December 2018

Volume 9 Issue 2

JoCsE

Journal Of Computational Science Education

Promoting the Use of Computational Science Through Education

ISSN 2153-4136 (online)

Volume 9, Issue 2 **December 2018**

JOCSE

Journal Of Computational Science Education

The Journal Of Computational Science Education (JOCSE), ISSN 2153-4136, is published quarterly in online form, with more frequent releases if submission quantity warrants, and is a supported publication of the Shodor Education Foundation Incorporated. Materials accepted by JOCSE will be hosted on the JOCSE website, and will be catalogued by the Computational Science Education Reference Desk (CSERD) for inclusion in the National Science Digital Library (NSDL).

Subscription: JOCSE is a freely available online peer-reviewed publication which can be accessed at `http://jocse.org`.

Contents

Introduction to Volume 9 Issue 2

Steven I. Gordon
Editor
Ohio Supercomputer Center
Columbus, OH
sgordon@osc.edu

Forward

This issue begins with an article by Taub, Armoni, and Ben-Ari that studied the impacts of programming simulations on the underlying physics knowledge of high school students. They used concept maps to compare students physics conceptual knowledge with those of experts at various stages of a course that used simulation programs to investigate physical phenomena. They found significant impacts on student?s knowledge of physics concepts.

The article by Liang describes a series of graph algorithm visualizations that were used to teach graph algorithms in an undergraduate data structures and algorithms course. They found that the visualizations helped the student understanding of the algorithms.

Chen et.al. present a student paper that used a genetic algorithm to help optimize the selection of a feature set in the study of protein folding. They conclude that their method provides an efficient and accurate way to choose features used in molecular dynamics simulations.

The student paper by Luke et. al. describes the use of an artificial neural network model to assist with DNA sequencing. The model was tested against a number of published sequences and found to be highly accurate.

Finally, Vasquez, Mohl, and Leung in their student paper developed and parallelized a preprocessing program for that generates the input files for the OncoMiner genetic sequencing tool. The programs were then used to process the 35 datasets from patients with acute myeloid leukemia.

Physics Conceptual Understanding in a Computational Science Course

Rivka Taub
Department of Education
Hevel-Yavne Regional Council
Israel
rivka.t@hevel-yavne.org.il

Michal Armoni
Science Teaching Department
Weizmann Institute of Science
234 Herzl. St. Rehovot 7610001
Israel
michal.armoni@weizmann.ac.il

Mordechai (Moti) Ben-Ari
Science Teaching Department
Weizmann Institute of Science
234 Herzl. St. Rehovot 7610001
Israel
moti.ben-ari@weizmann.ac.il

ABSTRACT

Students[1] face many difficulties dealing with physics principles and concepts during physics problem solving. For example, they lack the understanding of the components of formulas, as well as of the physical relationships between the two sides of a formula. To overcome these difficulties some educators have suggested integrating simulations design into physics learning. They claim that the programming process necessarily fosters understanding of the physics underlying the simulations. We investigated physics learning in a high-school course on computational science. The course focused on the development of computational models of physics phenomena and programming corresponding simulations. The study described in this paper deals with the development of students' conceptual physics knowledge throughout the course. Employing a qualitative approach, we used concept maps to evaluate students' physics conceptual knowledge at the beginning and the end of the model development process, and at different stages in between. We found that the students gained physics knowledge that has been reported to be difficult for high-school and even undergraduate students. We use two case studies to demonstrate our method of analysis and its outcomes. We do that by presenting a detailed analysis of two projects in which computational models and simulations of physics phenomena were developed.

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics • **Networks** → Network reliability

DOI: https://doi.org/10.22369/issn.2153-4136/9/2/1

KEYWORDS

Computational science, conceptual understanding, concept maps

ACM Reference format:

R. Taub, M. Armoni, M. Ben-Ari. 2018. Physics Conceptual Understanding in a Computational Science Course. In *Journal of Computational Science Education*, 14 pages. https://doi.org/10.1145/123 4

1 INTRODUCTION

Students face many difficulties while trying to understand physics principles, concepts and formulas [15, 19, 26]. These misconceptions exist in physics areas that are strongly related to everyday experiences such as mechanics [28], as well as in other areas that are less related to everyday experiences such as electromagnetism [17].

Widespread instructional methods aiming at overcoming these difficulties involve computer simulations—programs that model systems or processes [10]—in physics teaching. One approach for such involvement is by students' use of simulations, with or without controlling some of their variables [37]. Another approach is by programming simulations of physics phenomena [4]. Programming physics simulations has the potential to promote physics conceptual understanding in two ways. First, it enables dealing with real-life problems [35], a possible opportunity for conceptual change of misconceptions that are related to real-life experiences. Second, programming the physics phenomena may unfold students' physics knowledge, leaving no "black boxes" [4].

The research presented in this paper aims at investigating the physics learning taking place while programming physics simulations. Moreover, it investigates physics learning in a unique context, a computational-science course where the physics learning is not one of the direct goals of the course. Instead, the course's goal is to expose the students to different computational methods, while the physics content is addressed mainly through examples demonstrating how to apply these methods.

Here we report on the evolution of the students' physics conceptual knowledge taking place during the course. This knowledge was evaluated at the beginning and at the end of the process of developing computational models, and at different stages in between. The students' knowledge in each stage was compared to that of physics experts. To represent the experts' and students' knowledge we relied on the framework of concept maps [29], a powerful tool for knowledge representation, while making several modifications to this tool.

Originally, concept maps were intended to be used by students to express their own knowledge as a learning tool or as assessment tool. In Section 2, we elaborate on various ways for using concept maps as an assessment tool. In this study concept maps were used in the following manner: We asked physics experts to represent as concept maps the physics knowledge the students were supposed. We then followed the evolution of students' physics knowledge, represented as concept maps at various points during the learning process, and compared these concept maps with those of the experts.

This paper opens with a review of the relevant literature, continues with a description of the research context—the computational-science course—and the research methodology, presents the findings and summarizes them.

2 LITERATURE REVIEW

This section reviews literature on the difficulties students experience while learning physics, on the knowledge area of computational science and on concept maps as tools for assessing the evolution of knowledge.

