on a solid support, additional units are stepwise hybridized with bridging sequences. An initial unit is a combination of the root sequence (`RS`) with a restriction site and the starting unit (`start`).

First, initial oligomers representing `RS` and `start` are secured on the solid phase. To extend the tuples on the surface unit by unit, the following process is repeated. **1.** Expose the surface to unit oligomers and bridging oligomers. Oligomers of an adjacent unit hybridize to the free tail of each tuple with the help of bridging oligomers. **2.** Ligate the hybridized units. **3.** Cap the tail of each tuple with ddTTP, which is chemically inert. **4.** Remove tuples which contain the same unit twice. This process will be explained in detail later. **5.** Cut the terminal with a restriction enzyme, *Bam*HI. Only tuples with `TS` are cut, and their caps are removed. Tuples which failed in the last ligation step (Step 2) remain capped.

This process is repeated for n times, where n is the number of units in the tuple.

Ext operation

The accuracy of EXT operation was examined by the application of PCR right after the first EXT operation. Specifically, three bridging oligomers, `revU0-U1`, `revU0-U3`, and `revU0-U6` were added in the first EXT operation, and the correct tuples only were generated. PCR between `fwdRS-U0` and `revU`i`-TS` ($i = 1, 3, 6$) gave the product of 80 base pairs eluted around an elution time of 14 min. In contrast, PCR with other primers did not produce any significantly amplified DNA fragments. The correctness was confirmed at two ligation temperatures 60 ℃ and 65 ℃.

In addition to the accuracy, the efficiency of EXT operation is important. The efficiency was measured by the peak area of a product obtained by the PCR, because the peak area reflects the amount of tuples generated by EXT operation. The PCR product of `RS-U0-U1-TS` was observed to be more than that of `RS-U0-U3-TS` or `RS-U0-U6-TS`, which means that `revU0-U1` could make a more stable hybrid duplex with `fwdU1-TS` than the other two oligomers. This was probably the effect of GC contents of the units (GC content of `fwdU1`, `fwdU3`, and `fwdU6` are 0.6, 0.4, and 0.4 respectively.)

Cap operation

The CAP operation covers the 3' end of tuples on a solid support with an unreactive group, which will be later removed by DEBLK operation. The efficiency of CAP was examined by comparing the amount of PCR-amplified `RS-U0-U1-TS`. In one PCR, CAP is applied to `fwdRS-U0` before EXT operation, and in the other, `fwdRS-U0` is extended without capping. The cycle in the PCR was as small as 15 so that the quantitative balance was preserved. The 80 bp PCR product was greatly reduced when the beads were treated with CAP operation, although 100% suppression was not achieved.

Deblk operation

The DEBLK operation trims the 3' end of tuples by a restriction enzyme. Its efficiency was examined by repeating the synthetic cycle on a solid phase. After the first EXT operation, one-fiftieth of beads were analyzed by PCR to display the sequences of tuples on the beads. PCR between `fwdRS-U0` and `revU1-TS` gave a product of 80 bp eluted at 14.0 min. The rest of the beads were then subjected to CAP, DEBLK and EXT operation. After the second EXT operation, PCR between `fwdRS-U0` and `revU2-TS` gave a product of 95 bp eluted at 14.3 min. The existence of the 95 bp product proved the execution of DEBLK operation. However, PCR between `fwdRS-U0` and `revU1-TS` also gave a product of 80 bp, which means some tuples remained inactivated after DEBLK operation. From the peak size of PCR results, the efficiency of DEBLK operation was extrapolated 80%.

Rmvdc operation

Restriction enzymes cut double-stranded DNA only, and not single-stranded DNA. Each restriction enzyme has its own specific recognition site to cut. Using this enzyme, DNA tuples containing the same unit twice can be removed from the surface. Assume that a tuple has a duplicate unit `U`i. When the tuple is mixed with a tandem repeat of the complementary sequence of the unit `revU`i`-U`i`-TS`, it will form either a partial hybridization at each of the duplicate locus, or a full-length hybridization with a bulge loop (Fig 5.10). Note that the existence of `revTS` facilitates the binding specificity to the free termini of tuples. The stability of a hybrid depends on the free energy of the hybrid. The lower the free energy is, the higher is the stability. The free energy of the hybrid is the sum of the free energy of each part composing the hybrid. In general, single-stranded regions are regarded as the standard state of the free energy calculation; thus a zero free energy is assigned to them. double-helical regions have a negative free energy, while loops such as bulge, internal, or bifurcation have a positive free energy. The free energy of a double-helical region is approximately proportional to the length of the region, and the free energy of a loop increases monotonously with its size. From this rough consideration, Case 3 in Fig 5.10 is considered most stable.

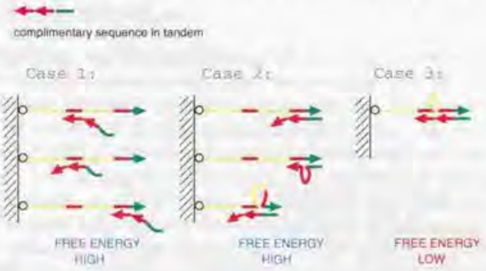


Figure 5.10: The structure of possible hybrids formed between a tuple and a tandem repeat oligomer.

To be more precise, Case 3 is most stable only when the length of units is large enough to compensate the increase in free energy caused by the bulge loop formation. Let the length of units be n bases. The free energy of the double-helical region of a unit is given by $-2\times(n-1)$ kcal/mol, assuming that the sequence of units has the uniform and average stability of base-stacking [20]. On the other hand, the free energy of a bulge loop of m bases long is given by $+1.5\log m$ for large m [30]. By this calculation, it is shown that the formation of full-length hybridization of 15 bp units is preferred even with a bulge loop of 1000 bp in between. (Careful adjustment in the temperature is necessary.) To remove only those strands that the tandem oligomers have fully hybridized, the tuples to be removed are made double-stranded by polymerase. Then, a restriction enzyme can cut those tuples at their root sequence (Fig 5.11).

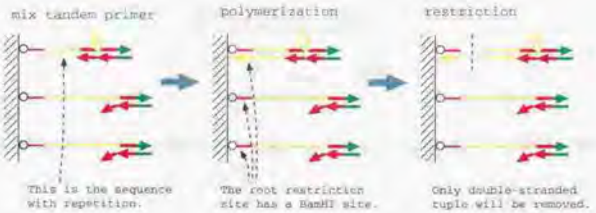

mix tandem primer
polymerization
restriction
This is the sequence with repetition.
The root restriction site has a BamHI site.
Only double-stranded tuple will be removed.

Figure 5.11: Removal of sequences with repetition

5.6 DNA State Machine

With standard molecular biological techniques, the best way to utilize the molecular parallelism has been to execute what is called *generate and search*, as was initially performed by Adleman. He first generated all the candidate solutions for an instance of HPP at random, then chose the correct solution among them in standard lab operations. Lipton's method for solving 3-SAT is also along the line of generate and search paradigm.

However, the power of generate and search with molecules is currently insufficient for solving actual problems, mainly because the selection of a particular DNA molecule (search process) is much more difficult than generation; it often happens in standard lab procedures that different molecules contaminate, or that the target molecule is lost. In order to make DNA computing practically reliable, it is desirable that molecular operations generate only required DNA sequences, without selection procedures. As a step toward this goal, this section introduces a new model of DNA computing, in which the generation of DNA molecules is constrained and programmable.

In previous approaches, programs are implemented as a sequence of operations that are uniformly applied to all molecules in a tube. It is a SIMD model of DNA computing. The new model is a significant progress from this existing paradigm, because the molecules represent both programs and inputs, and multiple programs with different inputs can run in parallel. It is a MPMD or MIMD model of DNA computing.

It is a method for simulating deterministic or non-deterministic state transitions. Both a finite automaton and the current state are represented in a DNA sequence, and different automata with different current states can run in parallel in a single tube within a short time. Therefore, this method can be applied to implement a general state machine with DNA molecules.

5.6.1 Whiplash Model

A finite automaton (or a program) and the current state of evaluation (or a head in the sense of Turing machine) are encoded in one single-stranded DNA with a spacer in between (Fig 5.12). State transition from the initial state is performed by molecular operations. In each transition, the head reads information from the automaton to change the state: single-stranded DNA forms a hairpin structure where the current state hybridizes to an appropriate position in the automaton, then the 3' end of the state is extended by one symbol to obtain the next state using polymerization. This is called a whiplash model.[2] Its contribution is twofold: theoretical as well as realistic.

[2] The name 'whiplash' was coined by Erik Winfree at Caltech.

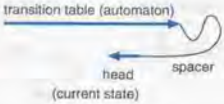


Figure 5.12: Current state of an automaton hybridizes to the automaton.

In theory, this method implements successive and parallel state transitions as a single-tube reaction. This is, to the author's knowledge, the first molecular computation in which programs are implemented in molecules and evaluated in parallel. The transitions are realized as a succession of thermal cycles, realizing the parallel evaluation of multiple automata in a single tube.

For a new laboratory technique, a method to stop polymerization exactly after the head copies one symbol is introduced. For this purpose, we use a stopper sequence consisting of a deoxyribonucleotide whose complement is missing in the polymerization buffer. This technique is called *polymerization stop*.

5.6.2 Molecular representation

The transition table of a finite (deterministic or non-deterministic) automaton is a sequence of triples of the form s'-c-s, where s denotes the current state and s' the next state corresponds to the input symbol c. Lower-case letters are used to denote DNA sequences corresponding to automaton states, which are denoted with upper-case letters. For example, the table of the automaton M in Fig.5.13 is represented by the sequence `p-0-s-q-1-s-q-1-p-r-0-p`.

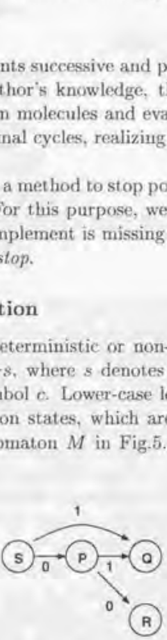


Figure 5.13: Automaton M

To realize state transitions, we concatenate the complement $\overline{s}$ (The bar denotes the complementary sequence.) of the current state s of the automaton as in Fig 5.12, where a spacer sequence with an appropriate length is inserted in between the table and the state.

The state transition corresponding to the input symbol c is performed as follows. First, $\overline{c}$ is attached to the end of the sequence using a novel method for concatenating DNA molecules on a solid-phase [71]. Then, the terminal part becomes the subsequence $\overline{s}$-$\overline{c}$, which can hybridize to the triple s'-c-s in the transition table by forming a hairpin structure. By polymerization, the next state $\overline{s}'$ is copied to the terminal. This cycle is repeated until sufficiently many polymerization proceeds for all the state transitions. Note that even if the head hybridizes to the previously visited states, no harmful effect will be induced.

In order to stop the extension exactly after one symbol is added to the 3' end, both input symbols and automata contain a stopper sequence (denoted by `stop`) in between

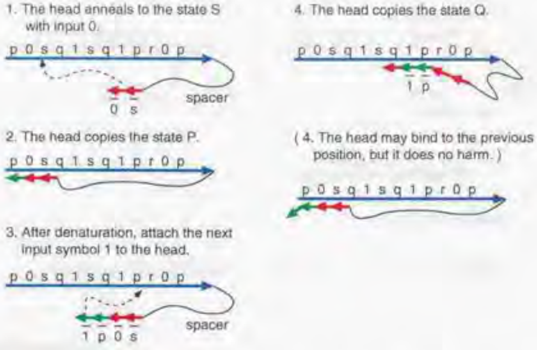

1. The head anneals to the state S with input 0.
p 0 s q 1 s q 1 p r 0 p
$\overline{0}$ $\overline{s}$
spacer
2. The head copies the state P.
p 0 s q 1 s q 1 p r 0 p
3. After denaturation, attach the next input symbol 1 to the head.
p 0 s q 1 s q 1 p r 0 p
$\overline{1}$ $\overline{p}$ $\overline{0}$ $\overline{s}$
spacer
4. The head copies the state Q.
p 0 s q 1 s q 1 p r 0 p
$\overline{1}$ $\overline{p}$
(4. The head may bind to the previous position, but it does no harm.)
p 0 s q 1 s q 1 p r 0 p

Figure 5.14: Evaluation steps for automaton M

symbols. The actual implementation of the stopper sequence is described later.

Non-deterministic automata do not contain the stopper sequence inside the triple s'–c–s. When the head $\overline{s}$ hybridizes to s'–c–s, the fragment $\overline{s}'$–$\overline{c}$ is copied by polymerization, regardless of the input symbol c.

In the above evaluation step, all automata in a tube are attached the same input symbol stepwise by an external procedure. This method is originally used for the selection, or the learning, of automata which accept the given sequence of input symbols. For the flexible design of DNA tuples, however, the model is used in a different way.

