In Parallel Problem Solving from Nature: PPSN IV, Berlin. H.-M. Voigt and W. Ebeling and I. Rechenberg and H.-P. Schwefel (Eds.), LCNS Vol. 1141, pp. 696--705, Springer-Verlag, Berlin 1996.

Parallel Computing with DNA: Toward the Anti-Universal Machine

Klaus-Peter Zauner and Michael Conrad

Wayne State University, Dept. of Computer Science,
Detroit MI 48202, USA, e-mail: biocomputing@cs.wayne.edu,
www: http://www.cs.wayne.edu/biolab/

Abstract. A DNA-based biomolecular string processing scheme demonstrated by Adleman has attracted wide attention. While it is not known to what degree the scheme can scale up, it nevertheless introduces a new and interesting concept which seems so far to have been overlooked. The key point is that the Adleman scheme involves building specific hardware for a single problem instance. This opens a design degree of freedom that is not limited to biomolecular architectures.

1 Introduction

DNA computing, first developed by L. Adleman [1], raises novel questions about the relationship between the structure and function of computational systems. The scheme has primarily been discussed from the point of view of its programmability [13]. The implication is that it shares the main property of a conventional digital computer, namely the property of being general purpose and in the limit being computation universal.

The universality property has in fact been demonstrated for the scheme as a whole [11]. However, caution is necessary, since this is a class property, not a property of the actual systems that can be constructed. This sharp deviation from the usual concept of universality is due to the fact that the structure of any given realization is specific for a particular problem instance. Nevertheless such extreme special purposiveness can in principle afford computational advantages for certain problem domains. We will illustrate the reasons for this by developing a close analysis of the manner in which the Adleman system works. Though the scheme does not have practical value at the present time [10], it serves as a good springboard for identifying features that should apply to other possible machines and to biological organisms.

2 Universality versus Specificity

In the absence of space and time bounds general purpose machines would be universal [7]. The universal Turing machine is the simplest and best known example. Such a machine (or formalism) can emulate any other machine. Actual machines are, of course, never universal in this arbitrary sense, since they are

always subject to memory and time limitations. We call such finite machines quasi-universal and adopt the convention that the state includes the memory. The program can then be regarded as encoded in the state.

Universality is not always an advantage. In general higher performance can be achieved with specialized hardware. A typical example is an analog computer used to solve a differential equation. The equation is fixed in the physical structure, but the parameter values and initial conditions can be varied. The hardware in this case is clearly problem specific, since the physical structure has to be changed to deal with different differential equations.

The difference between quasi-universal and problem specific machines is that the problem is represented in the state in the former case and in the physical structure in the latter. The physical structure might be viewed as a material state, but the two cases differ widely in the amount of energy required to change the state. Thus when a digital computer is reprogrammed we ordinarily view it as the same machine, whereas an analog computer (or a specially cut digital circuit) would more naturally be viewed as a different machine. As noted above, both machine types can process different instances of a problem, and in both cases this is achieved by starting from different initial states. For present purposes we will therefore distinguish the structure of a system from the state of this structure, using the latter to refer to changes in a given machine structure.

The opposite extreme of universality would be a machine that encodes a single problem instance in its structure. We will call such a machine an *instance machine*. The distinguishing feature of an instance machine is that it is not possible to process different instances of a problem by starting from different initial states. More generally, any computing device with the following two properties qualifies as an instance machine:

1. The physical structure of the machine is specific for a single problem instance (but we exclude the trivial case of a system that can only be used to answer questions about its own behavior).
2. The time evolution of the machine leads to a state or structure that can be interpreted as a solution to the problem (i.e., representation of the problem in the system structure should lead to the development of a solution).

3 Key features of DNA computing

The Adleman scheme may be the first which employs actual instance machines. In fact all realizations of it are instance machines.

Each realization is a probabilistic parallel machine with a high degree of fine-grained parallelism. The new feature is that the self-assembly properties of DNA are used to achieve this high degree of parallelism. Problems are represented with DNA sequences. The solution of the problem proceeds through the free energy minimization associated with the self-assembly (or annealing) of these DNA molecules. That the physical structure of the machine is specific for a single problem is due to the fact that special DNA sequences are necessary for each problem instance.

The actual computation proceeds in three phases (Fig. 1). First, in a synthesis phase, a structural representation of the input (i.e., the problem instance) is created. In the second phase the structures formed during the synthesis stage undergo a state to state development driven by free energy minimization. In the third stage the result of the computation is extracted from an analysis of the final structure.