2.1 Difficulties in Physics Understanding

Research on difficulties and alternative conceptions that students have when dealing with mechanics shows that students' intuitive knowledge differs from the formal knowledge. McDermott [28] reviewed studies that explored mechanics-related difficulties. Populations in these studies ranged over different age groups, from middle- and high-schools to universities, and included students who studied physics less than a year to those who studied for several years. Interestingly, the results obtained were very similar, pointing to the persistence of difficulties and misconceptions in mechanics. For example, Gunstone and White [18] discovered that when dealing with questions related to gravity, students tend to mix velocity and acceleration, and mass and weight. Similar results regarding the confusion between velocity and acceleration were found by Trowbridge and McDermott [46].

Bagno, Berger, and Eylon [2] found that high-school physics students provided a vague description of the components of a formula. For instance, when referring to the formula $\sum \vec{F} = m\vec{a}$, students related only to one force $\vec{F}$ and ignored the net force. As another example, the students explained the meaning of the variable t in a formula as 'time', while an accurate explanation should have been 'the time elapsed since $t = 0$,. Another difficulty described by Bagno et al. [2] is that many students were unable to explain the conditions under which a formula can be applied. For instance, in the formula $x = x_0 + at + \frac{1}{2}at^2$, 80% of the students did not mention the fact that the formula applies only for objects moving with constant acceleration. Another study reported by Shaffer and McDermott [40], examined whether 20,000 college and university students were able to associate the direction of the acceleration and the net force denoted by the formula $\sum \vec{F} = m\vec{a}$. They found that when asked about the direction and magnitude of the acceleration of a ball moving on a ramp, only 20% of the students answered correctly. The others thought that the direction of the acceleration is toward the bottom because 'gravity causes the motion'. The authors explained that some of the students did not associate the direction of the acceleration with that of the net force.

Research on students' learning geometrical optics, in particular light propagation, also uncovered difficulties. Galili and Hazan [15] reported on a conception students hold that claims that a single ray is emitted from each point of the light source. The authors explained that this conception is not incorrect but incomplete from a scientific view: the complete conception should be that multiple rays emanate from each point of the light source in all directions. Chang, Chen, Guo, Chen, Chang, Lin et al. [7] examined conceptions of elementary, middle-school and high-school students regarding different topics in classical physics. One of their findings was related to the images created by lenses and mirrors, showing that the students tended to use point-by-point conception to describe the refraction of lens and perceived light as a kind of material. When asked what would happen to the image of an object standing in front of a partially covered convex lens of a camera, most students answered that a part of the image would disappear. This is in contrast to the scientifically correct answer stating that the size of the image would stay the same, although it would look darker. Similar results were found by other researchers such as Galili [14].

Studies point to difficulties that students face regarding temperature and heat. For example, Thomaz, Malaquias, Valente, and Antunes [45] suggested five common students' misconceptions that students: (a) believe that heat is a kind of substance; (b) cannot differentiate between heat and temperature; (c) confuse temperature and the 'feel' of an object; (d) believe that application of heat to a body always results in a rise in temperature; and (e) misunderstand the temperature of a phase transition. Jasien and Oberem [22] reported on the following three difficulties physics students, and pre- and in-service teachers face: (a) the meaning of thermal equilibrium; (b) the physical basis for heat transfer and temperature change; and (c) the relationships between specific heat, heat capacity, and temperature change.

Difficulties students face when dealing with physics topics, such as mechanics, geometrical optics and heat are closely related to conceptions stemming from everyday experiences. Some topics, however, have no obvious parallel experience in everyday life. Electromagnetism is one such example [17]. While learning electromagnetism students were found to (a) be unable to link electrostatics and electrodynamics [13]; (b) be unable to connect between macro and micro relationships in electric circuits [5]; (c) confuse related concepts such as current, voltage, energy and

power [40]; (d) incorrectly determine the direction of the induced magnetic field; and (e) claim that the path of an electric charge in a magnetic field is always circular [3].

2.2 Computational Science

Computational science is a field that deals with different aspects of the construction of computational models. The Journal of Computational Science [42] describes it as an interdisciplinary field that uses advanced computing and data analysis to understand and solve complex problems. It claims that computational science has reached predictive capabilities that join the traditional experimentation and theory.

Yasar and Landau [47] explain that computational science is a field that integrates natural sciences, applied mathematics and computer science (CS), and uses the common elements of these disciplines to develop models of scientific systems; they add that computational science is not only the intersection between the three domains but also has content of its own (Fig. 1).

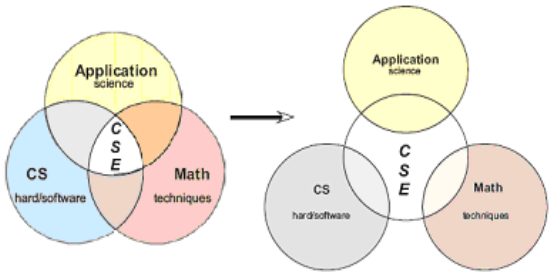


Figure 1: Left: Early view of CSE (Computational Science and Engineering) as the intersection between science, applied mathematics and computer-science. Right: Current view of CSE as sharing common concerns with these disciplines and also having content of its own [from 47].

Computational modeling is perceived to provide opportunities to promote students' conceptual knowledge. One of the most influential views regarding programming as a way to enhance scientific learning is described by Papert [31]. Learning by programming is claimed to be significantly better than learning by watching television or even reading. Programming a computer is an active learning process that empowers the learner due to the active creation of knowledge. Papert [31] explains that programming provides a tool to concretize formal and abstract knowledge. Since programming is about teaching the computer how to think, programming requires the learner to think about thinking. For example, children tend to think that in learning, they either get a right or wrong answer. But when programming a computer, solution is rarely right the first time the program is run.

Physics instructors suggest combining programming computational models as a way to improve physics learning [4, 38]. The rationale behind this suggestion is that such a combination requires that physics knowledge be organized and represented as computational models of physical systems, that is, computer programs. Abelson, Sussman, and Sussman [1] explain that computer programs are more than just sets of instructions for a computer to perform tasks. They also serve as frameworks for organizing ideas about processes. They deal with *data* that represent objects in a given system, and *procedures* that represent the rules for manipulating the data. These attributes of computer programs enable computational-science students to organize their ideas about physical objects and processes.

