Natural Computing **3:** 427–442, 2004.
© 2004 *Kluwer Academic Publishers. Printed in the Netherlands.*

Development, evaluation and benchmarking of simulation software for biomolecule-based computing

DERREL BLAIN[1], MAX GARZON[1,*], SOO-YONG SHIN[2], BYOUNG-TAK ZHANG[2], SATOSHI KASHIWAMURA[3], MASAHITO YAMAMOTO[4,5], ATSUSHI KAMEDA[4] and AZUMA OHUCHI[4,5]
[1]*Computer Science Division, The University of Memphis, Memphis, TN 38152-3240, U.S.A. (*Author for correspondence, e-mail: mgarzon@memphis.edu);* [2]*Biointelligence Lab, School of Computer Science and Engineering, Seoul National University, Seoul 151-742, Korea;* [3]*Graduate School of Engineering, Hokkaido University, North 13, West 8, Kita-ku, Sapporo 060-8628, Japan;* [4]*CREST, Japan Science and Technology Corporation (JST);* [5]*Graduate School of Information Science and Technology, Hokkaido University, North 14, West 9, Kita-ku, Sapporo 060-0814, Japan*

Abstract. Simulators for biomolecular computing, (both *in vitro* and *in silico*), have come to play an important role in experimentation, analysis, and evaluation of the efficiency and scalability of DNA and biomolecule based computing. Simulation *in silico* of DNA computing is useful to support DNA-computing algorithm design and to reduce the cost and effort of lab experiments. Although many simulations have now been developed, there exists no standard for simulation software in this area. Reliability, performance benchmarks, user interfaces, and accessibility are arguably the most important criteria for development and wide spread use of simulation software for BMC. The requirements and evaluation of such software packages for DNA computing software are discussed, particularly questions about software development, appropriate user environments, standardization of benchmark data sets, and centrally available common repositories for software and/or data.

Key words: access and common repository, benchmarking, DNA-based computing, evaluation, simulation, software architecture, user interfaces

1. Introduction

In 1994, Adleman suggested a methodology to employ DNA molecules for computation. Subsequent developments have created the new field of biomolecular computing (Adleman, 1994). Potential applications of the field capitalize on the many advantages of biomolecules, such as compactness and autonomous action. Applications include the solution of conventional algorithmic problems, building infrastructures for nano-technology, molecular memories, and gene analysis, among others. To bring these applications into being, however, requires resolving a number of basic issues about biomolecules and their reactions. These include the inherent uncertainty in chemical reactions, particularly with hybridization processes, the relatively high costs of biotechnical experimentation and research, and the gap between computer science and biochemical research practices. These conditions have important consequences in the use of biomolecules for computing purposes, such as potentially unacceptable levels of error (i.e., false positives and false negatives), efficiency (molecules being wasted in unintended reactions), and scalability (can computing with biomolecules resolve problems truly inaccessible to ordinary computers?) when compared to their conventional counterparts (Garzon et al., 2003). To overcome these problems, it is necessary to understand the behavior of biomolecules in chemical reactions in greater detail than commonly required for traditional biological applications.

This is one of two papers that discuss aspects and issues of a methodology to address these problems, namely using conventional computers to simulate the basic processes that confer on molecules their computational capabilities. Although the proposal has been made to use this methodology as an alternative to DNA-computing itself (Garzon, 2003), this discussion concerns a very different goal, i.e. what are the problems in developing, implementing, and using a reliable, efficient, and user-friendly tool for addressing biomolecular computing experiments *in silico*. There are two major aspects to this solution, namely first finding the appropriate models of the phenomena at play, and second developing, evaluating, and using the model as software. The companion paper (Rose et al., 2004) discusses issues concerning choosing appropriate models as tools for understanding the computational capabilities of molecules. In this paper, we follow up with a discussion of the issues concerning software developed for this purpose. In Section 2, we focus on the desirable properties of such software, including choice of programming languages and computing platform, developed to assist in optimizing biomolecule-based protocols for computing *in vitro*. In Section 3, we discuss aspects of another important problem: proper validation and benchmarking of the simulations' performance. In Section 4 we discuss issues concerning user interfaces in software developed for this purpose. The discussion is kept fairly independent between these in an attempt to make it applicable to other models and software with similar aims and characteristics. Finally, we conclude in Section 5 with some unresolved issues worthy of consideration by someone who is to embark on similar activity in the field. The purpose is to make these experiences useful to someone considering developing a tool for a similar problems in this field, or even analogous problems in another.