Suppose we have a DNA strand which contains all functional components used in DNA tuples. From among this collection, we want to select some components in a certain order, for generating each DNA vector. This process can be realized by attaching an input vector to the collection of components as in Fig 5.15. The input vector is represented as the list of fragment pairs. To the 3' downstream from the 5' terminal, it contains the list of the ending and the beginning part of every adjacent components in the DNA tuple to be generated. For example, when the DNA tuple to be generated contains n functional components $i_1, i_2, \cdots, i_n$, the input vector is the concatenation of $n-1$ pairs $\{$ $\texttt{begin}i_{k+1}$-$\texttt{end}i_k$ $\}$ $(k = 1 \cdots n-1)$, where $\texttt{begin}i_k$ denotes the beginning 20 bp of the component i_k, and $\texttt{end}i_k$ denotes its ending 20 bp. In the same way as the DNA automata, the head alternately anneals to the input vector and the functional components, forming the DNA tuple in the defined order.

This process is essentially the same as the parallel evaluation of μ-formula shown by Hagiya, Arita, Kiga *et al.* A μ-formula is a Boolean formula in which each variable occurs at most once. Unlike other approaches towards evaluation of Boolean circuits with molecules [59, 74, 22], ours represent both Boolean formulas and their variable assignments with DNA molecules. Since the DNA molecules representing Boolean formulas can be regarded as programs, and since the molecules for variable assignments can be regarded as inputs, our approach allows multiple programs with different inputs to run in parallel. For its implementation strategy, readers are referred to our previous work [42].

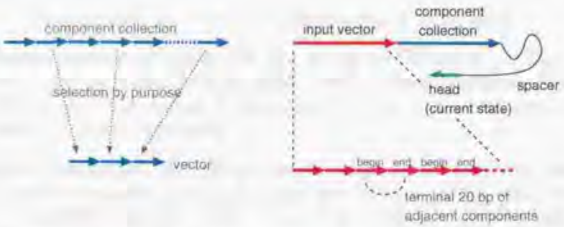


Figure 5.15: Implementation of DNA tuple synthesis

5.6.3 Experiments

Polymerization stop

We implemented the technique of polymerization stop using triplets as stopper sequences. This technique, however, requires encoding using only three deoxyribonucleotides.

The use of artificial nucleosides such as isoguanosine or 5-methylisocytidine [78] for stopper sequences resolves this difficulty. When all four common deoxyribonucleotides can be used for encoding, more information can be encoded in sequences of the same length. Another advantage is that it is easier to adjust the gc-contents at an appropriate level.

Successive transition

Two experiments of the whiplash model were performed: one was that of a single transition by extension of a hairpin structure, and the other was that of two successive transitions. Both processes were performed in a single tube by a simple thermal program consisting of cycles for denaturation, annealing, and polymerization.

In both experiments, symbols were encoded with three out of the four common deoxyribonucleotides (dATP, dGTP, dCTP and dTTP). When a polymerization buffer lacks the one kind of base out of the common four, a repetition of the missing deoxyribonucleotide works as a stopper sequence in polymerization. In the actual experiments, we used a triplet 'ggg' or 'aaa' as the stopper, in which case the corresponding buffer lacked dCTP or dTTP, respectively.

Both experiments were successful, but the successive transition was more inefficient than expected. It seems this difficulty comes from that the hybridization to the previously annealed site is thermodynamically more preferred than the hybridization priming polymerization for the second transition. This effect can be understood because the former case forms longer hybridization, which is more stable. This problem can be resolved by adjusting the temperature in an isothermal reaction, which will be discussed in the next section.

5.7 Discussion

This chapter introduced laboratory techniques for the systematic design of vectors.

The first two techniques dealt with the relational operations and the solid-phase synthesis of DNA tuples.

For relational operations, the following problems remain to be solved.

- **Maximum variety of tuples.** We only tested 2 × 2 Cartesian product, but much larger set must be tested for practicality. DNA of small population is likely to be exterminated in PCR process, so the estimation of the maximum size which can be processed at a time is necessary.

- **Selection technique.** We only considered exact matching in Join operation, but more complex matching procedure is necessary. DNA with an inosine can hybridize any of natural four bases, and can be used as a wild-card of DNA bases. For the matching using an inosine, more sophisticated data coding is required.

This approach is currently not under progress, mainly because PCR or gel electrophoresis requires more than an hour to perform. Also, its automation is hard to achieve in this manner.

In this respect, the solid-phase synthesis has more advantageous aspects than the relational-operation strategies, though its purpose is rather specific (synthesis only):

- The method is much faster. Only simple and fast DNA operations are employed, and the number of PCR application is minimized.

- It can be performed on a fully automated machine. The full automation is essential not only for the speedup of computation but for the error-free computation.

However, the examination of the feasibility of RMVDC operation and its improvement is still on the way. For a practical application, the establishment of its experimental conditions is a necessary step.

The new model of DNA computing is a remarkable progress over the previous models in that it can exhibit a great computational power in a single-tube reaction, and that the model can be used both in the synthesis of DNA tuples and in the logical computation with DNA molecules. The improvement of the experimental conditions for the whiplash model is still ongoing. The recent work by Sakamoto, Kiga, Komiya *et al.* showed that isothermal reactions greatly increase the efficiency of state transitions. They also reported the use of unnatural bases (isoguanosine and 5-methylisocytidine nucleotides) prevents unwanted annealings [84].

Our past experiments as well as those of Sakamoto, Kiga, Komiya *et al.* do not mix multiple automata or DNA strands, but the parallel execution of multiple automata necessitates that intra-molecular reaction occur exclusively. In experiments, single-stranded DNA were cooled down on ice to facilitate intra-molecular reaction. This issue is considered important; there is a report of the stable formation of a DNA hairpin *in vitro* [19], and the adjustment in experimental conditions greatly improves the efficiency and the accuracy of extension. So far, only two measures were employed in our experiments: (1) keeping the concentration of DNA molecules sufficiently low, and (2) quickly cooling down molecules during the annealing process. The effectiveness of these measures should be verified in future experiments. Another preventive measure is to use the surface chemistry. By binding DNA molecules sparsely on a streptavidin-coated magnetic beads, it is possible to avoid inter-molecular interaction.

The refinement of experimental conditions like these should be further pursued for accessing the feasibility and the prospect of DNA computing.

Materials and Methods

5.7.1 Relational Operations

Sequence Design

An optimal sequence for computation should not form secondary structure by itself, and should bind only to the specific site where it is expected to anneal. Short DNA such as PCR primers can satisfy this condition, but this condition is hard to satisfy for longer DNA. Sequences were designed to satisfy the following conditions. **1.** Both 5 bp ends of each data (or tag) do not appear in the ends of other data (or tag). **2.** The GC content of each data (or tag) is not greatly biased (GC = 33 ∼ 76%). **3.** A tag sequence does not form a stable structure by itself. **4.** Single stranded DNA for generating each tuple anneals at the expected sites.

The computer program designed by Akira Suyama (`secdyn2` in Fortran77) was performed for the secondary structural analysis to check the third and the fourth conditions above. This program calculates the optimal energy of two annealing DNAs.

Data Representation

Each tuple was made from single stranded DNA of 60 bp or 45 bp, which were mail-ordered and PAGE purified. A sequence for a primer is either 15 bp or 30 bp long, and is purified with OPC (Sawaday Technology, Japan). The sequence number used in Fig 5.2 and in later sections, e.g. `seq1` or `seq2`, indicates the sequence in the following table.

DNA sequences (forward strands only)

```
1: TACATAAAGTACCTT   2: ACACCTCTCTGCAAA   3: ACAGACTGTACGCGA
4: TGAACTTTACCTCTA   5: TGGATCCTCGAGGGC   6: GCCCGACGCATTGTT
7: AGCCTACCTCCCTAA   8: ATGAGGTACCTCCGC   9: TAGTATGAAAGCCAT
10:GCCACTCTCCTCGAC   11:TACGTGCCGTGTGCT   12:AAGACTTTTAATCTA
13:GCCCACGGAGACCCA   14:TCGCAGGGACAACGA   15:AAGAATCAGAGGATA
16:TAGAATTCATCCGGC   17:GTCAAAGTTCGAGTT   18:TGGGACGTGCGTGAT
19:GTCATGGAGCCGTTA   20:AAGAGCTTGGAGTGT   21:TAGTTGGAGACTGCT
22:TTCCTCGCCGAGCTA
```

Molecular Analysis

In total, six experiments were performed: **1.** simple polymerization or **2.** PCR from single-stranded templates for making tuples; **3.** concatenation with overlaps or **4.** with bridges; **5.** rotation with biotin or **6.** without it.

Each PCR amplification was performed on Biometra UNO-Thermoblock in a total volume of 100 μl using the following procedure (except for simple polymerization): initial denaturation step, 95 ℃ for 1 min; 25 cycles with a denaturation step at 95 ℃ for 30 s, annealing at 40 ℃ for 1 min if primers are 15 bp (and at 60 ℃ if primers are 30 bp), and extension at 72 ℃ for 1 min. There was no final extension. For each amplification, 5 units of *Taq* DNA polymerase (except for simple polymerization, 10 units; Ex *Taq* DNA polymerase, TAKARA Japan) was used together with 50 pmol of each specific primer, 2.5 mM dNTPs. The concentration of templates in each amplification was 100 fmol/100 μl if not explicitly mentioned. All products were analyzed using capillary electrophoresis system (BECKMAN P/ACE 5510) with LIFluor daDNA1000Kit and with System Gold, The Personal Chromatograph analyzing software. Its output is a graph with 'relative fluorescence unit' (RFU) in y-axis and elution time in x-axis. The amount of double-stranded DNA was measured by RFU; for DNA of 120 bp, 10 RFU corresponds to about

5 fmol/μl. The length of DNA was measured by the elution time in comparison with that of molecular markers.

Simple polymerization

The generation of each tuple was performed with the following procedure: initial denaturation step, 95 ℃ for 1 min; 20 cycles with annealing at 40 ℃ for 1 min, and extension at 72 ℃ for 1 min. The concentration of templates was 40 pmol each with 10 units of *Taq* DNA polymerase.

PCR from single-stranded templates

In Fig 5.4, a tuple `A` was amplified with a reverse primer of 30 bp, which consisted of the 15 bp end of a tuple `A` and the 15 bp beginning of a tuple `C` (or `D`). The overhanging 15 bp in this primer was later used as a sticky bridge in the concatenation. If necessary, each tuple was elongated by 15 bp on either or both ends. The annealing temperature in PCR was 40 ℃.

Circularization in diluted solution

Two concatenated tuples `A-C-E` and `A-D-E` were separately amplified by PCR with a phosphorylated primer `fwd1` and with a primer `rev18` which was not phosphorylated. Primers were removed by filtration (Millipore UFC3TGC, 8400 RPM). A tuple `A-C-E` and a tuple `A-D-E` (150 fmol/100 μl each) were incubated for 3 h (37 ℃) with 800 units of T4 DNA ligase (New England Biolabs, USA) and with 10 μl of 10× buffer in a total volume of 100 μl. The product of ligation was amplified by PCR with three set of primers: `fwd1-2` and `rev17-18` for checking linear DNA, `fwd10-11` and `rev8-9` for checking circularized DNA, and `fwd10-11` and `rev21-9` for checking concatemers of mixed data.

Using biotin for circularization

Streptavidin-coated magnetic beads (20 μl, Dynabeads-M280 streptavidin, Veritas) were washed twice with binding and washing buffer (B&W buffer: 2 mM NaCl in TE buffer) and were suspended in 40 μl of BW buffer. A concatenated tuple `A-C-E` was amplified by PCR with a phosphorylated primer `fwd1` and a non-phosphorylated primer `rev18` in the presence of biotin-16-dUTP (1 mM, Böhringer Mannheim). Some uracils with biotin were integrated, instead of thymins, into amplified double-stranded DNA. The PCR product (40 μl, 0.1 mM) was mixed with these beads (40 μl) and was slowly shaken for 15 min. After the beads were pelleted by a magnet, clear upper part of this solution was removed and beads were resuspended in B&W buffer. The beads were washed twice in this way. The pelleted beads were incubated with ligases for 2 h (37 ℃) in the same way as in Section 5.7.1. The product of ligation was amplified by PCR with three set of primers: `fwd1-2` and `rev17-18` for checking linear DNA, `fwd10-11` and `rev8-9` for checking circularized DNA, and `fwd10-11` only for checking head-to-head concatemers.