Research on combining computational-science elements in physics introductory courses shows positive effects. Redish and Wilson [35] developed an introductory physics course that was based on the computerized M.U.P.P.E.T environment. The authors introduced programming at the beginning of the traditional calculus-based introductory physics course at the University of Maryland. They found several benefits for teaching physics in a computer-based environment, among them are: (a) using the environment to overcome a lack of intensive mathematical knowledge; (b) exposing students to research methods that professional physicists use, and (c) being able to discuss real-world problems such as projectile motion with air resistance.

Chabay & Sherwood [4] list the pros and cons of learning physics while programming. One benefit is that when programming the physics phenomena, there are no "black boxes" of the physics knowledge at the basis of the simulation. Another benefit is the link generated between different representations of the same physics idea: an algebraic equation and programming code. Among the negative aspects of using programming for learning physics, they mention that a large portion of the students have no background in programming and therefore teaching programming takes up a lot of time needed for physics learning.

Sherin [41] compared between what he termed *algebraic physics* and *programming physics*. Two groups of his students solved physics problems. One group solved ordinary textbooks problems (algebraic physics) and the other (programming physics) was asked to develop simulations on phenomena similar to those underlying the problems solved by the algebraic physics group. He concluded that the algebraic notation of the physics formulas does not naturally displays causal relationships between variables; therefore students tend to infer the existence of equilibrium between the two sides of an equation instead of causal relationships. In contrast, programming physics leads more naturally to understanding processes and causality, stemming from the importance of the order of the lines in the program.

2.3 Concept Maps

Researchers use different methods to assess learners' conceptual knowledge, among which are open-ended and multiple-choice questionnaires. In order to use such questionnaires as research tools, they are designed by the researchers before the teaching and learning process, and they require the students to express their conceptual knowledge, as answers to the pre-defined questionnaire. In the current research, however, the situation was somewhat different. First, the physics topics that the students' projects were dealing with were not defined in advance. Instead,

the students decided on these while already working on the projects and learning the related physics material. In some cases, the students even changed the subject of the project after working on it for a few lessons. In addition, each project dealt with a different subject. Therefore, we could not prepare in advance a questionnaire for determining the students' conceptual knowledge before and after working on their projects. Second, we wished to capture and assess several stages in the development of the students' knowledge, not only pre- and post-working on the projects. It seemed that the students would find it too exhausting to answer assessment questions several times during their projects. Moreover, we could not know in advance exactly when these stages would occur. For these reasons, we looked for a method of using the students' discourse in order to assess stages in the development of their knowledge. Concept maps were our choice.

As noted by Novak and Cañas [29] concept maps were first proposed by Novak in 1972. Novak and Gowin [30] described them as spatial arrays that represent elements of knowledge as nodes together with links among them. Here we follow Ruiz-primo[36] and define a concept map as a graph consisting of nodes and labeled lines and/or arrows.[2] The nodes denote the important concepts in a domain. The lines and arrows denote relations between pairs of concepts (nodes). The labels on the lines or the arrows tell how the two concepts are related. The combination of two nodes and a labeled line or arrow is called a *proposition*. A proposition is the basic unit of meaning in a concept map.

The psychological foundations of concept maps lie in the attempts to characterize the knowledge of experts, and to assess the distance of learners' knowledge from it. Research on the cognitive aspects of science learning suggests that the knowledge of experts, apart from being more extensive than that of novices, is organized in a cognitive structure, a *schema* [8, 11, 32].

Novak and Cañas [29] explained how to construct a good concept map, emphasizing that "a concept map is never finished" (p. 12): (a) create a context by identifying a segment of a text, a laboratory or field activity, or a particular problem or question that one is trying to understand; (b) identify the key concepts that apply to this context and construct a preliminary map; (c) seek links between the concepts; and (d) revise the map, by repositioning the concepts or refining the links in ways that lead to more clarity and a better over-all structure.

Originally, concept maps were intended to be used by students to express their own knowledge as a learning tool or as assessment tool. As an assessment tool, concept maps are effective in identifying both valid and invalid ideas held by students. They can be as effective as other, more time-consuming, assessment tools for identifying the relevant knowledge a learner possesses before or after instruction [27, 29, 36]. Concepts maps are being extensively used to assess knowledge structures [21, 43]. For example, Jacobs-Lawson and Hershey [21] used concept maps to evaluate students' knowledge in psychology courses. They, too, concluded that concept maps are effective in such assessments.

There are various strategies for using concept maps for assessment which differ on several dimensions: The phase of the teaching process in which concept maps are used, the methods used for analyzing and evaluating the concept maps (direct evaluation or by comparison to a target concept map), and the manner in which concept maps are drawn (by the students, by the teachers, or by the researchers). For example, Hasemann and Mansfield [20] used concept maps drawn by 4th-grade students to assess their mathematics knowledge before the teaching process, right after it, and two years later. Ghaffar, Iqbal, and Hashmi [16] used concept maps to represent a learning objective through a concept map describing the knowledge of an expert. Novak and Gowin [30] suggested evaluating students' concept maps by comparing them to a criterion map (representing sufficient knowledge, which may be partial, compared to an expert's knowledge). McClure et al. [27] used concept maps to take a snapshot of students' knowledge and examined various assessment methods, some of which used a direct scoring method and some used master maps. Peterson and Treagust [33, 34] used concept maps in a pre-post research setting. Lomask, Baron, Greig, and Harrison [25] used concept maps that were developed by teachers from students' essays.

Our use of concept maps was a combination of several of the strategies described above. We used them as a qualitative assessment tool to analyze the development of conceptual understanding. Hence, the students' knowledge was monitored at various points during the learning process. We followed the strategy used by Lomask et al. [25] in which concept maps were developed from students' essays. Thus, our concept maps were not created by the students; rather, we created the concept maps, using them to reflect the students' knowledge. However, unlike Lomask et al. who relied on written essays, our concept maps were based on students' audio-recorded discourse. Finally, we evaluated students' knowledge as reflected in the concept maps by comparison to an expert's map. To this end, we asked physics experts to represent as concept maps the physics knowledge the students were supposed to acquire.