Before we proceed, it is probably necessary to articulate some arguments as to the value of such software simulations. Some of these arguments have been discussed in some detail in (Garzon et al., 2003), but we review them here briefly and add a few more for completeness. The major advantages, as mentioned above, are connected with the standard advantages in the use of models and simulations in scientific endeavors (Banks et al., 2004, Chapters 1 and 4). Science, in general, can be regarded as the construction of simplified models with enough shreds of reality to permit refined and improved understanding of phenomena. These models usually result in better understanding, better predictive power, and perhaps more efficient devices for equivalent purposes. Of course, they are also based on a number of assumptions that usually break down outside some restricted realm, limiting their use and effectiveness. However, a simulator may provide some important hints to understanding. In particular for biomolecular computing it may help with understanding the issues of reliability, efficiency and scalability (Garzon et al., 2003). A simulator can save a lot of time, labor and money because biochemical experiments cost too much or may consume too much of a technician's time and resources. More importantly, they may deliver results that cannot be obtained at all through other means, such as closed form solutions or actual biochemical experiments. (Banks et al., 2004, Sec 1.1–1.2). A more complete and detailed discussion of these general issues for BMC can be found in Garzon et al. (2003).

Our purpose here is not to survey all extant software developed for BMC, but rather to carry a systematic analysis of guidelines and criteria for design, development, evaluation and use of simulation products in general. Various types of simulators have been developed recently, each designed for a specific purpose and with particular considerations in mind. For example, motivated by the so-called word design problem, originally (Nishikawa et al., 1999) originally describes a hybrid simulator based on differential equations governing first-order chemical reactions and combinatorial models to select possible reactions. Likewise, (Shin et al., 2003, 2004) have developed Nucleic Acid Computing Simulation Toolkit (*NACST*), a system based on multi-objective optimization to simulate reactions and optimize encoding for DNA computing. (Garzon et al., 1999, 2003, 2004) have developed a more general-purpose simulator based on discrete events and local interactions between molecules. Abstract data structures representing single, double, and multiple stranded DNA molecules, which are more complexly intertwined, have been developed. These abstract strands move about in a virtual test tube by random diffusion. When they encounter each other, local interactions are triggered that result in the formation of pre-defined products. As these interactions compound over space and time, the likely outcomes of the reactions emerge. A full description of this system can be found in (Garzon et al., 2004). A good summary of other simulations for BMC for the DNA word design problem can be found in (Hartemink and Gifford, 1998; Garzon and Oehmen, 2001; Feldkamp et al., 2003).

In this paper, we use these three systems to illustrate the various issues under consideration. We will follow a question-based approach that focuses attention on what we consider to be the several critical issues a practitioner may face when designing or evaluating simulations tools of this sort.

2. Software development issues

To simulate or not to simulate?
Issues in software design cannot properly be addressed separately from the requirements of the application or model for which the software is being created (addressed in the modeling paper.) Assuming that the decision is "yes, a simulation is desirable" and that appropriate modeling has been done, the next step is to design the software to implement the chosen model. We discuss in this Section the major issues we faced when designing, developing, and using *Edna*, a comprehensive package for simulation of reactions of DNA strands, DNA complexes, RNA, enzymes, a wide variety of other biomolecules, and variants thereof (Garzon et al., 2003, 2004).

We consider that the ideal simulator is actually an assemblage of several smaller submodules (sub-simulators), where each submodule expertly emulates a different basic reaction (i.e., hybridization, ligation, etc.) with high accuracy. This idea is well implemented in the simulation by Nishikawa (Nishikawa et al., 1999). Increasing the accuracy of each of the modules increases the *reliability* of the whole simulator. In addition, each module can be built independently, which is appropriate since it is clear that good simulators may be very complex. The labor required for construction of such an assemblage clearly cannot be performed by isolated, or even a few, individuals.