5.7.2 Stepwise Generation

Sequence Design

Sequences were designed to satisfy the following conditions. **1.** Sequences for units and `RS` and `TS` share no common subsequences of more than 5 bases long in order to prohibit inaccurate association of single-stranded DNA oligomers in a reaction mixture. **2.** The GC content of each data (or tag) is not greatly biased (GC = 33 ~ 76%). **3.** A tag sequence does not form a stable structure by itself. **4.** Single stranded DNA for generating each tuple anneals at the expected sites.

Data Representation

Single-stranded DNA oligomer representing the initial unit was 40 bases long: the 5' end part (25 bases) for `RS` and the 3' end part (15 bases) for the `start`. This initial oligomer is designated by `RS-start`. The 5' end of `RS-start` is biotinylated to attach on streptavidin-coated magnetic beads through the biotin-streptavidin interaction. The `RS` part has the restriction site of *Eco*RI.

Single-stranded DNA oligomers representing other units than the initial unit were also 40-base long with two parts but with a different structure. The unique 5' end part (15 bases) was used to specify unit *i* (`U`*i*) itself. The 3' end part was the `TS` (25 bases), which was used to block coupling of more than one unit to the end of extending tuples while executing EXT operation. TS part includes a *Bam*HI restriction site so that it can be removed in DEBLK operation.

The sequences of unit oligomers are as follows:

RS-U0:	5'-b-aaacgtctctccacgagcgcGAATT-Cggcctacttaagag-3'
U1-TS:	5'-p-ttgttacgcagcccG-GATCCataggagactaagagagacg-3'
U2-TS:	5'-p-attgccgaggtactG-GATCCataggagactaagagagacg-3'
U3-TS:	5'-p-taccgaaagtatgaG-GATCCataggagactaagagagacg-3'
U4-TS:	5'-p-tcgtgtgccgtgcaG-GATCCataggagactaagagagacg-3'
U5-TS:	5'-p-tgtgaggttcgagaG-GATCCataggagactaagagagacg-3'
U6-TS:	5'-p-ttgagcttgaaactG-GATCCataggagactaagagagacg-3'

where '`b`' and '`p`' designate a biotin and a phosphate group, respectively. The recognition sites of *Bam*HI and *Eco*RI are capitalized. The sites cleaved with these restriction enzymes are designated by '–'.

The bridging sequences were the reverse complement of the two unit sequences to be connected lining up side by side. For example, the sequence of bridging oligomer connecting `fwdU0` and `fwdU1` is the reverse complement of the unit sequences of `fwdU0-U1`, and is designated `revU0-U1`. Thus, `revU0-U1` is `5'-cgggctgcgtaacaactcttaagtaggccg-3'`. All DNA oligomers, synthesized on an automated DNA synthesizer and purified with an OPC cartridge, were obtained from Sawaday Technology, Inc. (Japan).

Ext operation

First of all, single-stranded initial oligomer `fwdRS-U0` was immobilized on streptavidin-coated magnetic beads. Twenty microliters (containing 20 μg) of Dynabeads M-280 Streptavidin (DYNAL A. S., Norway) were washed twice with 100 μl of binding and washing buffer (B&W buffer: TE buffer (10 mM Tris-HCl pH 7.6, 0.1 mM EDTA) containing 1 M NaCl) using a Dynal MPC, and resuspended in 50 μl of B&W buffer. Ten pmol of biotinylated `fwdRS-U0` in an equal volume of B&W buffer was added and incubated at 25 ℃ for 15 min while gently shaking the tube. The beads were then washed once with 100 μl of B&W buffer and twice with 100 μl of TE buffer at room temperature.

For EXT operation, beads were mixed with 1 pmol each of unit oligomers `fwdU`*i*`-TS` ($i = 1, 2, \cdots 6$) and bridging oligomers `revU`*i*`-U`*j* (i, j: all pairs of connected units) in 50 μl of *Taq* ligation buffer (20 mM Tris-HCl pH 7.6, 25 mM CH_3COOK, 10 mM $(CH_3COO)_2Mg$, 10 mM DTT, 1mM NAD, 0.1% Triton X-100). To avoid a misligation, the reaction mixture was heated up to 94 ℃ before an addition of 20 units of *Taq* DNA ligase (NEW ENGLAND

BIOLABS, USA). The beads were then incubated at 65℃ for 10 min while gently shaking the tube. After the ligation reaction, the beads were washed twice with 100 μl of TE buffer at 94 ℃ to remove excess units and bridging oligomers.

Cap operation

Beads were incubated at 37 ℃ for 10 min in 50 μl of cacodylate buffer (120 mM sodium cacodylate pH 7.2, 1.2 mM CoCl$_2$, 0.6 mM 2-mercaptoethanol) containing 10 nmol of dideoxyribosyl thymidine triphosphate (ddTTP) and 20 units of terminal deoxynucleotidyl transferase (TdT) (TOYOBO, Japan). After the capping reaction, the beads were washed twice with 100 μl of TE buffer at room temperature.

Deblk operation

Beads were mixed with 5 pmol each of revUi-Uj-TS oligomers ($i = 1, 2, \cdots, 5$) in 50 μl of H buffer (50 mM Tris-HCl pH7.5, 100 mM NaCl, 10 mM MgCl$_2$, 1 mM DTT). The reaction mixture was heated to 94 ℃ for 1 min and then cooled down to 37 ℃ in 5 min to facilitate annaling of the rev Ui-Uj-TS oligomers to tuples on the beads. After adding 200 units of *Bam*HI, the beads were incubated at 37 ℃ for 5 min. The beads were then washed twice with 100 μl of TE buffer at 94 ℃, making the tuples single-stranded.

Rmvdc operation

Beads were suspended in 50 μl of Ex *Taq* buffer (TAKARA, Japan) containing 4 μl each of the 0.2 mM dNTPs and 5 pmol each of rev Ui-Ui-TS oligomers ($i = 1, 2, \cdots, 5$), which anneal to tuples on the beads. The reaction mixture was heated up to 94 ℃ and 1.25 units of Ex *Taq* DNA polymerase (TAKARA, Japan) was added to initiate the polymerization. The beads were then incubated at 80 ℃ for 1 min. After stopping the polymerization reaction with the addition of 2 μl o f 500 mM EDTA, the beads were washed once with 100 μl of TE buffer at room temperature. The beads were then resuspended in 50 μl of H buffer and treated for 5 min at 37 ℃ with 200 units of *Eco*RI to remove tuples with duplicate units from the beads. After an *Eco*RI digestion, the beads were washed twice with 100 μl of TE buffer at 94 ℃, making the tuples single-stranded.

PCR amplification

Beads were subjected to a PCR in 50 μl of Ex *Taq* buffer containing 4 μl each of the 0.2 mM dNTPs and 10 pmol each of fwdRS and revTS primers. PCR amplification was performed on UNOII Thermocycler (BIOMETRA, Germany) using the temperature profile of the initial denaturation at 94 ℃ for 1 min followed by 20 cycles of the denaturation at 94 ℃ for 20 sec, the annealing at 40 ℃ for 1 min, and the extension at 72 ℃ for 1 min. The hot start technique was used, so 1.25 units of Ex *Taq* DNA polymerase was added just after the initial denaturation step.

After the first PCR amplification, 1 μl of the supernatant was subjected to the second PCR in the same way as the first PCR but the fwdRS primer biotinylated at the 5' end was used. After ethanol precipitation and removal of excess primers by centrifugal filtration, the PCR product was suspended in 50 μl of B&W buffer and then attached to 20 μg of streptavidin-coated magnetic beads through biotin at the 5' end of the fwdRS primer strand. The double-stranded tuples immobilized on the beads were denatured in 100 μl of TE buffer for 2 min at 94 ℃. Dissociated strands were washed away twice with 100 μl of TE buffer at 94 ℃.

s1	tccattctaa
s2	ctcacccttatacat
s3	cgtttcag
ini	cgacaagcttggg

s1	gcacgatctaggaaa
s2	tcccgttctgggcct
s3	gtggtttgcgctcgt
ini	ccaaa

Table 5.3: Sequences for one-step (left) and two-step (right) transitions

PCR was used again for obtaining the generated tuples with primers fwdRS and revTS. The beads were suspended in 50 μl of Ex *Taq* buffer containing 4 μl each of the 0.2 mM dNTPs and 10 pmol each of the primers. PCR amplification was performed using the same temperature profile and the same hot start technique as above.

PCR products in the supernatant were resolved on a P/ACE5510 capillary electrophoresis system (BECKMAN, USA) using LIFluor dsDNA1000 Kit. The laser-induced fluorescence detection equipped with a P/ACE5510 was used for the high sensitivity detection of PCR products. The intensity of fluorescence signal was calibrated using a double-stranded DNA solution, the concentration of which was determined by a UV absorbance at 260 nm. The peak area of 700 RFU·ETU (RFU: relative fluorescence unit; ETU: elution time unit) on an electropherogram corresponded to 20 fmol/μl double-stranded DNA solution of 40 bp. The elution time of PCR products on an electropherogram was calibrated using the standard molecular weight marker supplied with LIFluor dsDNA1000 Kit.

5.7.3 DNA State Machine

Sequence Design

For the single-step transition, the triplet 'ggg' was used as a stopper sequence, while for the two successive transitions, the triplet 'aaa' was used for the stopper (Table 5.3). All the sequences are single-stranded if not specially mentioned. A sequence with a bar indicates its Watson–Crick complement. For example, $\overline{\texttt{s1}}$ represents the complementary sequence of s1. A hyphen between sequences indicates their concatenation, i.e., phosphodiester bonding between the sequences. For example, s1-s2 represents the concatenated sequence of s1 and s2 in this order.

The sequence $\texttt{ini-s1-s2-s3-}\overline{\texttt{s2}}$ is chemically synthesized in this experiment. Note that the subsequence ini contains the stopper ggg. Two more sequences are synthesized as markers for electrophoresis ($\texttt{ini-s1-s2-s3-}\overline{\texttt{s2}}\texttt{-}\overline{\texttt{s1}}$ and $\texttt{ini-s1-s2-s3-}\overline{\texttt{s2}}\texttt{-}\overline{\texttt{s1}}\texttt{-}\overline{\texttt{ini}}$).

Single Transition

The buffer in this experiment lacks dCTP. In 20 μl scale, the buffer contains 2 μg of DNA, 250 μM of dATP, dGTP and dTTP, and 2 units of Ampli*Taq* DNA polymerase (PERKIN ELMER, USA). Other conditions are the same as given by the supplier.

For comparison, we also prepared the buffer containing 250 μM of dCTP together with other three deoxyribonucleotides. In this case, it is expected that polymerization does not stop and produces $\texttt{ini-s1-s2-s3-}\overline{\texttt{s2}}\texttt{-}\overline{\texttt{s1}}\texttt{-}\overline{\texttt{ini}}$.

Transition was performed by initial denaturing at 90 ℃ for 3 min, followed by an extension step, 68 ℃ for 10 min, using PERKIN ELMER DNA Thermal Cycler 480. Finally, the reaction mixture was cooled down on ice.

Electrophoresis

Samples were analyzed by electrophoresis on 12% polyacrylamide gels with or without 8 M urea. The band of the transition product was compared with those of the two markers: one is ini-s1-s2-s3-$\overline{\mathtt{s2}}$-$\overline{\mathtt{s1}}$ and the other is ini-s1-s2-s3-$\overline{\mathtt{s2}}$-$\overline{\mathtt{s1}}$-$\overline{\mathtt{ini}}$.

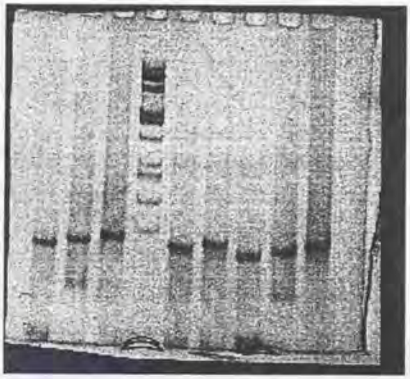


Figure 5.16: Electrophoresis without urea

The result of electrophoresis without urea is shown in Fig.5.16. From the left, the lanes correspond to:

(1) the initial DNA ini-s1-s2-s3-$\overline{\mathtt{s2}}$,
(2) the marker ini-s1-s2-s3-$\overline{\mathtt{s2}}$-$\overline{\mathtt{s1}}$,
(3) the marker ini-s1-s2-s3-$\overline{\mathtt{s2}}$-$\overline{\mathtt{s1}}$-$\overline{\mathtt{ini}}$,
(4) some known standard markers,
(5) the polymerization product with three deoxyribonucleotides,
(6) the polymerization product with four deoxyribonucleotides,
(7) the initial DNA ini-s1-s2-s3-$\overline{\mathtt{s2}}$,
(8) the marker ini-s1-s2-s3-$\overline{\mathtt{s2}}$-$\overline{\mathtt{s1}}$, and
(9) the marker ini-s1-s2-s3-$\overline{\mathtt{s2}}$-$\overline{\mathtt{s1}}$-$\overline{\mathtt{ini}}$.