3 METHODOLOGY

3.1 The Research Question

How does students' conceptual physics knowledge change when developing computational models in the context of a computational-science course?

3.2 The Research Setting

The research was conducted in a 3-year computational-science course intended for talented high-school students (10th to 12th grades). This was an elective course, for which the students earned credit that was reflected in their matriculation diploma. During the course the students learned about different models such as static, mechanistic and stochastic, and used them to

[2] Although Ruiz-Primo and Araceli (2000) use only lines, we sometimes use arrows to demonstrate the direction of the connection between two nodes.

represent scientific phenomena, mainly physics phenomena. Learning about these models required the combination of physics, mathematics and computer science (CS).

Most of the learning during the course was done independently by pairs of students under the guidance of a textbook, while the teacher served as a mentor. All the classes took place once a week for 3 hours in the afternoon after regular school hours. In each of three years, the students developed (in pairs) mid and final projects of their choice, most of which dealt with physics material they had not learned before.

This research was carried out among 10th- and 11th-grade students (during the first and second year of data collection, respectively). During the course these students learned programming concepts, the Java language, kinematics, dynamics and optics. The researchers were not involved in the teaching of this course. The software the students used in these classes was Easy Java Simulations [3] (10th-11th grades) and Maxima [4] (11th grade).

Easy Java Simulation (EJS) is a software package created by Francisco Esquembre [9, 12]. It enables the construction of computational models by providing a user-friendly environment for Java. The intended users are science students, teachers and researchers who want to avoid putting too much effort into programming and more emphasis on the scientific content. To achieve that the user interface can be created without any programming knowledge on the part of the simulation's designer. Therefore she/he may focus on the algorithmic component when designing the scientific model. This software breaks the modeling process into three activities that are selected by the user: (a) documentation, (b) modeling, (c) interface design. In the modeling activity the designer represents the physical solution as an algorithm implemented in Java.

Maxima is a Computer Algebraic System (CAS), for manipulating symbolic and numerical expressions, including differentiation, integration, ordinary differential equations, systems of linear equations, polynomials, and more. In the computational-science course, 11th-grade students used Maxima for studying random models in a CAS environment, studying differential equations, finding analytic and numeric solutions of differential equations, writing a program to solve linear equations, and more.

During the research we observed and recorded the work on seven final projects. Five of them were designed and implemented by pairs of 10th-grade students and the other two by pairs of 11th-grade students. All students volunteered to participate in the research, and the research, including its methodology of data collection, was approved by the Ministry of Education. Of the volunteers we chose all the girls (three) since we wanted to have both genders represented in the research population. All together during the research we analyzed the work of 12 students, since one pair of students was observed working on two projects, in both 10th grade and 11th grade, respectively. The work on each project lasted approximately 10 hours (four lessons).

The choice of which physics phenomena to simulate was done independently by the students and it ranged over many topics. Four projects dealt with mechanics (a circular motion of a car, an anti-missile system, collision of two balls on an inclined plane, and a Frisbee game) one dealt with optics (lenses and mirrors), one with electricity and magnetism (the Lorentz force – an electric charge moving in a magnetic field), and one with thermodynamics (the diffusion equation – air heated by fire).

3.3 Research Tools

Two types of research tools were used to collect data in this study.

1. The work of the six pairs of students on the seven final projects was documented in detailed using the Debut screen-capture software. [5] It recorded their computer screens, including the work on the programming files, the mouse actions, and the students' voices while talking to each other during their work.
2. Observations of the students' work while taking field notes. One researcher (the first author) joined each lesson one to three pairs, observing their work and taking field notes. This enabled her to notice non-auditory gestures that could not be recorded and to get an impression of the students' working style, for example, how the work was divided between the two students.

As noted above, students' participation was voluntary and they were aware of the data collection process.

3.4 Analysis

Analysis of the students' discourse was conducted using concept maps [30] aimed at assessing evolution of the students' physics conceptual knowledge.

To express the students' conceptual knowledge in physics, we relied on excerpts from the students' discourse taken from the students' work on the computational models (approximately ten hours per project). Based on the excerpts we created concept maps. The students were not involved in the creation of the concept maps. For each episode in the students' discourse we drew several maps that represented the evolution in their understanding of physics concepts that were relevant to their project, and of the relationships among them. This was done by:

1. Identifying the main physics concepts discussed by the students in a specific episode.
2. Linking between the physics concepts according to the physics formulas and principles. Almost all the physics phenomena that the students modeled evolve in time, such as circular motion or a flying discus. For this reason, some of the links between concepts express time evolution. For example, the link $\vec{v} \rightarrow \vec{x}$ shows that a

[3] http://fem.um.es/Ejs/
[4] http://maxima.sourceforge.net/
[5] www.nchsoftware.com

change in velocity of an object yields a change in its position over time. Other links, however, stem from physics definitions. For example, the link $\vec{v} = \frac{d\vec{y}}{dt}$ shows that the velocity **is** the derivative of the position with respect to the time.

3. Drawing an expert's concept map expressing the concepts and links in (1) and (2).
4. Drawing concept maps that represent the students' knowledge of these concepts in several points along of the learning process: initial, final, and at least one point in between.
5. Comparing between the students' initial and final concept maps to evaluate the evolution in their conceptual knowledge (relevant to their project).
6. Comparing between the expert's concept map and the students' final concept map to evaluate the level of the students' conceptual knowledge (relevant to their project).

Actions (1)-(6) were applied twice (for validation purposes) by the third researcher, by a physics educator and by a physicist who is also a computer scientist. Disagreements were discussed and resolved, and the maps were changed accordingly.

4 FINDINGS

The choice of which physics phenomena to simulate was done independently by the students and ranged over many topics. Still, it was possible to identify some general findings that repeated themselves in several different projects. This section opens with two case studies (two projects) exemplifying in detail the analysis process and its outcomes. Then we presents the general findings of our overall analysis of all seven projects.

4.1 Case Studies

This section focuses on two projects, describing their analysis in detail and presenting its results. These case studies enable a deeper insight of the concept-map-based analysis process and its rationale. The first project was developed by two 11th-students simulating a Frisbee game. The second was developed by two 10th-grade students, simulating an electric charge moving in a magnetic field.