2.1. *Special versus general-purpose*

What are the appropriate choices between special versus general-purpose? Useful simulation software for biomolecular computing requires a careful balance between two extremes. On the one hand, a simulator with too specific a purpose, such as (Feldkamp et al., 2003), requires a large effort for each new problem as it arises. The simulation code would need to be adapted for each situation, preventing it from reaching a level of validity and maturity that comes with long term use. The result is too little payoff. On the other hand, too general-purpose a simulator would be cumbersome to use and lack the detail to simulate specific protocols for producing usefully predictive data. Ideally software should be created to facilitate many different sorts of simulations and experiments (such as DNA memories, computational problems HPP, SAT, PCR amplification). Special-purpose simulation software packages could be written for each of these. However, a general platform which provided components for all these (and other experiments not yet developed) is more desirable. Like all software, a longer track record gives greater chance for the validity of the simulation software to be tested and substantiated. Further, if the simulated molecules and reaction protocols could be presented as choices among modules to users, then considerable effort could be saved in developing new experiments. Experimenters could rely on and reuse previous work in the software.

In our thinking, an ideal simulation platform for this arena should walk a careful middle-way between the too specific and the too general. This choice will guide the design requirements for the simulation software. It must possess the following three specific characteristics:

1. *Reliable enough* to mirror actual reactions at an appropriate granularity to be believably useful;
2. *Versatile and modular enough* to simulate different sorts of molecules and environments without too many changes, if any;
3. *Fast enough* to produce meaningful results in a reasonable period of time.

Reliability requires that results from simulated experiments be reproducible, not only on the platform on which the simulation was initially developed, but also by other researchers on other, perhaps different, machines and architectures. Versatility means that the simulation can be applied to problems other than those for which it was specifically created, perhaps even applied to problems of which the original developers are unaware. The computational burden in these types of simulations is enormous. A design criterion of speed is necessary. Other design choices will be influenced by this need. We turn to these next.

2.2. *Tradeoffs*

What are the tradeoffs between payoffs versus cost, speed, and ease of programming?

Software that could emulate the biochemical environment with a high degree of fidelity (Banks et al., 2004) for visualization and reliable simulation would be ideal (Garzon et al., 2003; West et al., 2003), and it is this choice on which we base our software design recommendations. We recommend that appropriate data structures representing biomolecules be coupled with a motion model in a virtual test tube programmed to simulate various changing chemical conditions. This basic architecture has proved to provide an adequate balance between general- and specific-purpose simulations at the proper degree of realism to be practically programmed and implemented, while producing results that can be validated by comparison to the corresponding experiments in the wet tube.

Although simulations where the molecules do not move in the tube have produced useful results (Nusser and Deaton, 2003), every surrogate deterministic model of motion will cast doubt on the realism, and therefore reliability of the results of the simulation. Though the actual data structures need not be moved about in the virtual tube, some means for causing individual molecules to react with each other in an non-deterministic manner must be created. We have experimented both with applying random vectors of movement and with modeling Brownian motion using cellular automata.

Many problems in biomolecular computing are usefully modeled as interactions of objects in a three-dimensional space (Schlick, 2002), therefore parallelism in the simulation can be achieved by decomposing the problem spatially. Achieving this parallelism increases the complexity of the implementation, but this tradeoff is necessary. Implementing the parallelism in working software and hardware means that platforms which support this capability must be chosen. Today, for most organizations, clusters of computers using a message passing scheme for parallelism (such as MPI) is the best choice. MPI has been a proven solution for many sorts of parallel programming problems. Active development and interest in MPI continues, and it appears to be a viable parallel programming platform for the foreseeable future.

2.3. *Most appropriate languages*

What are the most appropriate languages? How important is the choice? We consider developing a user-interface as a second stage in development of a good simulation package. Therefore the choice of programming language should be essentially made independently of the visualization package itself. Visual language tools that may make interfaces easier to program thus do not necessarily have an advantage here.

Given that simulations often require hours or even days of continually running execution on multiple machines, the choice of language must be made on the criterion of speed. We propose that the simulation of biomolecules in software naturally corresponds to the object oriented programming paradigm. DNA objects fit the object paradigm rather than vectors of forces (Schlick, 2002). The choices are thus among C+ +, Python, and Java, which are widely supported with freely available compilers and development environments for major operating systems. Python, while popular for many scripting tasks, is not yet as widely used as C+ + and Java. It also lacks important meta-language programming constructs such as templates. Java is a well known language and does have good library support for the language, but in this situation it does have performance penalties not associated with C+ +. According to Fox (2002), "When Java Grande started, the performance penalty for using Java was severe (up to two orders of magnitude slower than equivalent Fortran or C+ + code). Now much progress has been made and the penalty depends on the application but is typically at most a factor of two on scientific applications." While an admirable achievement by Java, C+ + is still the choice for performance for speed.