Let us concretize this picture by tracing an example through the three stages. Given a directed graph and a choice of a start and an end vertex in the graph,

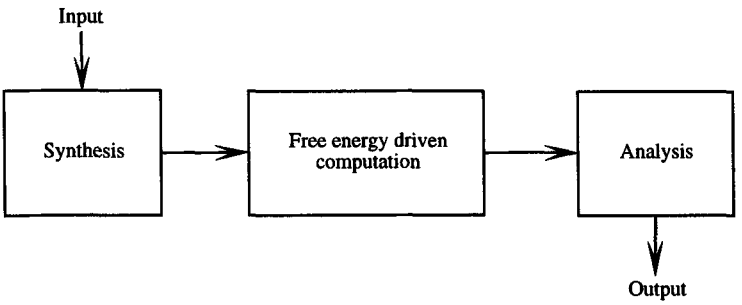

Fig. 1. The three phases of the DNA based computation.

the question can be asked: does there exist a path from the start vertex to the end vertex that visits every vertex in the graph exactly once? Such a path, called a (directed) Hamiltonian path, is frequently of interest in optimization problems. For the graph in Fig. 2 and the choice of A and D as the start and end vertices a Hamiltonian path exists and is indicated in the figure by solid arrow heads.

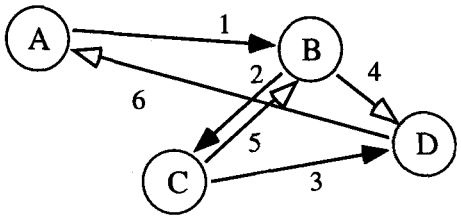

Fig. 2. Hamiltonian path problem.

The first phase of the computation is the encoding of the graph in DNA base sequences. To this end a short oligonucleotide of fixed size with an arbitrary, but unique base sequence is formally assigned to each vertex in the graph. The oligonucleotides are illustrated in Fig. 3 as dashed boxes. The first half of the sequence is indicated by the lower case letter of the vertex and the second half by the primed lower case letter.

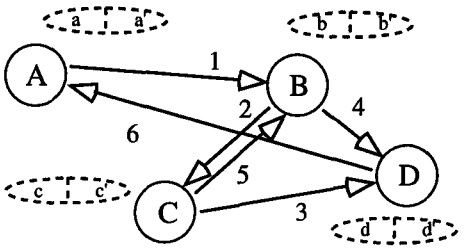

Fig. 3. Encoding of the vertices through base sequences.

From this formal assignment a molecular representation of the graph is derived as follows. For each edge in the graph a nucleotide sequence is synthesized with the property that its first half is identical with the second half of the sequence formally assigned to the vertex from which it originates. The second half of the sequence that represents the edge is identical to the first half of the sequence assigned to the vertex that the edge enters. (See edge 2 in Figs. 3 and 4 and note that the start A and the end D are exceptions.)

Then for each intermediate vertex, such as B and C in the example graph, the oligonucleotide corresponding to the complement of the sequence formally assigned to the vertex is synthesized. The set of molecules which represents the graph displayed in Fig. 2 is shown in Fig. 4, with complements indicated by bars over the lower case letters. The structure of the eight molecules completely specifies the graph.

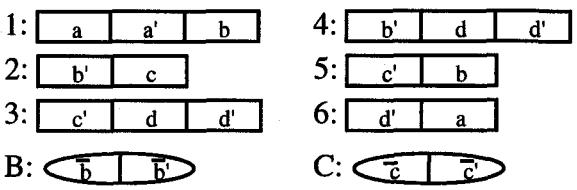

Fig. 4. Molecular representation of the directed graph shown in Fig. 2 and 3.

At this point phase 2 of the computation begins. The molecules that represent the graph interact with each other to form self-assembled supermolecular complexes through the association of complementary base sequences. Each of the complexes is a structural representation of a path in the graph. Three examples from the set of possible complexes are shown in Fig. 5. If a Hamiltonian path exists in the graph encoded in the reactant molecules then with high probability a self-assembled complex will be formed that represents this path.