4.1.1 Case Study 1. Students S5 and S6 (11th grade) decided to develop a simulation of a discus thrown at a specific initial velocity and moving in the air, affected by the wind. The physics description of the motion relates to:

1. Projectile motion of the discus under the effect of the forces exerted by gravity, aerodynamic lift, aerodynamic drag (air resistance), and the wind.
2. Spinning of the discus due to an angular momentum provided by the thrower.

Developing such a simulation is challenging for high-school students since the high-school physics syllabus that they study is limited to motion primarily under the effect of a constant force such as $m\vec{g}$. Varying forces that the students encounter are the harmonic force, inverse square forces (such as the electrostatic force), and the magnetic force ($\sum \vec{F} = q\vec{v}\times\vec{B}$). The syllabus does not, however, treat motion under other varying forces that are exerted, for example, by the air on a moving object. Moreover, the syllabus does not include the physics of spinning objects. Accordingly, the teacher advised the students to first simulate the projectile motion of the discus under the effect of the constant and varying forces, and only later on to add its spinning. As it turned out, developing the simulation of projectile motion was challenging enough for them and lasted five lessons (approximately twelve hours), leaving no time for the spinning force. For this reason, the new physics material that the students had to study was dealing with the effect of the forces mentioned in item (1) above on the motion of the discus.

The conceptual knowledge required in order to develop such a simulation appears in Fig. 2 represented as a (high-level) expert's concept map.

The discus is being thrown at an initial velocity $\overrightarrow{v_0}$. Four forces affect the motion of the discus: gravitation (assumed to be constant), aerodynamic lift and drag, and the wind. The students assumed the wind to have a constant velocity. The net of these forces generate an acceleration as expressed by Newton's second law $\sum \vec{F} = m\vec{a} = m\frac{d\vec{v}}{dt}$. Since the velocity is a vector, it is represented by magnitude and angle (v, α). The velocity is defined as the derivative of $\vec{x}$, the position of the discus. For each Δt , the drag and lift forces are being changed because of change in the velocity, leading to a new net force, changing the velocity and accordingly the position of the discus.

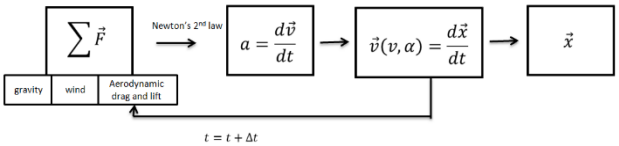


Figure 2: Expert's concept map representing the conceptual knowledge of the physics involved in the project of S5 and S6.

Here we focus on an example demonstrating the development of a subset of the physics involved, relying on data taken from several episodes scattered along the full work on this project, which lasted about 12 hours.

The example deals with the interrelationships between the net force and the velocity of the discus, repeatedly calculated every Δt. Figure 3 presents the relevant parts of the expert's concept map. This map is more detailed than the previous one. It shows, as previously, the effect of the net force on the velocity. In addition, it shows that the observed velocity of the discus ($\vec{v}$) is the sum of the discus' velocity relative to the air and the velocity of the wind (assumed by the students to be constant and in the direction of the x axis). The speed of the discus relative to the air changes the forces of aerodynamic lift and drag.

Since this example focuses on this aspect of the project, the students' evolving concept maps will be compared to the expert map in Fig. 3, and not to the wider map of Fig. 2.

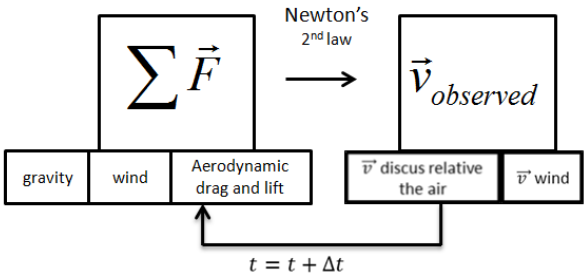


Figure 3: Expert's concept map of the interrelationships between the net force and the velocity of the discus.

The students started the project by studying the relevant physics principles and formulas from a scientific paper the teacher gave them. The relevant material that was explained in this paper includes: (a) the observed velocity of the discus is the sum of the velocity of the discus relative to the air and of the velocity of the wind; (b) the calculations of the discus' velocity and position are time dependent; and (c) the velocity of the discus relative to the air affects the aerodynamic forces of drag and lift.

In what follows we describe three learning episodes. We will see that the students did not understand these issues very well, and that their understanding evolved while developing the simulation.

Episode 1: After reading the paper given to them by the teacher, the students said:

S5: I don't understand the meaning of *Vrel* [velocity of the discus relative to the air].

S6: I want to start programming; we'll figure it out later.

S5 and S6 began with declaring the program variables corresponding to the physics variables. They started with *Vd* and *Rd,* the variables corresponding to the magnitude and angle of the velocity of the discus **relative** to the air. They continued by declaring *Vrel* and *R,* corresponding to the magnitude and angle of the **observed** velocity of the discus. The reason they chose to name the variable *Vrel* and not *Vobserved* was that they referred to this velocity as **relative to the ground**. Choosing such variable names was the first step in confusing between the two kinds of velocities of the discus.

The students wrote the programming segment corresponding to the physics formulas that appeared in the scientific paper (Fig. 4). It started with calculating *Vrel,* the observed velocity of the discus by summing the wind velocity and the discus velocity relative to the air. It continued with calculating the x and y components of the acceleration.

Two logical errors exist in this segment. First, it is written just once, thus it will be executed only once and will not cause a change in velocity over time. Second, the calculation of the contribution of the aerodynamic drag and lift forces to the acceleration (based on Newton's second law) is affected by *Vrel*, which is the sum of the velocity relative to the air *and* the wind velocity. Instead, it was supposed to be affected only by the discus' velocity relative to the air.

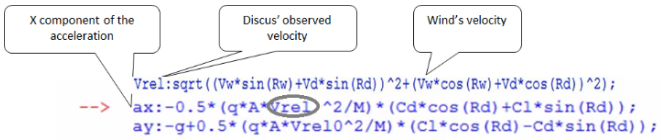


Figure 4: Code of the calculation of the discus' *observed* velocity and of the x and y components of the acceleration.