Parallelism is also necessary for speed. Choosing the most widely supported tool for parallel implementations, MPI, limits the choice of language for programming the implementation to languages for which MPI has supporting bindings. MPI has interfaces for FORTRAN, C, C+ +, Python and others. Although it is true that both MPI, and the

C+ + bindings for it, require a relatively high degree of skill to be used properly, the computational power necessary for even small simulations is still much larger than available with single-machine solutions. Therefore, for now, we consider the use of C+ + and MPI necessities.

3. Evaluation and benchmarking

In general, validation of a simulation and its model is ''one of the most important and difficult tasks facing'' a developer (Banks et al., 2004, p. 367). After all, the real payoff of the simulation is to produce accurate predictions about the real phenomena. Otherwise, experts working with the real phenomena will be naturally skeptical of the results. Poor evaluation will probably lead to unverifiable predictions or even improper understanding. This, in turn, will eventually lead to dismissal of the simulations and the underlying model. A good evaluation will have the opposite effect, increasing the level of credibility, use, and acceptance (Banks et al., 2004). A general discussion of validation advantages, disadvantages, and methods can be found in (Banks et al., 2004, Chapter 10).

In addition to the software issues *per se*, a simulation package should therefore be properly validated, evaluated, and documented. Evaluation puts an additional set of constraints on the most effective way to proceed. These constraints cannot be met if evaluation is made *a posteriori*. It must be part of the construction process. In this regard, simulation software methodologies differ substantially from the standard software development process, where specifications are made in advance and validation reduces to verifying that the specifications are met. Here, the development cycle is more akin to that in developing human–computer interfaces (Dix et al., 1997), where evaluation affects the design process since the early stages and the design-prototype-evaluate cycle is traversed much more often. In this section, we address the question of what is the most appropriate criterion to evaluate a simulator.

3.1. *Evaluation criteria*

In general, validation can be performed on several different aspects of the simulation. First, no simulation can be successful without a valid model (Rose et al., 2004). Assuming the model is valid, validation also requires calibration and fine tuning parameters of the simulation with real-world phenomena. Second, there is validation in terms of input/

output, which usually requires a data set appropriately representative of and covering the phenomenon at hand. The data may have to be artificially generated if not available for the original phenomenon. The ultimate validation is for the simulation to produce unpredicted results, which are indistinguishable from real-world phenomena. Evaluating simulators for biomolecular computing is not easy because a good simulator should have several, sometimes conflicting, properties, as discussed above. Although every property is important, of the properties mentioned above, *reliability* appears to be the most important and appropriate evaluation criterion, although perhaps the most difficult to obtain. Speed may be settled by the progress of computer science and technology. In addition, even if versatility is insufficient, the simulator will achieve some useful outcomes within limitations. However, a simulator lacking sufficient *reliability* is useless.

If reliability is the major criterion, the methodology for evaluation must therefore be based on comparison with experimental results. However, by just performing a lot of chemical reactions blindly, one cannot expect to obtain any useful results. Simulation results must be compared to the outcomes of analogous protocols executed *in vitro*. It is necessary to specify the criteria for acceptable reliability in the comparison of the results, preferably in advance. If so, it is also desirable to specify the degree of similarity for acceptable results. And of necessity, the choice of the data set becomes important in the discussion.

3.2. *Standard data sets*

In several fields such as bioinformatics, image processing, and algorithm design, the solution to the problem of choosing data sets has been resolved by establishing a standard data format and a standard set of performance benchmarks that a simulation must pass in order to be acceptable. Specific examples are the solar flare data set,[1] machine learning data set,[2] and the Virginia set for FPGA routing.[3] Lessons learned in evolutionary computation also illustrate a possible peril in evaluation here: every simulation can be made to look good (or bad) by choosing an appropriate set for validation.

No appropriate set has been formally established for DNA-based computing. The most complete validation of a simulation software package appears to be *Edna* and it is only for two applications, the Hamiltonian Path Problem (West et al., 2003) and a new procedure for code word design, the PCR-based selection protocol of (Deaton et al., 1997). In addition to Adleman's instance, the validating set includes randomly chosen digraphs of size varying from 5 to 9 vertices and a variable density of directed edges (20, 40 and 60%). The reliability of the simulations provided by *Edna* are over 99% accurate (e.g. providing the right answer) among 480 runs of the data set. Of these, only Adleman's experiment has been performed *in vitro*. The PCR selection protocol presents a better picture since the simulations have parallelled a set of experiments *in vitro* with which they are very much in agreement (Garzon et al., 2003). These preliminary steps already warrant two basic conclusions. First, benchmark data sets should be standardized for validation. Second, acceptable performance on a standard requires some agreed upon well defined thresholds.