Phase 3 involves extracting the solution. The self-assembled (or hybridized) complexes are held together by hydrogen bonds. Sequences that are lined up next to each other in these complexes are first enzymatically linked to form

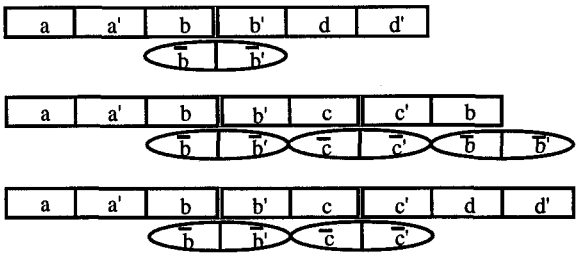

Fig. 5. Examples of complexes produced in the self-assembly phase.

continuous strands. The strands are then analyzed to extract the molecules that encode the Hamiltonian path, if such molecules exist. (More than one molecule may be possible, first because the problem may have more than one solution and second because a given solution is typically encoded by more than one strand.)

Molecules corresponding to a Hamiltonian path must satisfy three conditions. The first is that the molecule must start with the sequence encoding the start vertex and end with the sequence encoding the end vertex. If in a polymerase chain reaction (PCR) the primers are chosen according to the sequences that encode for the start and the end vertex, only DNA segments that encode for a path with the correct start and end vertices will undergo exponential amplification (Fig. 6). Such specific amplification is the key feature of the PCR technique.

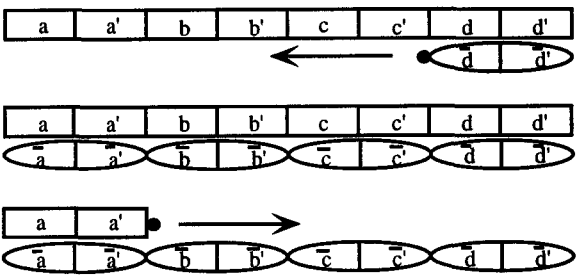

Fig. 6. Selective amplification by polymerase chain reaction (PCR).

The second necessary condition is that the molecules corresponding to problem solutions have exactly the length of the sequences assigned to the vertices multiplied by the number of vertices in the graph. Gel electrophoresis provides a method of ensuring this by separating DNA strands migrating through a polymer gel according to their length (Fig. 7).

The molecules that are known to encode the correct start and end vertices and to have the correct length are checked for whether they encode all intermediate vertices. Hybridization probes of the sequences complementary to the

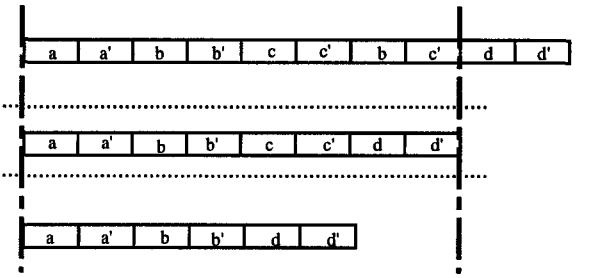

Fig. 7. Separation through gel electrophoresis.

ones formally assigned to the vertices are used to ensure this third necessary condition. Each vertex is checked separately. The probe is attached to a solid support and DNA molecules that encode the vertex in question will bind to this probe; other DNA molecules can be washed out. The process is illustrated in Fig. 8 for vertex C.

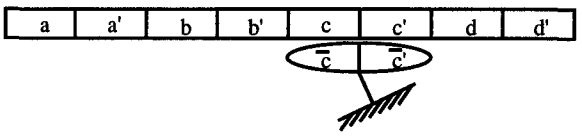

Fig. 8. Identification with hybridization probes.

Finally, the molecules that remain can be sequenced to yield the Hamiltonian path.

4 General Instance Machine Characteristics

At first sight it might seem that instance machines, such as used in the Adleman scheme, would always be less desirable than quasi-universal machines or than machines that are designed for a specific class of problems. This is not necessarily the case. Under some circumstances instance machines may be able to solve larger problems or be able to deal with an important case in a shorter amount of time.

The DNA example considered in the previous section provides a useful paradigm. We can use this example to elicit the main characteristics of instance machines, and to consider how these bear on issues of size and speed.