We conclude that the students correctly perceived the observed velocity of the discus as including its velocity relative to the air and the wind velocity. They incorrectly thought that the discus *observed* velocity affects the forces and thus the acceleration and they did not see the dependence of the velocity on time.

The concept map that describes the students' understanding appears at Fig. 5. It shows the students' perception of the effect of the sum of the velocities on the aerodynamic forces. However, it does not contain a time loop, indicating the students' lack of understanding of the time dependency of the process.

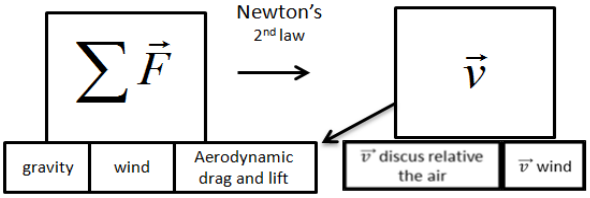


Figure 5: First concept map of the students S5 and S6.

Episode 2: Executing the simulation led S6 to understand that there are problems in the program they wrote. She re-checked it several times, reflecting on what they wanted to achieve:

S6: In order to calculate the velocity I need to first calculate the acceleration and then calculate its anti-derivative.

She then suddenly said:

S6: Wait, does that mean that I need to calculate it *every time*?

S5: Hmmm… I don't know.

S6: Do I actually need to calculate all the previous values and use them for calculating the velocity repeating it again and again?

The above excerpt indicates that S6 gained a new insight, regarding the time dependency of the calculation of the velocity (proving that she did not understand it in the previous episode). These desired repetitions are the algorithmic description of a programing loop which calculates the physical change in the acceleration and consequently in the velocity of the discus.

After several more discussions, the students programmed a for-loop (Fig. 6) representing the progress of time (t=0 to t=10). It includes a repetitive calculation of the: acceleration of the discus (as was explained for Fig. 5), the discus velocity relative to the air, and the observed discus velocity including the wind. Although here the students understood the dependency of the calculation on time, they were still in error in that they calculated the acceleration according to the discus observed velocity and not the relative velocity.

The concept map that describes the students' conceptual knowledge at this stage appears at Figure 7. Again, it shows the students' perception on the effect of the sum of the velocities on

the aerodynamic forces. It differs from the previous map, however, in that it contains a time loop, indicating an understanding of the time dependency of the calculations.

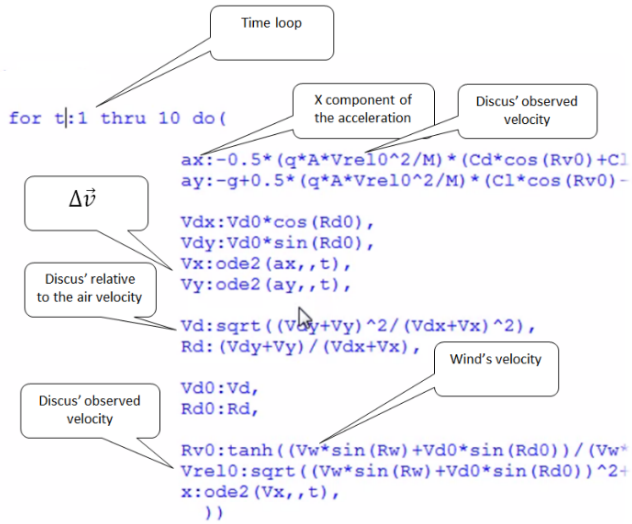


Figure 6: The for-loop written by the students.

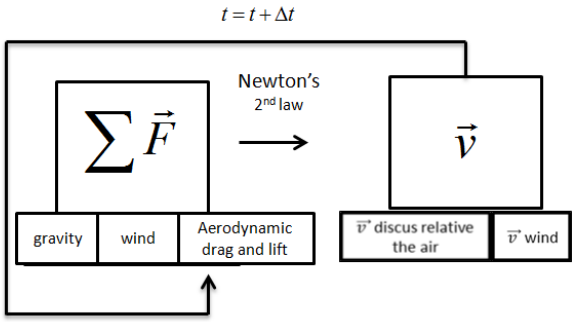


Figure 7: The second concept map of the students S5 and S6.

Episode 3: S5 and S6 faced many difficulties when trying to execute their program, debug and understand it. Significant effort was invested trying to understand the meaning of each of the velocities:

S6: There are the wind's velocity, the discus velocity, and *Vrel*.

S5: Wait, what was our meaning here [pointing on the segment of calculating the anti-derivative of the acceleration]? Which velocity is it?

S6: I can't remember.

At this point the students consulted their teacher for help in writing a correct program. Since they still did not understand the meaning of each velocity, they tried to copy the components of the formulas into the program without understanding them. This led to long sessions of correcting the code, the students trying again and again to understand the variables. Eventually, S6 said:

S6: Oh! *Vrel* is equal to *Vdiscus* **minus** *Vwind*.

In this excerpt S6 means that *Vrel* is not what they previously thought— the observed discus velocity, summing the velocity relative to the air and the wind velocity. Instead, it is the velocity relative to the air, that is, the observed velocity minus the velocity of the wind. The following method was generated (Fig. 8):

```
public void Vrel (vdx,vw) {
  V=Math.sqrt((vdx-vw)*(vdx-vw)+vdy*vdy);
  return(V);
  }
```

Figure 8: Code of calculation of relative to the air discus' velocity.

The relevant concept map describing the students' current conceptual knowledge is now equivalent to the expert's one (Fig. 9).

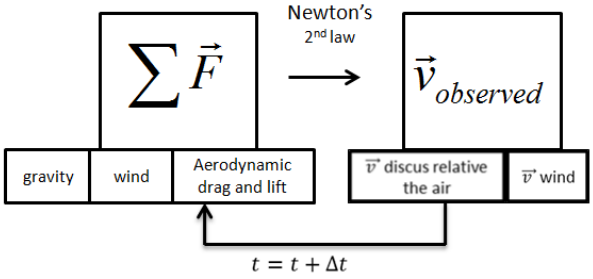


Figure 9: Third concept map of the students S5 and S6.