In lanes (1), (2) and (3), DNA molecules were dissolved in water and renatured (cooled down on ice). Lanes (7), (8) and (9) contain the same molecules as in (1), (2) and (3), respectively, but they were dissolved in the polymerization buffer without AmpliTaq DNA polymerase; they were also renatured before electrophoresis.

The DNA molecules in these lanes each produced a single sharp band, indicating that they each formed a single stable structure, probably a hairpin structure. In contrast, the DNA molecules which were not renatured showed a smeared band as well as another band unmoved during the electrophoresis. These bands probably indicate the formation of various aggregates (data not shown).

In lane (5), the band of the transition product corresponds to that of the marker ini-s1-s2-s3-$\overline{\mathtt{s2}}$-$\overline{\mathtt{s1}}$ in (8). On the other hand, there is no band corresponding to that of ini-s1-s2-s3-$\overline{\mathtt{s2}}$-$\overline{\mathtt{s1}}$-$\overline{\mathtt{ini}}$ in (9). This means that hairpin structures were properly formed and their extension was stopped after $\overline{\mathtt{s1}}$ was copied. On the other hand, the extension with all four deoxyribonucleotides produced a band in lane (6), corresponding to the marker in lane (9).

Denaturing gel electrophoresis with 8 M urea exhibited a pattern similar in correspondence between the transition products and the markers to Fig.5.16 (data not shown).

Double Transition

The sequence used is `ini-s2-s1-s3-s2-`$\overline{\mathtt{s1}}$ as defined in Table 5.3. The expected extension is shown in Fig.5.17.

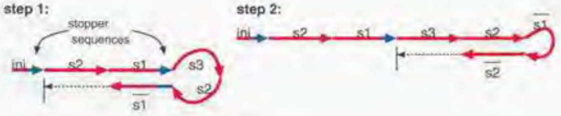


Figure 5.17: Expected extension

The buffer used in this experiment lacks dTTP. In 20 μl scale, the buffer contains 1 μg of DNA, 250 μM of dATP, dGTP and dCTP, and 2 units of Ampli*Taq* DNA polymerase (PERKIN ELMER, USA). Other conditions are the same as given by the supplier.

We programmed PERKIN ELMER DNA Thermal Cycler 480 as follows: initial denaturation step, 90 °C for 1 min; extension step, 68 °C for 1 min; four cycles with denaturation step at 90 °C for 1 min, cooling down on ice for 1 min, incubation at 40 °C for 30 sec, and extension at 68 °C for 30 sec. The cooling step is for forming hairpin structures before extension.[3]

Electrophoresis

Primer 1	`gctcgtcgac`
	`gcacgatctaggaaa`
	`gtgg`

Primer 2	`gacggaattc`
	`aaaaaaaaaaaaaaaaaaaa`
	`gtgg`

Table 5.4: Primers for cloning

Samples were analyzed by electrophoresis on 12% polyacrylamide gels containing 8 M urea. The bands of the transition products were compared with those of the markers, which had been chemically synthesized.

The product of the first extension step exhibited two bands in electrophoresis. One is that of `ini-s1-s2-s3-`$\overline{\mathtt{s2}}$. The other is considered that of `ini-s1-s2-s3-`$\overline{\mathtt{s2}}$`-`$\overline{\mathtt{s1}}$.

After the four cycles of denaturation, annealing, and cooling, the above two bands disappeared and the band corresponding to that of the marker `ini-s2-s1-s3-s2-`$\overline{\mathtt{s1}}$`-`$\overline{\mathtt{s2}}$`-`$\overline{\mathtt{s3}}$ emerged.

[3] When cooled down, tubes were temporarily detached from the thermal cycler and put on ice.

The DNA molecules that have undergone the successive transitions was processed by terminal deoxynucleotidyl transferase, and thus a poly-T sequence was added to the tail of each DNA molecule. It was then amplified by PCR with the primers of the sequences shown in Table 5.4. Primer 1 includes `s1` and Primer 2 has a poly-A sequence. The product of PCR was digested with *Eco*RI and *Sal*I, and was ligated to the corresponding sites in vector pUC119. The sequences of the clones were determined, confirming that the successive transitions certainly occurred as we expected.

Bibliography

[1] L Adleman. Molecular Computation of Solutions to Combinatorial Problems. *Science*, 266:1021–1024, 1994.

[2] KK Agarwal. An Algorithm for Computing the Automorphism Group of Organic Structures with Stereochemistry and a Measure of its Efficiency. *J Chem Inf Comput Sci*, 38:402–404, 1998.

[3] KK Agarwal and HL Gelernter. A Computer-Oriented Linear Canonical Notational System for the Representation of Organic Structures with Stereochemistry. *J Chem Inf Comput Sci*, 34:463–479, 1994.

[4] T Akutsu. A New Method of Computer Representation of Stereochemistry: Transforming a Stereochemical Structure into a Graph. *J Chem Inf Comput Sci*, 31:414–417, 1991.

[5] T Akutsu. An $O(n^2)$ Time Algorithm for Computing a Canonical Form of a Chemical Structure Which Has a Planar Graph Structure. *Trans Info Process Soc Japan (in Japanese)*, 33(12):1486–1496, 1992.

[6] T Akutsu. A Polynomial Time Algorithm for Finding a Largest Common Subgraph of almost Trees of Bounded Degree. *IEICE Trans Fund Electr Comm Comput Sci*, 9:1488–1493, 1993.

[7] M Amos, A Gibbons, and D Hodgson. Error-resistant Implementation of DNA Computations. In L Landweber and EB Baum, editors, *DNA based computers II*, volume 27 of *DIMACS series in Discret Math and Theor Comp Sci*, pages 151–162. AMS, 1996.

[8] M Arita. SIMFLY2: Simulation of a Fly Embryo. In *Proc 6th Genome Informatics Workshop*, pages 29–38. Universal Academy Press, 1995.

[9] M Arita, M Hagiya, and T Shiratori. GEISHA system: an Environment for Simulating Protein Interaction. In *Proc 5th Genome Informatics Workshop*, pages 80–89. Universal Academy Press, 1994.

[10] M Arita, M Hagiya, and A Suyama. Joining and Rotating Data with Molecules. In *Proc IEEE 4th Intern Conf on Evolutional Computation (ICEC'97)*, pages 243–248, 1997.

[11] M Arita, A Suyama, and M Hagiya. A Heuristic Approach for Hamiltonian Path Problem with Molecules. In JR Koza et al., editors, *Proc 2nd Annual Conf on Genetic Programming*, pages 457–462. Morgan Kaufmann, 1997.

[12] R Balducci and RS Pearlman. Efficient and Exact Solution of the Ring Perception Problem. *J Chem Inf Comput Sci*, 34:822–831, 1994.

[13] N Barkai and S Leibler. Robustness in Simple Biochemical Networks. *Nature*, 387:913–917, 1997.

[14] E Baum. Building an Associative Memory Vastly Larger Than the Brain. *Science*, 268:583–585, 1995.

[15] MW Bern, EL Lawler, and AL Wong. Linear-Time Computation of Optimal Subgraphs of Decomposable Graphs. *J Algor*, 8:216–235, 1987.

[16] D Boneh, R Lipton, and C Dunworth. Breaking DES using a Molecular Computer. In RJ Lipton and EB Baum, editors, *DNA based computers I*, volume 27 of *DIMACS series in Discret Math and Theor Comp Sci*, pages 37–66. AMS, 1995.

[17] D Bray. Protein Molecules as Computational Elements in Living Cells. *Nature*, 376:307–312, 1995.

[18] D Bray, BR Bourret, and IM Simon. Computer Simulation of the Phosphorylation Cascade Controlling Bacterial Chemotaxis. *Mol Bio of the Cell*, 4:469–482, 1993.

[19] RR Breaker and GF Joyce. Emergence of a Replicating Species from an *in vitro* RNA Evolution Reaction. *Proc Natl Acad Sci USA*, 91:6093–6097, 1994.

[20] KJ Breslauer, R Frank, H Blöcker, and LA Marky. Predicting DNA Duplex Stability from the Base Sequence. *Proc Natl Acad Sci USA*, 83:3746–3750, 1986.

[21] T Bugg. *An Introduction to Enzyme and Coenzyme Chemistry*. Blackwell, 1997.

[22] W Cai, AE Condon, RM Corn, et al. The Power of Surface-Based DNA Computation. In *Proc 1st Ann Intern Conf on Comput Mol Bio (RECOMB '97)*, pages 67–74, 1997.

[23] RE Carhart. Erroneous Claims Concerning the Perception of Topological Symmetry. *J Chem Inf Comput Sci*, 18:108–110, 1978.

[24] MJ Chung. $O(N^{2.5})$ time Algorithms for the Subgraph Homeomorphism Problem on Trees. *J Algor*, 8:106–112, 1987.

[25] DG Corneil and CC Gotlieb. An Efficient Algorithm for Graph Isomorphism. *J Assoc Comp Mach*, 17(1):51–64, 1970.

[26] DG Corneil and DG Kirkpatrick. A Theoretical Analysis of Various Heuristics for the Graph Isomorphism Problem. *SIAM J Comput*, 9(2):281–297, 1980.

[27] A Cornish-Bowden. *Fundamentals of Enzyme Kinetics*. Portland Press, 1995.

[28] FHC Crick. *What Mad Pursuit*. Basic Books, 1988.

[29] MI Davis and ML Ellzey Jr. A Technique for Determining the Symmetry Properties of Molecular Graphs. *J Comput Chem*, 4(2):267–275, 1983.

[30] C DeLisi and DM Crothers. Prediction of RNA Secondary Structure. *Proc Natl Acad Sci USA*, 68:2682–2685, 1971.

[31] TM Devlin. *Textbook of Biochemistry with Clinical Correlations*. Wiley-liss, 4th edition, 1997.

[32] GM Downs, VJ Gillett, JD Holliday, and MF Lynch. Review of Ring Perception Algorithms for Chemical Graphs. *J Chem Inf Comput Sci*, 29:172–187, 1989.

[33] D Eppstein. Finding the k Shortest Paths. In *Proc 25th Ann Symp on Foundations of Comp Sci (FOCS'94)*, pages 154–165, 1994.

[34] BT Fan, A Barbu, A Panaye, and J Doucet. Detection of Constitutionally Equivalent Sites from a Connection Table. *J Chem Inf Comput Sci*, 36:654–659, 1996.

[35] BT Fan, A Panaye, J Doucet, and A Barbu. Ring Perception. A New Algorithm for Directly Finding the Smallest Set of Smallest Rings from a Connection Table. *J Chem Inf Comput Sci*, 33:657–662, 1993.

[36] J Faulon. Isomorphism, Automorphism Partitioning, and Canonical Labeling Can Be Solved in Polynomial-Time for Molecular Graphs. *J Chem Inf Comput Sci*, 38:432–444, 1998.

[37] J Figueras. Ring Perception Using Breadth-first Search. *J Chem Inf Comput Sci*, 36:986–991, 1996.

[38] RD Fleischmann, MD Adams, O White, RA Clayton, EF Kirkness, AR Kerlavage, CJ Bult, JF Tomb, BA Dougherty, JM Merrick, et al. Whole-genome Sequencing and Assembly of *Haemophilus influenzae* rd. *Science*, 269:496–512, 1995.

[39] S Fortune, J Hopcroft, and J Wyllie. The Directed Subgraph Homeomorphism Problem. *Theor Comput Sci*, 10:111–121, 1980.

[40] MR Garey and DS Johnson. *Computers and Intractability*. W.H. Freeman, 1979.

[41] S Goto, T Nishioka, and M Kanehisa. LIGAND Database for Enzymes, Compounds, and Reactions. *Nucleic Acids Res*, 27:377–379, 1999.

[42] M Hagiya, M Arita, D Kiga, K Sakamoto, and S Yokoyama. Towards Parallel Evaluation and Learning of Boolean μ-Formulas with Molecules. In H Rubin and DH Wood, editors, *DNA Based Computers III*, volume 48 of *DIMACS series in Discret Math and Theor Comp Sci*, pages 57–72. AMS, 1999.

[43] K Hayashi and N Sakamoto. *Dynamic Analysis of Enzyme Systems*. Springer, 1986.

[44] JE Hopcroft and RE Tarjan. A $V \log V$ Algorithm for Isomorphism of Triconnected Planar Graphs. *J Comput Sys Sci*, 7:323–331, 1973.

[45] K Horikoshi, T Toraya, T Kitazume, and R Aono. *Enzyme – Its Science and Technology (in Japanese)*. Kodan-sha, 1992.