4. Interfaces and availability

As mentioned above, many researchers have built various software programs for their own purposes, including DNA sequence design (Deaton et al., 1997; Feldkamp et al., 2001; Kobayashi et al., 2002) and biochemical reaction simulation (Hartemink and Gifford, 1998; Rose and Deaton, 2000; Hinze et al., 2001; Garzon et al., 2003). However, most of these software applications are not easily available to researchers in the field. Also, the packages run on various platforms such as Linux, MS Windows, or World Wide Web, and they are implemented in various programming languages such as Java or C/C++. In this section, we address the availability and interface of DNA computing software. We focus on the following three questions:

1. Are graphical interfaces worth the effort?
2. What is the most appropriate platform: stand-alone or distributed architecture?
3. Should simulation packages be centrally available at a common repository at this stage of development?

4.1. *Graphical interface*

Are graphical interfaces worth the effort?
Before answering the question, some terminology is required. A graphical interface is a type of user interface using graphical widgets such as icons, pictures, and menus. Since it is important for new, external users such as biologists to easily access to the simulation package, the answer to the question will be "Yes."

Most of the aforementioned simulation programs do not support, or support only minimally at best, a graphical user interface (GUI). Since those who have developed the programs use their own simulator, the importance of GUI for their software has been more or less neglected. However, to maximize the usefulness of the packages for others, it is necessary to provide many parameters to control the simulation options. Tuning many different parameters on a command-line interface is difficult and tedious. Examining and analyzing the simulation results manually is also cumbersome and, more importantly, very inefficient because a typical simulation instance may produce many data files. Therefore, the importance of the GUI for setting up and visualizing the experiment only becomes greater.

One of the obstacles to building a nice GUI is the development cost. From the developer's point of view, GUIs can be an additional burden of time and tools. Moreover, use of a GUI sometimes entails overhead in execution time. An intermediate solution to this difficulty might be a non-graphical menu-based interface: a combination of the command-line interface and GUI. If the GUI requires heavy use of resources and special machines, the non-graphical menu-based interface may lower the overhead. From this point of view, the NACST system (Shin et al., 2003, 2004) adopts a Qt library for the GUI. The merit of using Qt library is its cross-platform support, i.e. it runs on any of the commercial MS-Windows, Linux, and UNIX machines. NACST has a GUI which can tune the parameters and show results graphically. Another design consideration is the right tradeoff in coupling the interface to the simulation engine. We would like to note here that command-line interfaces enable scripting and piping, which are useful capabilities for batch jobs. To avoid losing the desirable qualities of either interface, the modularity described above should also extend to the interface design, keeping it as separate as possible from the underlying simulation engine. In an earlier version of *Edna* one couldn't make any changes to one without having to change the other as well. The two are only loosely coupled now through use of CGI and PHP (Garzon et al., 2004). One module can even fail without bringing down the other.

4.2. *Appropriate architecture*

What is the most appropriate architecture, distributed system versus single system?

Whether a distributed system platform or single system workstation is the most appropriate depends on the facilities available to the simulation-software developers and the needs of its users.

A stand-alone system allows the users to run their programs on any single machine that has the program installed. A stand-alone system can be a good alternative in simple cases or when the users can afford a powerful machine. For example, if the simulator runs simple operations such as string matching, the stand-alone application would be a good choice. For more complicated operations such as predicting and simulating secondary structure of bio-molecules, a distributed system is a more promising choice. The distributed system approach can also have an advantage, if acting as a server, of collecting the user data and results that can be used to improve the service quality of future requests and the quality of the system itself.

The best design, we believe, is one that can run in either environment. To some extent the necessity for parallelism, discussed above, demands a distributed environment, and development for a distributed system is what we recommend. Further, we have discovered in our own work that with judicious design an application can be created which runs as both a stand-alone single thread application, when needed, and as a parallel program when the machines are available to support it.