To summarize this case study, S5 and S6 achieved three main insights:

1. The dependence of the calculation of the discus velocity on time. The students understood the need to repeatedly calculate the velocity based on its previous values.
2. The difference between the variables of the observed velocity of the discus and one the relative to the air. After confusing the two variables for three lessons, the students finally understood which of the discus velocities is the observed and which is relative to the air.
3. The cause and effect relationships between the discus velocity relative to the air and the aerodynamic forces. After achieving the second insight of the difference between the two discus' velocities, the students correctly understood that only the discus *relative* velocity affects the aerodynamic forces and hence the acceleration.

Many factors affected the development in the students' conceptual understanding: the computational environment, the scientific paper, the teacher, the conversations between the students and more. It is not clear which one of these affected each one of the described episodes. Still, during observations, we noticed the following:

The students did not understand the physics formulas well enough before programming them; they simply copied parts of the formulas. The statement of S6 in the first lesson clearly demonstrates this point: "I want to start programming; we'll figure it [the meaning of the variables in the formula] out later." The

reason may be that the students preferred implementing the formulas as programming segments instead of properly understanding them. Or, they may have thought that programming the formulas will assist them in understanding them. Either way, it seems reasonable to conclude that the need to program prevented the student from achieving understanding, at least initially.

The students used names of programming variables that were very hard to distinguish from each other. For example, they used *Vd, Vx, Vy,* and *Vrel* to represent different kinds of velocities. Since programming the simulation was a long process that lasted around twelve hours, the students could not remember the meaning of each variable. This made the process of debugging the code quite challenging. For example, when debugging, S5 asked: "Which velocity is it?" We believe that using more meaningful names may have assisted the processes of understanding both the physics and the programming.

On the other hand, it was clear that the students have gone through deep learning processes. The relevant physics material was challenging. The need to represent it in a program forced the students to explore the meaning of each concept and the interrelationships among the physics concepts. Moreover, the fact that the simulation was time dependent encouraged the students to understand the time dependency of the process. An analysis of the learning processes that occurred during the course and the interrelations between CS and physics during these learning processes was the focus of another publication (removed for anonymity) in which we used the perspective of Knowledge Integration [23, 24].

Another interesting observation concerns the students' strong motivation to accomplish their mission and develop a correct simulation. This motivation was expressed, for example, in their use of several sources for learning the relevant physics material, among which are the scientific paper, the teacher and the internet. Most of their time was used for independent learning and very little irrelevant activities such as chatting with friends. We believe that the type of the mission the students confronted yielded both enthusiasm and obligation. However, the interrelations between students' motivation and learning are not at the focus of this paper.

4.1.2 Case Study 2. S7 and S8 (10th grade) simulated an electric charge entering a force-free region and then moving into a constant magnetic field. The students decided that the charge would start moving along a straight line in the force-free region and then circulate in the magnetic field. The physics equations the students relied on were $x = vt$ for the force-free region and $\vec{F} = q\vec{v}\times\vec{B}$ for the magnetic field. They programmed the two motions of the charge. When executing the simulation they discovered that the circular motion did not appear as they expected. They tried various ways to solve this problem, but after two more lessons (approx. 5 hours) the students reached a dead end and decided to abandon the project. Despite their lack of success, they did gain physics knowledge while working on the project.

Fig. 10 presents the expert's concept-map of the formula $\vec{F} = q\vec{v}\times\vec{B}$, underlying the mechanism of the motion of an electric charge in a magnetic field.

It contains three vectors, each with direction and magnitude: $\vec{v}$, the velocity of the electric charge, $\vec{B}$the magnetic field, and $\vec{F}$, the magnetic force acting on the electric charge. The fourth concept is q, the charge of the particle, which has magnitude and a sign (plus or minus).

The velocity $\vec{v}$ and the charge q of the electric charge entering a magnetic field, jointly with the magnetic field $\vec{B}$, set the size and direction of the force $\vec{F}$. The force, (which is the net force in this case), in turn (according to Newton's second law) changes the velocity of the charge and consequently its location. For each Δt the force repeatedly changes the velocity causing the process to repeat itself as long as the charge moves in the magnetic field.

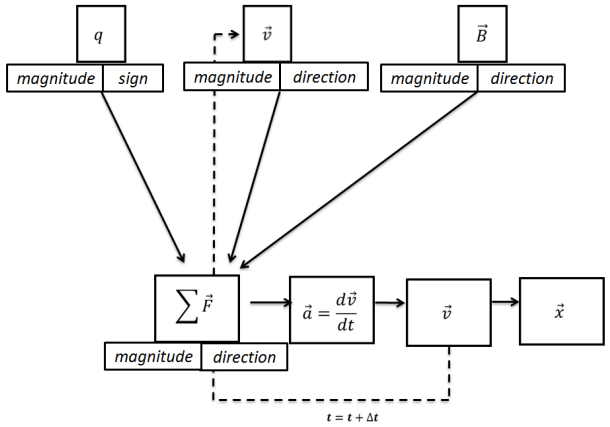


Figure 10: Expert's concept map representing the conceptual knowledge of the physics involved in the project of S7 and S8.

At the beginning of the episode taken from the second lesson of the project, the students demonstrated a vague understanding of the equation $\vec{F} = q\vec{v}\times\vec{B}$, but they did not understand the meaning of the concepts denoted by the equation nor the relationships among them. At the end of this episode they were able to explain the meaning of the concepts, and the casual relationships among them. Three concept maps representing the evolution in the students' understanding during this episode are presented.

This description starts at the stage when the students discovered that the circular phase of the motion is wrong, that is, the charge entered the magnetic field, "jumped" upwards, moved down and only then started the circle. The students debugged the simulation, but did not succeed in correcting it.

At this stage they consulted the teacher and clarified the physical meaning of the equations they wrote as programming code:

S7: What do we have here? *v*? What is *v*? [...] *B* is the magnetic field.

Teacher: What is the direction of the field?

S7: I don't care, we haven't decided yet.

According to the formula, the direction of the magnetic field affects the direction of the circular motion of the charge; therefore, in order to present this motion the direction of the field should be pre-set, though S7 did not think that this was necessary.

Fig. 11 presents the concept map representing the students' knowledge at this stage.