[46] JD Horton. A Polynomial-time Algorithm to Find the Shortest Cycle Basis of a Graph. *SIAM J Comput*, 16(2):358–366, 1987.

[47] E Hubicka and MM Syslo. Minimal Bases of Cycles of a Graph. In M Fiedler, editor, *Recent Advances in Graph Theory, Proc of the symp in Prague (Jun 1974)*, pages 283–293. Academia Praha, 1975.

[48] A Itai and M Rodeh. Finding a Minimum Circuit in a Graph. *SIAM J Comput*, 7(4):413–423, 1978.

[49] C Jochum and J Gasteiger. Canonical Numbering and Constitutional Symmetry. *J Chem Inf Comput Sci*, 17:113–117, 1977.

[50] DB Johnson. Finding All the Elementary Circuits of a Directed Graph. *SIAM J Comput*, 4(1):77–84, 1975.

[51] HF Judson. *The Eighth Day of Creation*. Simon & Schuster, 1996.

[52] V Kann. On the Approximability of the Maximum Common Subgraph Problem. In *Proc 9th Symp Theor Aspects of Comp Sci (LNCS 577)*, pages 377–388. Springer, 1992.

[53] T Kazic. Representing Biochemistry for Modeling Organisms. In TF Kumosinksi and MN Liebman, editors, *Mol Modeling from Virtual Tools to Real Problems*, pages 486–494. Amer Chem Soc, 1994.

[54] J Köbler, U Schöning, and J Toràn. *The Graph Isomorphism Problem*. Birkhäuser, 1993.

[55] S Kondo and R Asai. A Reaction-diffusion Wave on the Skin of the Marine Angelfish *Pomacanthus*. *Nature*, 376:765–768, 1995.

[56] AS LaPaugh and RL Rivest. The Subgraph Homeomorphism Problem. *J Comput Sys Sci*, 20:133–149, 1980.

[57] BME Linda, MS Stuart, and R McLeish. Representing Metabolic Pathway Information: An Object-oriented Approach. *Bioinformatics*, 14(9):803–806, 1998.

[58] ME Linder and AG Gilman. G-proteins. *Scientif Amer*, July 1992.

[59] R Lipton. DNA Solution of Hard Computational Problems. *Science*, 268:542–545, 1995.

[60] X Liu and DJ Klein. The Graph Isomorphism Problem. *J Comput Chem*, 12(10):1243–1251, 1991.

[61] P Mateti and N Deo. On Algorithms for Enumerating All Circuits of a Graph. *SIAM J Comput*, 5(1):90–99, 1976.

[62] HH McAdams and L Shapiro. Circuit Simulation of Genetic Networks. *Science*, 269:650–656, 1995.

[63] JJ McGregor. Backtrack Search Algorithms and the Maximal Common Subgraph Problem. *Software-Pract and Exper*, 12:23–34, 1982.

[64] K Mehlhorn and S Näher. LEDA: A Platform for Combinatorial and Geometric Computing. *Comm Assoc Comput Mach*, 38(1):96–102, 1995.

[65] H Meinhardt. Models for Maternally Supplied Positional Information and the Activation for Segmentation Genes in *Drosophila*. *Development*, 104:95–110, 1988.

[66] S Meyers and P Friedland. Knowledge-based Simulation of Genetic Regulation in Bacteriophage Lambda. *Nucleic Acids Res*, 8(1):1–9, 1980.

[67] J Michalka and SH Goodgal. Genetic and Physical Map of the Chromosome of *Haemophilus influenzae*. *J Mol Biol*, 45(2):407–421, 1969.

[68] AG Moat and JW Foster. *Microbial Physiology*. Wiley-liss, 3rd edition, 1995.

[69] HL Morgan. The Generation of a Unique Machine Description for Chemical Structures – A Technique Developed at Chemical Abstracts Service. *J Chem Doc*, 5:107–113, 1965.

[70] N Morimoto, M Arita, and A Suyama. Stepwise Genaration of Hamiltonian Path with Molecules. In *Proc 1st Intern Conf on Biocomputing and Emergent Computation (BCEC '97)*, pages 184–192. World Scientific, 1997.

[71] N Morimoto, M Arita, and A Suyama. Solid Phase DNA Solution to the Hamiltonian Path Problem. In H Rubin and DH Wood, editors, *DNA Based Computers III*, volume 48 of *DIMACS series in Discret Math and Theor Comp Sci*, pages 193–206. AMS, 1999.

[72] JD Murray and SA Levin, editors. *Mathematical Biology*, volume 19 of *Biomathematics*. Springer, 1993.

[73] H Ogata, S Goto, K Sato, W Fujibuchi, H Bono, and M Kanehisa. KEGG: Kyoto Encyclopedia of Genes and Genomes. *Nucleic Acids Res*, 27:29–34, 1999.

[74] M Ogihara and A Ray. Simulating Boolean Circuits on a DNA Computer. Technical Report 631, University of Rochester, 1996.

[75] O Owolabi. An Efficient Graph Approach to Matching Chemical Structures. *J Chem Inf Comput Sci*, 28:221–226, 1988.

[76] MF Paretti, RT Kroemer, JH Rothman, and WG Richards. Alignment of Molecules by the Monte Carlo Optimization of Molecular Similarity Indices. *J Comput Chem*, 18:1344–1351, 1997.

[77] JJ Perry and JT Staley. *Microbiology*. Saunders College Pub, 1997.

[78] JA Piccirilli, T Krauch, SE Moroney, and SA Benner. Enzymatic Incorporation of a New Base Pair into DNA and RNA Extends the Genetic Alphabet. *Nature*, 343:33–37, 1990.

[79] M Randič, GM Brissey, and CL Wilkins. Computer Perception of Topological Symmetry via Canonical Numbering of Atoms. *J Chem Inf Comput Sci*, 21:52–59, 1981.

[80] J Reinitz and D Sharp. Mechanism of *eve* Stripe Formation. Technical Report LA-UR-94-1915, Los Alamos National Lab, 1994.

[81] DH Rouvray. Similarity in Chemistry: Past, Present and Future. *Topics Curr Chem*, 173:1–30, 1995.

[82] G Rücker and C Rücker. Computer Perception of Constitutional (Topological) Symmetry: TOPSYM, a Fast Algorithm for Partitioning Atoms and Pairwise Relations among Atoms into Equivalence Classes. *J Chem Inf Comput Sci*, 30:187–191, 1990.

[83] Y Sakaki. *Vector DNA (in Japanese)*. Kodan-sha, 1986.

[84] K Sakamoto, D Kiga, Ken Komiya, H Gouzu, S Yokoyama, S Ikeda, H Sugiyama, and M Hagiya. State Transitions by Molecules. In *Proc 4th meeting on DNA based computers*, pages 87–99, 1998.

[85] JG Salway. *Metabolism at a Glance*. Blackwell, 1994.

[86] J Sambrook, F Fritch, and T Maniatis. *Molecular Cloning: A Laboratory Manual*. Cold Spring Harbor Lab, 2nd edition, 1989.

[87] A Sayre. *Rosalind Franklin and DNA*. Norton Library, 1978.

[88] DC Schmidt and LE Druffel. A Fast Backtracking Algorithm to Test Directed Graphs for Isomorphism Using Distance Matrices. *J Assoc Comp Mach*, 23(3):433–445, 1976.

[89] N Seeman, H Wang, B Liu, et al. The Perils of Polynucleotides: the experimental gap between the design and assembly of unusual DNA structures. In *Proc 2nd meeting on DNA based computers*, pages 191–205, 1996.

[90] E Selkov, M Galimova, I Goryanin, Y Gretchkin, N Ivanova, Y Komarov, N Maltsev, N Mikhailova, V Nenashev, R Overbeek, E Panyushkina, L Pronevitch, and E Selkov Jr. The Metabolic Pathway Collection: An Update. *Nucleic Acids Res*, 25(1):37–38, 1997.

[91] CA Shelley and ME Munk. Computer Perception of Topological Symmetry. *J Chem Inf Comput Sci*, 17:110–113, 1977.

[92] Y Shiloach. A Polynomial Solution to the Undirected Two Path Problem. *J Assoc Comp Mach*, 27(3):445–456, 1980.

[93] T Shimada, M Hagiya, M Arita, S Nishizaki, and CL Tan. Knowledge Based Simulation of Regulatory Action Lambda Phage. *Intern J of Artificial Intelligence Tools*, 4(4):511–524, 1995.

[94] RE Stobauch. Chemical Substructure Searching. *J Chem Inf Comput Sci*, 25:271–275, 1985.

[95] G Sundaram and SS Skiena. Recognizing Small Subgraphs. *Networks*, 25:183–191, 1995.

[96] AA Syslo. An Efficient Cycle Vector Space Algorithm for Listing All Cycles of a Planar Graph. *SIAM J Comput*, 10:797–808, 1981.

[97] RL Tatusov, AR Mushegian, P Bork, NP Brown, WS Hayes, M Borodovsky, KE Rudd, and EV Koonin. Metabolism and Evolution of *Haemophilus influenzae* Deduced from a Whole-genome Comparison with *Escherichia coli*. *Curr Bio*, 6(3):279–291, 1996.

[98] M Uchino. Algorithms for Unique and Unambiguous Coding and Symmetry Perception of Molecular Structure Diagram. I. Vector Functions for Automorphism Partitioning. *J Chem Inf Comput Sci*, 20:116–120, 1980.

[99] JR Ullmann. An Algorithm for Subgraph Isomorphism. *J Assoc Comput Mach*, 23(1):31–42, 1976.

[100] P Vismara. Union of All the Minimum Cycle Bases of a Graph. *Electr J of Combinat*, 4, 1997.

[101] T Wang and J Zhou. EMCSS: A New Method for Maximal Common Substructure Search. *J Chem Inf Comput Sci*, 37:828–834, 1997.

[102] JD Watson. *The Double Helix*. Penguin, 1968.

[103] JD Watson and FHC Crick. Molecular Structure of Nucleic Acids. *Nature*, 171:737–738, 1953.

[104] D Weininger. SMILES 1. Introduction to Methodology and Encoding Rules. *J Chem Inf Comput Sci*, 28:31–36, 1988.

[105] D Weininger, A Weininger, and JL Weininger. SMILES 2. Algorithm for Generation of Unique SMILES Notation. *J Chem Inf Comput Sci*, 29:97–101, 1989.

[106] WT Wipke and TM Dyott. Stereochemically Unique Naming Algorithm. *J Amer Chem Soc*, 96:4834–4842, 1974.

[107] J Xu. GMA: A Generic Match Algorithm for Structural Homomorphism, Isomorphism, and Maximal Common Substructure Match and its Applications. *J Chem Inf Comput Sci*, 36:25–34, 1996.

Appendix

This chapter shows an example session of AMR system. Its commands are Lisp-like.