4.3. *Common repository*

Finally, *Should simulations packages be centrally available at a common repository at this stage of development?*

Repositories are necessary for an impartial, comprehensive, community resource collection. They provide files and documents categorized and grouped according to various factors, allowing quick and easy access. Repositories serve as important information sources to incoming researchers. Therefore, the answer will be “Yes.”

There are two types of repository. On one hand, the repository can maintain a database of all the files and documents created at various stages of system development. In this case, the researchers can reuse the existing files and documents instead of re-developing them. Even the hosts can manage versioning, for example, by using available software (such as the Concurrent Versions System,CVS). On the other hand, the repository can make the completed packages centrally available. This type of repository has been broadly used in research fields, including the

Genetic Algorithm Archive[4] and Machine Learning Network Online Information Service.[5]

From the bio-molecular computation point of view, a common repository is necessary and important for the entire community as well as for simulation-package developers and users since this is an interdisciplinary research field. For example, the nearest neighbor (NN) reaction model is a widely-adopted biochemical model for DNA hybridization simulation. Biochemists can experimentally test the parameters of the NN model from the repository, and the computer scientists or bioinformaticians can use, and even improve, the parameter values. The exact form of repository is not important at the moment. As a first step, the existing simulation packages can be collected on a single site, and then the site can be extended and refined as it evolves. The Protein Data Bank[6] might provide us a reference model for such a site. Considering the growing interest in DNA computing and related DNA technologies, the usefulness of the DNA computing-related software packages appears poised to grow rapidly. Thus, collecting pre-existing software packages on a common repository seems to be a useful means to speed up the developments in the entire field.

5. Conclusions

This is one of two papers that discuss various aspects of and issues with facilitating the use of simulations on conventional computers of the basic processes that confer on molecules their computational capabilities. Assuming that the appropriate models of the phenomena at play are available, discussion is focused on developing, evaluating, and using simulation software for this purpose, including choice of programming languages, computing platform, and evaluation methodologies to assist in optimizing biomolecule-based protocols for computing *in vitro.*

In terms of software issues, three major criteria have been proposed. A simulation must be reliable enough in mirroring actual reactions at an appropriate granularity to be believable, versatile for simulating different sorts of molecules and environments without too many changes, if any, and fast enough to produce results more quickly than actual experiments *in vitro.*

In terms of evaluation, we have discussed the most appropriate criterion to evaluate simulators. It is *reliability.* Development of appropriate standardized data sets and performance benchmarks would be beneficial for further developments.

In terms of interface and availability, we have argued that the primary criterion should be wide availability and access. A graphical user interface must be loosely coupled to the simulation engine despite the additional cost entailed by the developer. The applications should also have a common repository. The research community, in particular, can benefit enormously from surveying, reusing, and updating the previous programs in the common repository, eventually developing high quality DNA computing simulators for BMC.

Acknowledgements

This paper is a joint effort by three different groups at three different places made possible by a discussion during a workshop organized at CEC03 for that purpose. Much of the work by the Memphis group has been done in close collaboration with the molecular computing consortium that includes Junghuei Chen, David Wood (The U. Delaware), and Jin Wu (The U. of Arkansas). Their support from the National Science Foundation grant QuBiC/EIA-0130385 is gratefully acknowledged. The Seoul group was supported in part by the Ministry of Education & Human Resources Development under the BK21-IT Program, the Ministry of Commerce, Industry and Energy through Molecular Evolutionary Computing project, and the NRL Program from Korean Ministry of Science & Technology. The ICT at Seoul National University provided research facilities for this study. The Hokkaido acknowledges support from CREST at JST and DRB.

Notes

[1] http://www.solarflarerecords.com/, http://hesperia.gsfc.nasa.gov/sftheory/
[2] http://www.ics.uci.edu/~mlearn/MLSummary.html
[3] http://www.cs.virginia.edu/~robins/FPGA/Fpgabenchmarks. html
[4] http://www.aic.nrl.navy.mil/galist/src/
[5] http://www.mlnet.org/cgi-bin/mlnetois.pl/?File = software.html
[6] http://www.rcsb.org/pdb/

References

Adleman LM (1994) Molecular computation of solutions to combinatorial problems. Science 266: 1021–1024

Arita M and Kobayashi S (2002) DNA sequence design using templates. New Generation Computing 20: 263–277

Banks J, Carson JS, Nelson BL and Nicol DM (2004) Discrete-Event System Simulation. p 131. Prentice-Hall, New Jersey