Table 1 summarizes the main characteristics and compares them to those of conventional (digital) computers. Recall that self-assembly (or annealing) is the key to DNA computing. This means that the state transitions are primarily

Table 1. Contrasts between universal and instance machine computing

	Conventional Computer	Instance Machine
State transition	Constraint controlled	Free energy dominated
Dissipation	Must be high enough to afford speed and reliability, low enough to sustain integrity of machine structure	Contributes to integrity of machine structure, allows for higher speed and reliability
Initial conditions	Initial state is part of problem representation, course of computation narrowly prescribed	Initial structure (or family of states) specifies problem, course of computation not sensitive to initial state
Potential size of state set	Programming requires precise control of system state, potential barriers limit number of states	Programming through control of system structure allows for closely similar states, expanded state set possible
Minimum size of state set	State set must be large enough to accommodate all classes of tasks, typically only a fraction of states used for a given problem instance	Only states relevant to problem instance need to be supported
Machine life cycle	Unlimited reuseability desired, reset mechanism must be built in	Only limited reuseability necessary, in deterministic case one run sufficient

driven by either energy minimization or entropy maximization. In a digital computer, by contrast, energy is irrelevant to the course of the computation. The designer takes great care to ensure that the different states of the machine are as similar as possible from the energy point of view and that any differences that do occur are precluded from affecting the state transitions. This feature is key to conventional programmability. The programmer therefore has the freedom to prescribe the course of the computation by setting constraints that restrict the dynamic degrees of freedom without considering energy/entropy aspects [12]. Constraints play a role in all computational systems, including DNA instance machines, but in the latter the energy differences between different states are the main controlling factor. Consequently the dynamics are self-organizing.

Of course digital computers must be plugged into a source of energy and must export heat to the environment. The energy serves to push the system over the potential barriers that separate the different possible states. For this reason dissipation is closely connected to both speed and reliability. It is possible on paper to construct universal computing models that are practically reversible [2], but these systems are nearly as likely to run backwards as forwards. Also, to ensure that the system undergoes correct state transitions it is important for states to be separated by significant potential barriers. The constraints that

encode the program followed by a digital machine limit the amount of dissipation that is possible, however. Clearly the rate of heat export must be high enough so that these constraints do not melt. This limitation is much less severe for systems with self-organizing dynamics, since the course of the computation as a whole, including structural changes, is driven by dissipation. The speed and reliability attainable is therefore potentially greater than for conventional machines.

The high parallelism of DNA computing actually has its origin in this speed and reliability property, since it depends on the high speed and reliability of DNA hybridization. The scheme also illustrates the distinction between programming by structure preparation as opposed to state preparation. The outcome of a conventional computation is highly sensitive to the initial state, since this encodes the program. The DNA computer, by contrast, encodes the problem it solves in its initial structure, and therefore in a large family of states. The course of the computation is accordingly highly stable to perturbations, since it follows a basin of attraction. Unlike a conventional constraint controlled machine, it is not necessary to support the existence of possible states that are never relevant to the problem at hand. Furthermore, the number of possible states that a conventional machine can assume is limited by the requirement for significant potential barriers.

Since an instance machine by definition deals with only one instance of a problem it is in principle unnecessary to reset it to its initial structure and run it again. Some rerunning would be useful for randomized computations and for testing the machine. The potential advantage is that it is unnecessary to restrict the design to materials with high reversibility (i.e., reuseability) and unnecessary to support reset mechanisms. The latter could be costly with free energy driven devices. In the case of the Adleman system the effort and energy required to use the same DNA bases for a different computation would be much greater than that required to start with a new batch of material. The machine is not only an instance machine, but a throw away instance machine.

5 Is there a Niche for Anti-Universality?

For an instance machine to be worthwhile the problem instance would have to be very important. The number of instances could be combinatorially explosive, but in practice it is often a particular instance that is of interest. Economic decision makers, for example, often are presented with particular instances of large graph problems. Another example would be the need to rapidly recognize a particular pattern in a complex background. In some domains it may be sufficient to have solutions for a small number of arbitrary instances. This is the case when it is required to judge the quality of a particular heuristic, approximate, or suboptimal solution procedure. The availability of a small number of optimal solutions provides a useful benchmark. Developing test sets against which to assess genetic algorithms would be an example [14].

DNA computing is far from being competitive with conventional machines in any of these domains, and may never become so. The Hamiltonian path problem

originally used to illustrate its operation is NP complete. Thus it is almost certainly the case (though not yet proved) that the number of resources required to solve the problem increases exponentially with problem size. The Adleman system cannot overcome this combinatorial explosion, just as conventional systems cannot. The advantage that it could conceivably have would be its enormous parallelism that converts the exponential time burden that conventional machines face into an exponential materials burden in terms of the amount of DNA required. Currently, however, it is not possible to exploit this tradeoff due to numerous practical limitations connected with the biochemical techniques available.