Input Data

Input data are the definition of compound structures and enzymatic functions in glycolysis. The total number of actually defined compounds is over 600, but only the related compounds are shown here.

```
define "CO2"            $O=C=O$;
define "H2O"            $O$;
define "H"             $[H+]$;
define "Pi"            $O=P(O)(O)O$;

define "NAD+"
        $NC(=O)C(=C4)C=CC=[N+]4[C@H](O5)[C@H](O)[C@H](O)[C@H]5COP
(=O)(O)OP(=O)(O)OC[C@@H](O1)[C@@H](O)[C@@H](O)[C@@H]1N2C=NC3=C2N=CN=C3N$;

define "NADH"
        $NC(=O)C(CC=C4)=CN4[C@H](O5)[C@H](O)[C@H](O)[C@H]5COP(=O)
(O)OP(=O)(O)OC[C@@H](O1)[C@@H](O)[C@@H](O)[C@@H]1N2C=NC3=C2N=CN=C3N$;

define "NADP+"
        $NC(=O)C(=C4)C=CC=[N+]4[C@H](O5)[C@H](O)[C@H](O)[C@H]5COP
(=O)(O)OP(=O)(O)OC[C@@H](O1)[C@@H](O)[C@@H](OP(=O)(O)O)[C@@H]1N2C=NC3=C2N=CN=C3N$;

define "NADPH"
        $NC(=O)C(CC=C4)=CN4[C@H](O5)[C@H](O)[C@H](O)[C@H]5COP(=O)
(O)OP(=O)(O)OC[C@@H](O1)[C@@H](O)[C@@H](OP(=O)(O)O)[C@@H]1N2C=NC3=C2N=CN=C3N$;

define "CoA"
        $SCCNC(=O)CCNC(=O)C(O)C(C)(C)COP(=O)(O)OP(=O)(O)OC[C@@H]
(O1)[C@@H](OP(=O)(O)O)[C@@H](O)[C@@H]1N2C=NC3=C2N=CN=C3N$;

define :CYTOSOL;
define :MITOCHON;

in :CYTOSOL {

// ACETATE
    define "ethanol"
        $CCO$;

    define "acetaldehyde"
        $CC=O$;

    define "acetate"
        $CC(=O)O$;
    define "acetyl p"
        $CC(=O)OP(=O)(O)O$;

    define "oxalo acetate"
        $C(C(=O)O)C(=O)C(=O)O$;

    define "acetyl CoA"
        $CC(=O)SCCNC(=O)CCNC(=O)C(O)C(C)(C)COP(=O)(O)OP(=O)(O)OC[C@@H]
(O1)[C@@H](OP(=O)(O)O)[C@@H](O)[C@@H]1N2C=NC3=C2N=CN=C3N$;

    define "dihydroxy acetone p"
        $[1CH2](OP(=O)(O)O)[2C](=O)[3CH2]O$;

// TCA RELATED
    define "citrate"
        $O=C(O)CC(O)(C(=O)O)CC(=O)O$;

    define "cis-aconitate"
        $O=C(O)C=C(C(=O)O)CC(=O)O$;

    define "isocitrate"
        $OC(C(=O)O)[C@@H](C(=O)O)C(C(=O)O)$;

    define "citryl CoA"
        $O=C(O)C[C@@](O)(C(=O)O)CC(=O)SCCNC(=O)CCNC(=O)C(O)C(C)(C)COP(=O)
(O)OP(=O)(O)OC[C@@H](O1)[C@@H](OP(=O)(O)O)[C@@H](O)[C@@H]1N2C=NC3=C2N=CN=C3N$;

    define "L-citramalyl CoA"
        $O=C(O)[C@](O)(C)CC(=O)SCCNC(=O)CCNC(=O)C(O)C(C)(C)COP(=O)(O)OP(=O)
(O)OC[C@@H](O1)[C@@H](OP(=O)(O)O)[C@@H](O)[C@@H]1N2C=NC3=C2N=CN=C3N$;

    define "oxalo succinate"
        $O=C(C(=O)O)C(C(=O)O)CC(=O)O$;

    define "succinate"
        $O=C(O)CCC(=O)O$;

    define "succinyl CoA"
        $O=C(O)CCC(=O)SCCNC(=O)CCNC(=O)C(O)C(C)(C)COP(=O)(O)OP(=O)(O)OC
[C@@H](O1)[C@@H](OP(=O)(O)O)[C@@H](O)[C@@H]1N2C=NC3=C2N=CN=C3N$;

    define "2-oxo succinamate"
        $O=C(C(=O)O)CC(=O)N$;

    define "succinate sa"
        $C(C(=O)O)CC=O$;

// MALATE
    define "L-malate"
        $O=[4C](O)[3CH2][2C@H](O)[1C](=O)O$;

    define "L-lactate"
        $[3CH3][2C@H](O)[1C](=O)O$;

    define "fumarate"
        $O=C(O)/C=C/C(=O)O$;

// GLUCOSE, GLYCOLYSIS

    define "glucose"
        $[6CH2](O)[5C@@H](O)[4C@@H](O)[3C@H](O)[2C@@H](O)[1CH]=O$;
    define "glucose 6p"
        $[6CH2](OP(=O)(O)O)[5C@@H](O)[4C@@H](O)[3C@H](O)[2C@@H](O)[1CH]=O$;

    define "fructose"
        $[6CH2](O)[5C@@H](O)[4C@@H](O)[3C@H](O)[2C](=O)[1CH2]O$;

    define "fructose 1,6pp"
        $[6CH2](OP(=O)(O)O)[5C@@H](O)[4C@@H](O)[3C@H](O)[2C](=O)[1CH2]OP(=O)(O)O$;

    define "fructose 6p"
        $[6CH2](OP(=O)(O)O)[5C@@H](O)[4C@@H](O)[3C@H](O)[2C](=O)[1CH2]O$;

    define "glyceraldehyde 3p"
        $[3CH2](OP(=O)(O)O)[2C@@H](O)[1CH]=O$;

    define "1,3-bisphospho glycerate"
        $[3CH2](OP(=O)(O)O)[2C@@H](O)[1C](OP(=O)(O)O)=O$;

    define "3-phospho glycerate"
        $[3CH2](OP(=O)(O)O)[2C@@H](O)[1C](=O)O$;

    define "2-phospho glycerate"
        $[3CH2](O)[2C@@H](OP(=O)(O)O)[1C](=O)O$;

    define "glycerol 3p"
        $C(O)C(O)COP(=O)(O)O$;

// PENTOSE PATHWAY

    define "ribose 5p"
        $[5CH2](OP(=O)(O)O)[4C@@H](O)[3C@@H](O)[2C@@H](O)[1CH]=O$;

    define "ribulose 5p"
        $[5CH2](OP(=O)(O)O)[4C@@H](O)[3C@@H](O)[2C](=O)[1CH2]O$;

    define "sedoheptulose 7p"
        $[7CH2](OP(=O)(O)O)[6C@@H](O)[5C@@H](O)[4C@@H](O)[3C@H](O)[2C](=O)[1CH2]O$;

    define "6-phospho gluconate"
        $[6CH2](OP(=O)(O)O)[5C@@H](O)[4C@@H](O)[3C@H](O)[2C@@H](O)[1C](=O)O$;

    define "6-phospho glucono lactone"
        $[6CH2](OP(=O)(O)O)[5C@@H](O1)[4C@@H](O)[3C@H](O)[2C@@H](O)[1C]1=O$;

    define "D-erythrose 4p"
        $[4CH2](OP(=O)(O)O)[3C@@H](O)[2C@@H](O)[1CH]=O$;

    define "xylulose 5p"
        $[5CH2](OP(=O)(O)O)[4C@@H](O)[3C@H](O)[2C](=O)[1CH2]O$;

// PYRUVATE
    define "pyruvate"
        $[3CH3][2C](=O)[1C](=O)O$;

    define "phospho enol pyruvate"
        $[3CH2]=[2C](OP(=O)(O)O)[1C](=O)O$;

    define "3-phospho hydroxy pyruvate"
        $C(OP(=O)(O)O)C(=O)C(=O)O$;

    define "ADP"
    $OP(=O)(O)OP(=O)(O)OC[C@@H](O1)[C@@H](O)[C@@H](O)[C@@H]1[9N]2[4C]
```

```
([3N]=[2CH]3)=[5C]([6C](N)=[1N]3)[7N]=[8CH]2$;

    define "ATP"
    $OP(=O)(O)OP(=O)(O)OP(=O)(O)OC[C@@H](O1)[C@@H](O)[C@@H](O)[C@@H]1
[9N]2[4C]([3N]=[2CH]3)=[5C]([6C](N)=[1N]3)[7N]=[8CH]2$;

}

// DEFINITION OF ENZYMES

in ::CYTOSOL {

    // pentose phosphate pathway

    defun "trans aldorase"
           { "glyceraldehyde 3p" + "sedoheptulose 7p" =
                   "fructose 6p" + "D-erythrose 4p"; }

    defun "trans ketolase"
           { "sedoheptulose 7p" + "glyceraldehyde 3p" =
                  "ribose 5p" + "xylulose 5p"; }

    defun "glucose 6p dehydrogenase"
          { "glucose 6p" + "NADP+" =
                      "6-phospho glucono lactone" + "NADPH" + "H"; }

    defun "lactonase"
          { "6-phospho glucono lactone" + "H2O" ->
                                      "6-phospho gluconate"; }

    defun "6-phospho gluconate dehydrogenase"
          { "6-phospho gluconate" + "NADP+" ->
            "ribulose 5p" + "NADPH" + "CO2" + "H"; }

    defun "ribose 5-phosphate isomerase"
          { "ribulose 5p" = "ribose 5p"; }

    defun "ribulose phosphate 3epimerase"
          { "ribulose 5p" = "xylulose 5p"; }

    defun "transketolase"
          { "D-erythrose 4p" + "xylulose 5p" =
                   "fructose 6p" + "glyceraldehyde 3p"; }

    defun "phosphoketolase"
          { "xylulose 5p" + "Pi" =!
            "glyceraldehyde 3p" + "acetyl p" + "H2O"; }

   defun "alcohol dehydrogenase"
          { "acetaldehyde" + "NADPH" + "H" = "ethanol" + "NADP+"; }

   defun "acetaldehyde dehydrogenase"
          { "acetaldehyde" + "CoA" + "NAD+" =
               "acetyl CoA" + "NADH" + "H"; }

    //glycolysis
    defun "glucokinase"
           {  "glucose" + "ATP" = "glucose 6p" + "ADP"; }

    defun "malic enzyme"
           { "pyruvate" + "CO2" + "NADPH" + "H" <-
                          "L-malate" + "NADP+"; }
    defun "phosphoglucose isomerase"
            { "glucose 6p" = "fructose 6p"; }

    defun "phosphofructokinase-1"
            { "fructose 6p" + "ATP" -> "fructose 1,6pp" + "ADP"; }

    defun "fructose 1,6-bisphosphatase"
            { "fructose 1,6pp" + "H2O" -> "fructose 6p" + "Pi"; }

    defun "aldolase"
            { "fructose 1,6pp" = "dihydroxy acetone p" + "glyceraldehyde 3p"; }

    defun "glycerol 3p dehydrogenase"
            { "dihydroxy acetone p" + "NADH" + "H" ->
              "glycerol 3p" + "NAD+"; }

    defun "triose phosphate isomerase"
            { "dihydroxy acetone p" = "glyceraldehyde 3p"; }

    defun "glyceraldehyde 3p dehydrogenase"
            { "glyceraldehyde 3p" + "Pi" + "NAD+" =
                  "1,3-bisphospho glycerate" + "NADH" + "H"; }

    defun "phosphoglycerate kinase"
            { "1,3-bisphospho glycerate" + "ADP" =
                    "3-phospho glycerate" + "ATP"; }

    defun "phosphoglycerate mutase"
            { "3-phospho glycerate" = "2-phospho glycerate"; }

    defun "enolase"
            { "2-phospho glycerate" = "phospho enol pyruvate" + "H2O"; }

    defun "pyruvate kinase"
            { "phospho enol pyruvate" + "ADP" -> "pyruvate" + "ATP"; }

    // pyruvate malate cycle
    defun "lactate dehydrogenase"
            { "L-lactate" + "NAD+" = "pyruvate" + "NADH" + "H"; }

    defun "malate dehydrogenase"
            { "L-malate" + "NAD+" = "oxalo acetate" + "NADH" + "H"; }

    defun "phosphoenolpyruvate carboxykinase"
            { "oxalo acetate" + "GTP" =
                  "phospho enol pyruvate" + "GDP" + "CO2"; }

    defun "citrate lyase"
            { "citrate" + "CoA" + "ATP"
                -> "oxalo acetate" + "acetyl CoA" + "ADP" + "Pi"; }

    defun "dehydrogenase"
            { "3-phospho glycerate" + "NAD+"
                = "3-phospho hydroxy pyruvate" + "NADH" + "H"; }

} //  end of :CYTOSOL
```