Deaton R, Murphy RC, Rose JA, Garzon M, Franceschetti DR and Stevens Jr. SE, (1997) A DNA based implementation of an evolutionary search for good encodings for DNA computation. In: Proceedings of IEEE International Conference on Evolutionary Computation, pp 267–271. Indiana University

Dix A, Finlay J, Abowd G and Beale R (1997) Human–Computer Interaction, 3rd edn., Pearson-Prentice Hall Ltd

Feldkamp U, Rauhe H and Banzhaf W (2003) Software Tools for DNA Sequence Design

Feldkamp U, Saghafi S, Banzhaf W and Rauhe H (2001) DNA sequence generator - A program for the construction of DNA sequences. In: Proceedings of 7th DIMACS Workshop on DNA Based Computers, University of South Florida, Tampa, FL, 10–13 June 2001, pp 179–188. Springer-Verlag, Hidelberg

Fox G (2002) Grande Applications and Java, ISCOPE 2002 Conference http://grids.ucs.indiana.edu/ptliupages/publications/cisejg.pdf

Garzon M, Blain D, Bobba K, Neel A and West M (2003) Self-Assembly of DNA-like structures *in-Silico*. In: (Garzon, 2003), 185–200

Garzon M, Blain D, Neel A and Phan V (2004) Simulation environments for biomolecular computing. This issue

Garzon M (ed.) (2003) Biomolecular Machines and Artificial Evolution. Genetic Programming and Evolvable Machines, 4(2), Kluwer Academic Publishers

Garzon M (2003) Biomolecular computing *in silico*. Bulletin of the EATCS 79, pp 129–145

Garzon M and Oehmen C (2001) Biomolecular computation in virtual test tubes. In: Proceedings of 7th DIMACS Workshop on DNA Based Computers, Lecture Notes in Computer Science 2340, pp 91–100. Springer-Verlag

Garzon M, Deaton R and Rose J (1999) Soft molecular computing. In: Proceedings of DIMACS Workshop on DNA Based Computers V, pp 117–128. American Mathematical Society DIMACS Series 54

Hartemink A and Gifford D (1998) Thermodynamic simulation of deoxyoligonucletode hybridization of DNA computation. In: Proceedings of 4th DIMACS Workshop on DNA Based Computers, University of Pennsylvania, Philadelphia, PA, 15–19 June 1998, pp 25–38, AMS-DIMACS Series

Hinze T, Hatnik U and Sturm M (2001) An object oriented simulation of real occurring molecular biological processes for DNA computing and its experimental verification. In: Proceedings of 7th DIMACS Workshop on DNA Based Computers, University of South Florida, Tampa, FL,10–13 June 2001, pp 1–13. Springer-Verlag, Hidelberg

Nishikawa A, Hagiya M and Yamamura Y (1999) Virtual DNA simulator and protocol design by GA. In: Proceeding of GECCO'99 (Genetic and Evolutionary Computation Conference), 1999 pp 1810–1816

Rose J Wood D and Suyama A (2004) Modeling of oligonucleotide hybridization and error for DNA-based computers. Natural Computing, this issue, 2004

Nusser M and Deaton R (2003) DNA computing with *in vitro* selection. In: Genetic Programming and Evolvable Machines (M. Garzon, ed.) 4(2), pp 173–183. Kluwer Academic Publishers

Rose J and Deaton R (2000) The fidelity of annealing-ligation: a Theoretical analysis. In: Proceedings of The 6th International Workshop on DNA-Based Computers, Leiden University, Leiden, The Netherlands, 13–17 June 2000, pp 231–246. Springer-Verlag, Hidelberg

Shin S, Lee I, Kim D and Zhang B (2003) Multi-Objective Evolutionary Optimization of DNA Sequences for Reliable DNA Computing. Technical Report, Biointelligence lab, Seoul National University, Korea, 2003

Shin S, Jang H, Tak M, and Zhang B (2004) Simulation of DNA hybridization chain reaction based on thermodynamics and artificial chemistry. Technical Report, Biointelligence lab, Seoul National University, Korea

Schlick T (2002) Molecular Modeling and Simulation. p 33 Springer-Verlag, New York

West M, Garzon M and Blain D (2003) DNA-like genomes for evolution *in silico*. In: Proceedings of GECCO-2003, The Genetic and Evolutionary Programming Conference, Lecture Notes in Computer Science 2723, pp. 413–424. Springer-Verlag