But the instance machine approach illustrated by DNA computing carries over to other technologies, where aspects of it may find more immediate application. This would be the case for conventional electronic and optical computers, where limits are set by the effect of state changes on machine structure. For example, the lifetime of a transistor is limited by electromigration (i.e., the effect of switching operations on the distribution of atoms). This puts a limit not only on transistor size, but on the materials and geometries that can be used. These limitations would be irrelevant to an instance machine since the number of switching operations could be extremely small. Similarly, many materials with highly desirable optical computing properties have been discovered but cannot be used for conventional purposes because of low reversibility. The course of the computation in these designs would not be driven by free energy minimization, but admitting changes in the machine structure opens up a new design degree of freedom.

Device proposals that utilize protein self-assembly for pattern recognition go a step further [5, 6]. Input signals are coded as molecular shapes, which then self-assemble to form higher level structures whose shape features represent different classes of input patterns. Enzymes specific for these shape features control the output of the device. The pattern recognition problem is thus converted to a free energy minimization process. Unlike Adleman DNA computing, which is also based on self-assembly, the protein self-assembly device is not programmable in a conventional sense, since a fixed set of formal (physics independent) rules is not available for ascertaining the effect of protein modifications. This is actually an advantage from the standpoint of potential computing power, since the number of interactions that can contribute to the computation is much less restricted. Such systems must be bred to perform desired functions through an evolutionary procedure [4, 3].

Biological evolutionary systems also make use of structure creating and destroying processes to maintain the potentiality of solving new problems without having to pay the price of maintaining the capacity to solve all problems. This is possible because their problem solving capabilities are represented in their structure as determined by strong chemical bonds [15, 8], as it is in DNA computing or in the protein self-assembly design. The structure-based computing principle illustrated by the instance machine concept is arguably better suited to the analysis of natural biological systems than is the state-based computing concept utilized in conventional computing models.

Acknowledgment This work was supported by the U.S. National Science Foundation under Grant No. ECS-9409780.

References

1. Adleman, L.M.: Molecular computation of solutions to combinatorial problems. Science **266** (1994) 1021–1024
2. Bennett, C.H.: Logical reversibility of computation. IBM J. Res. Dev. **17** (1973) 525–532
3. Conrad, M.: Information processing in molecular systems. Currents in Modern Biology (now BioSystems) **5** (1972) 1–14
4. Conrad, M.: The price of programmability. In: The Universal Turing Machine: A Fifty Year Survey (Herken, R. ed.) pp. 285–307. Oxford: Oxford Univ. Press (1988)
5. Conrad, M.: Molecular computing. In: Advances in Computers (Yovits, M.C., ed.), vol. 31, pp. 235–324. Boston: Academic Press (1990)
6. Conrad, M.: Molecular computing: the lock-key paradigm. Computer IEEE **25** (1992) 11-20
7. Hopcroft J.E., Ullman, I.: Introduction to Automata Theory, Languages, and Computation, pp. 146–170. Reading: Addison-Wesley (1979)
8. Kondo, H.,Yamamoto, M., Watanabe, M.: Reversible intracellular displacement of the cytoskeletal protein and organelles by ultracentrifugation of the sympathetic ganglion. J. Submicrosc. Cytol. Pathol. **24** (1992) 241–250
9. Landauer, R.: Uncertainty principle and minimal energy dissipation in the computer. Int. J. Theoret. Phys. **21** (1982) 283–297
10. Linial, M., Linial, N.: On the potential of molecular computing. Science **268** (1995) 481
11. Lipton, R.J.: DNA solution of hard computational problems. Science **268** (1995) 542–545
12. Pattee, H.H.: Physical problems of decision-making constraints. In: The Physical Principles of Neuronal and Organismic Behavior (Conrad, M., Magar, M., eds.), pp. 217–225. New York: Gordon and Breach, Science Publishers (1973)
13. Reif, J.H.: Parallel molecular computation: models and simulations. Seventh Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA95), Santa Barbara, June 1995 (to appear)
14. Schwefel, H.-P.: Evolution and Optimum Seeking, pp.105–164. New York: Wiley (1995)
15. Skoultchi, A.I., Morowitz, H.J.: Information storage and survival of biological systems at temperatures near absolute zero. Yale J. Biol. Med. **37** (1964) 158–163