Output Data

Output data is the information of the graph-matching procedure for each enzymatic function. Enzymes are numbered from 10001, and the smaller numbers are the IDs of compounds. The tracing result of three shortest paths from a terminal carbon in glucose to pyruvate is also shown. The first path is the normal glycolysis. In the second path, the carbon goes through fructose 6p → xylulose 5p → ribulose 5p → ribose 5p → sedoheptulose 7p → fructose 6p. Note that the first position in fructose 6p and the revisited position is different. The third path further goes through erythrose 4p to make the second revisit to fructose 6p. Thus, the same carbon in glucose can reach any of the three carbons in pyruvate through glycolysis and pentose pathway only.

```
:>load "sample";

[ID:10001        TRANS ALDORASE  ]
        83 102 ::CYTOSOL <-> 82 105 ::CYTOSOL
83 102 :82 105

[ID:10002        TRANS KETOLASE  ]
        102 83 ::CYTOSOL <-> 92 106 ::CYTOSOL
102 83 :92 106

[ID:10003        GLUCOSE 6P DEHYDROGENASE  ]
        77 15 ::CYTOSOL <-> 104 16 4 ::CYTOSOL
77 :104

[ID:10004        LACTONASE  ]
        104 2 ::CYTOSOL -> 103 ::CYTOSOL
104 :103
Warning: please check the mapping.
 FROM [ID:103   6-PHOSPHO GLUCONATE     C6H13010P]::CYTOSOL

   TO [ID:99    RIBULOSE 5P     C5H1108P]::CYTOSOL

  AND [ID:1     CO2     CO2]

[{ [1C](103) [2C](1) } { [2C@H](103) [1CH2](99) } { [3C@@H](103)
[2C](99) } { [4C@@H](103) [3C@@H](99) } { [5C@@H](103) [4C@@H](99) }
{ [6CH2](103) [5CH2](99) } { [7O](103) [6O](99) } { [8P](103) [7P](99)
} { [9O](103) [8O](99) } { [10OH](103) [9OH](99) } { [11OH](103)
[10OH](99) } { [12OH](103) [11OH](99) } { [13OH](103) [12OH](99) } {
[14OH](103) [13O](99) } { [15OH](103) [14OH](99) } { [16O](103)
[1O](1) } { [17OH](103) [3O](1) } ]Score 2080


[ID:10005        6-PHOSPHO GLUCONATE DEHYDROGENASE  ]
        103 15 ::CYTOSOL -> 99 16 1 4 ::CYTOSOL
103 :99

[ID:10006        RIBOSE 5-PHOSPHATE ISOMERASE  ]
        99 ::CYTOSOL <-> 92 ::CYTOSOL
99 :92

[ID:10007        RIBULOSE PHOSPHATE 3EPIMERASE  ]
        99 ::CYTOSOL <-> 106 ::CYTOSOL
99 :106

[ID:10008        TRANSKETOLASE  ]
        105 106 ::CYTOSOL <-> 82 83 ::CYTOSOL
105 106 :82 83
```

```
[ID:10009       PHOSPHOKETOLASE  ]
        106 5 ::CYTOSOL <-> 83 21 2 ::CYTOSOL
106 :83 21

[ID:10010       ALCOHOL DEHYDROGENASE  ]
        19 16 4 ::CYTOSOL <-> 18 15 ::CYTOSOL
19 :18

[ID:10011       ACETALDEHYDE DEHYDROGENASE  ]
        19 17 13 ::CYTOSOL <-> 30 14 4 ::CYTOSOL
19 :30
Warning: please check the mapping.
 FROM [ID:568   ATP     C10H16N5O13P3]::CYTOSOL

  AND [ID:75    GLUCOSE         C6H12O6]::CYTOSOL

   TO [ID:77    GLUCOSE 6P      C6H13O9P]::CYTOSOL

[{ [1CH](75) [1CH](77) } { [2C@H](75) [2C@H](77) } { [3C@@H](75)
[3C@@H](77) } { [4C@@H](75) [4C@@H](77) } { [5C@@H](75) [5C@@H](77) }
{ [6CH2](75) [6CH2](77) } { [7OH](75) [9O](77) } { [8OH](75)
[12OH](77) } { [9OH](75) [13OH](77) } { [10OH](75) [14OH](77) } {
[11OH](75) [15OH](77) } { [12O](75) [16O](77) } { [10OH](568)
[11OH](77) } { [11P](568) [8P](77) } { [12O](568) [7O](77) } {
[13OH](568) [10OH](77) } ]Score 2220


[ID:10012       GLUCOKINASE  ]
        75 568 ::CYTOSOL <-> 77 567 ::CYTOSOL
75 568 :77 567
Warning: please check the mapping.
 FROM [ID:54    L-MALATE        C4H6O5]::CYTOSOL

   TO [ID:319   PYRUVATE        C3H4O3]::CYTOSOL

  AND [ID:1     CO2     CO2]

[{ [1C](54) [1C](319) } { [2C@@H](54) [2C](319) } { [3CH2](54)
[3CH3](319) } { [4C](54) [2C](1) } { [5O](54) [1O](1) } { [6OH](54)
[3O](1) } { [7OH](54) [4O](319) } { [8O](54) [5O](319) } { [9OH](54)
[6OH](319) } ]Score 1040


[ID:10013       MALIC ENZYME  ]
        54 15 ::CYTOSOL -> 319 1 16 4 ::CYTOSOL
54 :319

[ID:10014       PHOSPHOGLUCOSE ISOMERASE  ]
        77 ::CYTOSOL <-> 82 ::CYTOSOL
77 :82
Warning: please check the mapping.
 FROM [ID:568   ATP     C10H16N5O13P3]::CYTOSOL

  AND [ID:82    FRUCTOSE 6P     C6H13O9P]::CYTOSOL

   TO [ID:80    FRUCTOSE 1,6PP          C6H14O12P2]::CYTOSOL

[{ [1CH2](82) [6CH2](80) } { [2C](82) [5C@@H](80) } { [3C@@H](82)
[4C@@H](80) } { [4C@@H](82) [3C@H](80) } { [5C@@H](82) [2C](80) } {
[6CH2](82) [1CH2](80) } { [7O](82) [16O](80) } { [8P](82) [17P](80) }
{ [9O](82) [18O](80) } { [10OH](82) [20OH](80) } { [11OH](82)
[19OH](80) } { [12OH](82) [15O](80) } { [13OH](82) [14OH](80) } {
```

```
[14OH](82) [13OH](80) } { [15O](82) [12OH](80) } { [16OH](82) [7O](80)
} { [10OH](568) [11OH](80) } { [11P](568) [8P](80) } { [12O](568)
[9O](80) } { [13OH](568) [10OH](80) } ]Score 2210


[ID:10015        PHOSPHOFRUCTOKINASE-1  ]
        82 568 ::CYTOSOL -> 80 567 ::CYTOSOL
82 568 :80 567

[ID:10016        FRUCTOSE 1,6-BISPHOSPHATASE  ]
        80 2 ::CYTOSOL -> 82 5 ::CYTOSOL
80 :82
Warning: please check the mapping.
 FROM [ID:80    FRUCTOSE 1,6PP          C6H14O12P2]::CYTOSOL

   TO [ID:36    DIHYDROXY ACETONE P     C3H7O6P]::CYTOSOL

  AND [ID:83    GLYCERALDEHYDE 3P       C3H7O6P]::CYTOSOL

[{ [1CH2](80) [1CH2](36) } { [2C](80) [2C](36) } { [3C@H](80)
[3CH2](36) } { [4C@@H](80) [1CH](83) } { [5C@@H](80) [2C@H](83) } {
[6CH2](80) [3CH2](83) } { [7O](80) [4O](83) } { [8P](80) [5P](83) } {
[9O](80) [6O](83) } { [10OH](80) [7OH](83) } { [11OH](80) [8OH](83) }
{ [12OH](80) [9OH](83) } { [13OH](80) [10O](83) } { [14OH](80)
[10OH](36) } { [15O](80) [9O](36) } { [16O](80) [4O](36) } { [17P](80)
[5P](36) } { [18O](80) [6O](36) } { [19OH](80) [7OH](36) } {
[20OH](80) [8OH](36) } ]Score 2310


[ID:10017        ALDOLASE  ]
        80 ::CYTOSOL <-> 36 83 ::CYTOSOL
80 :36 83

[ID:10018        GLYCEROL 3P DEHYDROGENASE  ]
        36 14 4 ::CYTOSOL -> 88 13 ::CYTOSOL
36 :88

[ID:10019        TRIOSE PHOSPHATE ISOMERASE  ]
        36 ::CYTOSOL <-> 83 ::CYTOSOL
36 :83
Warning: please check the mapping.
 FROM [ID:83    GLYCERALDEHYDE 3P       C3H7O6P]::CYTOSOL

  AND [ID:5     PI      H3O4P]

   TO [ID:84    1,3-BISPHOSPHO GLYCERATE        C3H8O10P2]::CYTOSOL

[{ [1O](5) [10O](84) } { [2P](5) [11P](84) } { [3OH](5) [14OH](84) }
{ [4OH](5) [13OH](84) } { [5OH](5) [12O](84) } { [1CH](83) [1C](84) }
{ [2C@H](83) [2C@H](84) } { [3CH2](83) [3CH2](84) } { [4O](83)
[4O](84) } { [5P](83) [5P](84) } { [6O](83) [6O](84) } { [7OH](83)
[8OH](84) } { [8OH](83) [7OH](84) } { [9OH](83) [9OH](84) } {
[10O](83) [15O](84) } ]Score 1690


[ID:10020        GLYCERALDEHYDE 3P DEHYDROGENASE  ]
        83 5 13 ::CYTOSOL <-> 84 14 4 ::CYTOSOL
83 :84

[ID:10021        PHOSPHOGLYCERATE KINASE  ]
        84 567 ::CYTOSOL <-> 86 568 ::CYTOSOL
84 567 :86 568
```

```
Warning: please check this one to one mapping.
FROM [ID:86     3-PHOSPHO GLYCERATE     C3H707P]::CYTOSOL

  TO [ID:87     2-PHOSPHO GLYCERATE     C3H707P]::CYTOSOL

[{ [1C](86) [1C](87) } { [2C@H](86) [2C@@H](87) } { [3CH2](86)
[3CH2](87) } { [40](86) [40H](87) } { [90H](86) [50](87) } { [100](86)
[100](87) } { [110H](86) [110H](87) } ]Score 840


[ID:10022       PHOSPHOGLYCERATE MUTASE  ]
        86 ::CYTOSOL <-> 87 ::CYTOSOL
86 :87

[ID:10023       ENOLASE  ]
        87 ::CYTOSOL <-> 321 2 ::CYTOSOL
87 :321

[ID:10024       PYRUVATE KINASE  ]
        321 567 ::CYTOSOL -> 319 568 ::CYTOSOL
321 567 :319 568

[ID:10025       LACTATE DEHYDROGENASE  ]
        61 13 ::CYTOSOL <-> 319 14 4 ::CYTOSOL
61 :319

[ID:10026       MALATE DEHYDROGENASE  ]
        54 13 ::CYTOSOL <-> 22 14 4 ::CYTOSOL
54 :22

[ID:10027       PHOSPHOENOLPYRUVATE CARBOXYKINASE  ]
        22 582 ::CYTOSOL <-> 321 581 1 ::CYTOSOL
22 582 :321 581
Warning: please check the mapping.
 FROM [ID:41    CITRATE         C6H807]::CYTOSOL

   TO [ID:30    ACETYL COA      C23H38N7017P3S]::CYTOSOL

  AND [ID:22    OXALO ACETATE   C4H405]::CYTOSOL

[{ [10](41) [30](22) } { [2C](41) [2C](22) } { [30H](41) [40H](22) }
{ [4CH2](41) [1CH2](22) } { [5C](41) [5C](22) } { [60H](41) [60](22) }
{ [7C](41) [7C](22) } { [80](41) [80](22) } { [90H](41) [90H](22) } {
[10CH2](41) [1CH3](30) } { [11C](41) [2C](30) } { [120](41) [30](30) }
{ [130H](41) [30H](5) } ]Score 1750


[ID:10028       CITRATE LYASE  ]
        41 568 17 ::CYTOSOL -> 22 30 567 5 ::CYTOSOL
41 568 :22 30 567

[ID:10029       DEHYDROGENASE  ]
        86 13 ::CYTOSOL <-> 322 14 4 ::CYTOSOL
86 :322
it =
[ID:10029       DEHYDROGENASE  ]
        86 13 ::CYTOSOL <-> 322 14 4 ::CYTOSOL
86 :322

it =
[ID:10029       DEHYDROGENASE  ]
        86 13 ::CYTOSOL <-> 322 14 4 ::CYTOSOL
```

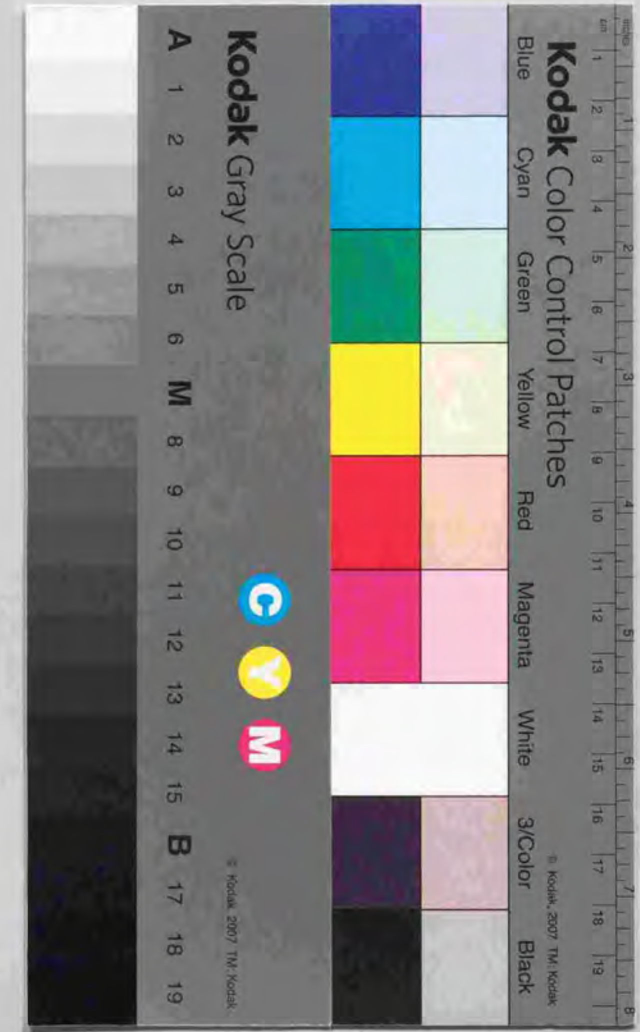
```
86 :322

it = true

:>goto ::CYTOSOL;
it = true
::CYTOSOL>trace 1 in "glucose" "pyruvate";
it = [path from 75 to 319]
::CYTOSOL>define result it;
it = [path from 75 to 319]
::CYTOSOL>head result;

it = {1 in [ID:75       GLUCOSE         C6H12O6]::CYTOSOL

[ID:10012       GLUCOKINASE  ]
        75 568 ::CYTOSOL <-> 77 567 ::CYTOSOL
75 568 :77 567
 1 in [ID:77    GLUCOSE 6P      C6H13O9P]::CYTOSOL

[ID:10014       PHOSPHOGLUCOSE ISOMERASE  ]
        77 ::CYTOSOL <-> 82 ::CYTOSOL
77 :82
 1 in [ID:82    FRUCTOSE 6P     C6H13O9P]::CYTOSOL

[ID:10015       PHOSPHOFRUCTOKINASE-1  ]
        82 568 ::CYTOSOL -> 80 567 ::CYTOSOL
82 568 :80 567
 6 in [ID:80    FRUCTOSE 1,6PP          C6H14O12P2]::CYTOSOL

[ID:10017       ALDOLASE  ]
        80 ::CYTOSOL <-> 36 83 ::CYTOSOL
80 :36 83
 3 in [ID:83    GLYCERALDEHYDE 3P       C3H7O6P]::CYTOSOL

[ID:10020       GLYCERALDEHYDE 3P DEHYDROGENASE  ]
        83 5 13 ::CYTOSOL <-> 84 14 4 ::CYTOSOL
83 :84
 3 in [ID:84    1,3-BISPHOSPHO GLYCERATE        C3H8O10P2]::CYTOSOL

[ID:10021       PHOSPHOGLYCERATE KINASE  ]
        84 567 ::CYTOSOL <-> 86 568 ::CYTOSOL
84 567 :86 568
 3 in [ID:86    3-PHOSPHO GLYCERATE     C3H7O7P]::CYTOSOL

[ID:10022       PHOSPHOGLYCERATE MUTASE  ]
        86 ::CYTOSOL <-> 87 ::CYTOSOL
86 :87
 3 in [ID:87    2-PHOSPHO GLYCERATE     C3H7O7P]::CYTOSOL

[ID:10023       ENOLASE  ]
        87 ::CYTOSOL <-> 321 2 ::CYTOSOL
87 :321
 3 in [ID:321  PHOSPHO ENOL PYRUVATE   C3H5O6P]::CYTOSOL

[ID:10024       PYRUVATE KINASE  ]
        321 567 ::CYTOSOL -> 319 568 ::CYTOSOL
321 567 :319 568
 3 in [ID:319   PYRUVATE        C3H4O3]::CYTOSOL

::CYTOSOL>head (tail result);

it = {1 in [ID:75       GLUCOSE         C6H12O6]::CYTOSOL
```

```
[ID:10012        GLUCOKINASE  ]
        75 568 ::CYTOSOL <-> 77 567 ::CYTOSOL
75 568 :77 567
 1 in [ID:77    GLUCOSE 6P      C6H13O9P]::CYTOSOL

[ID:10014        PHOSPHOGLUCOSE ISOMERASE  ]
        77 ::CYTOSOL <-> 82 ::CYTOSOL
77 :82
 1 in [ID:82    FRUCTOSE 6P     C6H13O9P]::CYTOSOL

[ID:10008        TRANSKETOLASE  ]
        105 106 ::CYTOSOL <-> 82 83 ::CYTOSOL
105 106 :82 83
 1 in [ID:106   XYLULOSE 5P     C5H11O8P]::CYTOSOL

[ID:10007        RIBULOSE PHOSPHATE 3EPIMERASE  ]
        99 ::CYTOSOL <-> 106 ::CYTOSOL
99 :106
 1 in [ID:99    RIBULOSE 5P     C5H11O8P]::CYTOSOL

[ID:10006        RIBOSE 5-PHOSPHATE ISOMERASE  ]
        99 ::CYTOSOL <-> 92 ::CYTOSOL
99 :92
 1 in [ID:92    RIBOSE 5P       C5H11O8P]::CYTOSOL

[ID:10002        TRANS KETOLASE  ]
        102 83 ::CYTOSOL <-> 92 106 ::CYTOSOL
102 83 :92 106
 3 in [ID:102   SEDOHEPTULOSE 7P        C7H15O10P]::CYTOSOL

[ID:10001        TRANS ALDORASE  ]
        83 102 ::CYTOSOL <-> 82 105 ::CYTOSOL
83 102 :82 105
 3 in [ID:82    FRUCTOSE 6P     C6H13O9P]::CYTOSOL

[ID:10015        PHOSPHOFRUCTOKINASE-1  ]
        82 568 ::CYTOSOL -> 80 567 ::CYTOSOL
82 568 :80 567
 4 in [ID:80    FRUCTOSE 1,6PP          C6H14O12P2]::CYTOSOL

[ID:10017        ALDOLASE  ]
        80 ::CYTOSOL <-> 36 83 ::CYTOSOL
80 :36 83
 1 in [ID:83    GLYCERALDEHYDE 3P       C3H7O6P]::CYTOSOL

[ID:10020        GLYCERALDEHYDE 3P DEHYDROGENASE  ]
        83 5 13 ::CYTOSOL <-> 84 14 4 ::CYTOSOL
83 :84
 1 in [ID:84    1,3-BISPHOSPHO GLYCERATE        C3H8O10P2]::CYTOSOL

[ID:10021        PHOSPHOGLYCERATE KINASE  ]
        84 567 ::CYTOSOL <-> 86 568 ::CYTOSOL
84 567 :86 568
 1 in [ID:86    3-PHOSPHO GLYCERATE     C3H7O7P]::CYTOSOL

[ID:10022        PHOSPHOGLYCERATE MUTASE  ]
        86 ::CYTOSOL <-> 87 ::CYTOSOL
86 :87
 1 in [ID:87    2-PHOSPHO GLYCERATE     C3H7O7P]::CYTOSOL

[ID:10023        ENOLASE  ]
```

```
        87 ::CYTOSOL <-> 321 2 ::CYTOSOL
87 :321
 1 in [ID:321   PHOSPHO ENOL PYRUVATE   C3H5O6P]::CYTOSOL

[ID:10024       PYRUVATE KINASE  ]
        321 567 ::CYTOSOL -> 319 568 ::CYTOSOL
321 567 :319 568
 1 in [ID:319   PYRUVATE        C3H4O3]::CYTOSOL
}

::CYTOSOL>head (tail (tail it));

it = {1 in [ID:75       GLUCOSE         C6H12O6]::CYTOSOL

[ID:10012       GLUCOKINASE  ]
        75 568 ::CYTOSOL <-> 77 567 ::CYTOSOL
75 568 :77 567
 1 in [ID:77    GLUCOSE 6P      C6H13O9P]::CYTOSOL

[ID:10014       PHOSPHOGLUCOSE ISOMERASE  ]
        77 ::CYTOSOL <-> 82 ::CYTOSOL
77 :82
 1 in [ID:82    FRUCTOSE 6P     C6H13O9P]::CYTOSOL

[ID:10008       TRANSKETOLASE  ]
        105 106 ::CYTOSOL <-> 82 83 ::CYTOSOL
105 106 :82 83
 1 in [ID:106   XYLULOSE 5P     C5H11O8P]::CYTOSOL

[ID:10007       RIBULOSE PHOSPHATE 3EPIMERASE  ]
        99 ::CYTOSOL <-> 106 ::CYTOSOL
99 :106
 1 in [ID:99    RIBULOSE 5P     C5H11O8P]::CYTOSOL

[ID:10006       RIBOSE 5-PHOSPHATE ISOMERASE  ]
        99 ::CYTOSOL <-> 92 ::CYTOSOL
99 :92
 1 in [ID:92    RIBOSE 5P       C5H11O8P]::CYTOSOL

[ID:10002       TRANS KETOLASE  ]
        102 83 ::CYTOSOL <-> 92 106 ::CYTOSOL
102 83 :92 106
 3 in [ID:102   SEDOHEPTULOSE 7P        C7H15O10P]::CYTOSOL

[ID:10001       TRANS ALDORASE  ]
        83 102 ::CYTOSOL <-> 82 105 ::CYTOSOL
83 102 :82 105
 3 in [ID:82    FRUCTOSE 6P     C6H13O9P]::CYTOSOL

[ID:10008       TRANSKETOLASE  ]
        105 106 ::CYTOSOL <-> 82 83 ::CYTOSOL
105 106 :82 83
 1 in [ID:105   D-ERYTHROSE 4P          C4H9O7P]::CYTOSOL

[ID:10001       TRANS ALDORASE  ]
        83 102 ::CYTOSOL <-> 82 105 ::CYTOSOL
83 102 :82 105
 4 in [ID:102   SEDOHEPTULOSE 7P        C7H15O10P]::CYTOSOL

[ID:10002       TRANS KETOLASE  ]
        102 83 ::CYTOSOL <-> 92 106 ::CYTOSOL
102 83 :92 106
```

```
 2 in [ID:92    RIBOSE 5P       C5H11O8P]::CYTOSOL

[ID:10006       RIBOSE 5-PHOSPHATE ISOMERASE  ]
        99 ::CYTOSOL <-> 92 ::CYTOSOL
99 :92
 2 in [ID:99    RIBULOSE 5P     C5H11O8P]::CYTOSOL

[ID:10007       RIBULOSE PHOSPHATE 3EPIMERASE  ]
        99 ::CYTOSOL <-> 106 ::CYTOSOL
99 :106
 2 in [ID:106   XYLULOSE 5P     C5H11O8P]::CYTOSOL

[ID:10008       TRANSKETOLASE  ]
        105 106 ::CYTOSOL <-> 82 83 ::CYTOSOL
105 106 :82 83
 2 in [ID:82    FRUCTOSE 6P     C6H13O9P]::CYTOSOL

[ID:10015       PHOSPHOFRUCTOKINASE-1  ]
        82 568 ::CYTOSOL -> 80 567 ::CYTOSOL
82 568 :80 567
 5 in [ID:80    FRUCTOSE 1,6PP          C6H14O12P2]::CYTOSOL

[ID:10017       ALDOLASE  ]
        80 ::CYTOSOL <-> 36 83 ::CYTOSOL
80 :36 83
 2 in [ID:83    GLYCERALDEHYDE 3P       C3H7O6P]::CYTOSOL

[ID:10020       GLYCERALDEHYDE 3P DEHYDROGENASE  ]
        83 5 13 ::CYTOSOL <-> 84 14 4 ::CYTOSOL
83 :84
 2 in [ID:84    1,3-BISPHOSPHO GLYCERATE        C3H8O10P2]::CYTOSOL

[ID:10021       PHOSPHOGLYCERATE KINASE  ]
        84 567 ::CYTOSOL <-> 86 568 ::CYTOSOL
84 567 :86 568
 2 in [ID:86    3-PHOSPHO GLYCERATE     C3H7O7P]::CYTOSOL

[ID:10022       PHOSPHOGLYCERATE MUTASE  ]
        86 ::CYTOSOL <-> 87 ::CYTOSOL
86 :87
 2 in [ID:87    2-PHOSPHO GLYCERATE     C3H7O7P]::CYTOSOL

[ID:10023       ENOLASE  ]
        87 ::CYTOSOL <-> 321 2 ::CYTOSOL
87 :321
 2 in [ID:321   PHOSPHO ENOL PYRUVATE   C3H5O6P]::CYTOSOL

[ID:10024       PYRUVATE KINASE  ]
        321 567 ::CYTOSOL -> 319 568 ::CYTOSOL
321 567 :319 568
 2 in [ID:319   PYRUVATE        C3H4O3]::CYTOSOL
}

::CYTOSOL>
```

III


Kodak Color Control Patches
Blue
Cyan
Green
Yellow
Red
Magenta
White
3/Color
Black
© Kodak, 2007 TM: Kodak
Kodak Gray Scale
A 1 2 3 4 5 6 M 8 9 10 11 12 13 14 15 B 17 18 19
C Y M
© Kodak, 2007 TM: Kodak