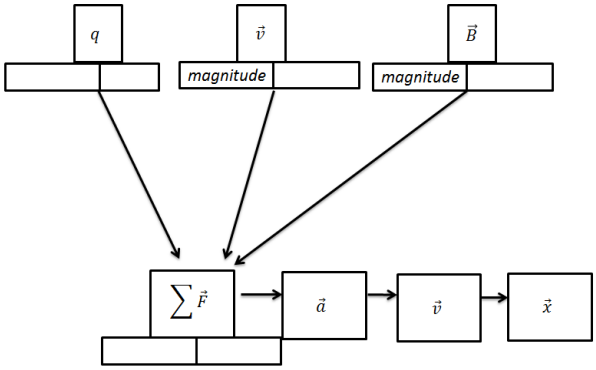


Figure 11: First concept map of the students S7 and S8.

This map shows that the students did not know the meaning of some of the concepts denoted in the equation. Moreover, although they knew that $\vec{B}$ is the magnetic field, they did not understand that its direction affects the direction of the force, thus affecting the direction of the charge's motion. Having to concretely represent the direction of the charge's motion in the visual simulation led the students to discuss the factors that influence it. They arrived at the following conclusion:

S7: The simulation is two dimensional; therefore, the charge cannot circulate inward. The magnetic field, therefore, has to be directed outward. It affects the direction of the force which in turn changes *v* every time, resulting in a circle.

The above understanding is expressed in the following concept map (Fig. 12):

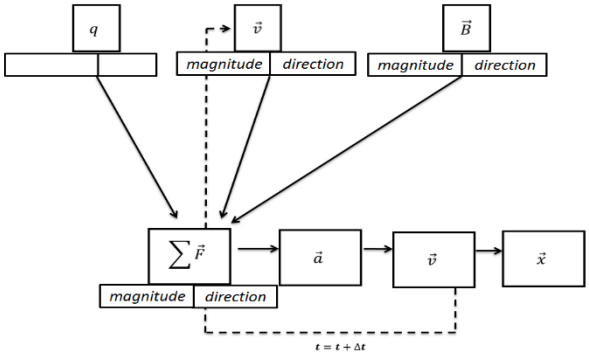


Figure 12: Second concept map of the students S7 and S8.

This map shows that the students understood the meaning of the variables and the relationships among them. Still, it is not complete, since q is missing. After discussing it some more with the teacher, S8 stated the following:

S8: The sign of the charge [the sign of q] affects the direction of the circle, as well.

The concept map representing the students' current understanding (Fig. 13) is similar to the expert's map (Fig. 10) with one exception. The expert's map includes $\vec{a} = \frac{d\vec{v}}{dt}$ but the students' map does not. This is because 10th-grade students have not yet learned derivatives.

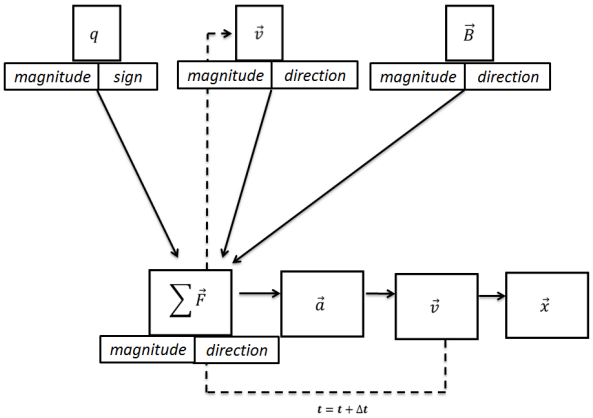


Figure 13: Third concept map of the students S7 and S8.

Although gaining a better understanding of the meaning of the equation, the students did not solve the problem of the charge "jumping" before circling in the magnetic field region. Their error was in another equation used for the circular motion inside the magnetic field: $\theta = \omega t$, where θ is the angle, ω is the angular velocity, t is the time. The variable t is assumed to be equal to zero when starting the circular motion. In their simulation, on the other hand, the time was greater than zero, since the charge was moving in the force-free region first.

The students kept making minor changes to the program and executed the simulation to check whether the problem was solved. After two more lessons they decided to abandon the subject and develop a new simulation.

4.2 General Findings

The other five projects were analyzed in a manner similar to that of the two case studies. In all projects a development of students' conceptual knowledge was evident. As the work on the projects progressed, students' concept maps improved and became more similar to the corresponding expert map. Our findings indicated that this development was fostered by the need to program a physics phenomenon and represent it as a simulation. In particular, the following patterns, which were demonstrated in the two case studies above, were also found in other projects:

Understanding the time dependency of physical processes. In both case studies, the students' initial concept map did not contain a link representing time dependency, but such a link was present in the consequent maps. This was the also case for other projects. Most of the simulations designed by the students represented physical processes that progress in time. For this reason, simulations' design demands an explicit use of the time variable and loops. The loops may be implicit in the software (as in EJS) or explicitly written by the students (as in Maxima). In both cases, recognizing the need for such loops is related to students' better

understanding of time dependency, an understanding that is known to be hard for physics students [2].

*Understanding the meaning of the components in a formula***.** In both case studies, the students' concept maps depicted a misunderstanding or a partial understanding of the meaning of a certain variable (which represents a component in the formula that corresponds to the simulation) that was later resolved. This was also evident in the other projects. All the students worked with formal physics formulas that they later translated into programming statements. One of the initial phases in program design is declaring the variables that represent the physics variables, and deciding on their type (integer or float). Even this simple action forced the students to try and understand the meaning of the programming variables and consequently the physics ones.

Understanding the cause and effect relationships in the formula. In both case studies an understanding of the cause and effect relationships in the formula developed during the work on the project. In the first case study these were cause and effect relationships between the discus velocity relative to the air and the aerodynamic forces. In the second case study these were the cause and effect relationships between the direction of the magnetic field, the direction of the force, and the direction of the charge's motion. Many times students started the projects with a vague understanding of the physics formulas, as also reported in the literature [2]. The "step by step" nature of the algorithm and the resulting program, in which a single-line physics equation needs to be implemented in several program lines, forces the students to decide what is the cause in the formula and what is the effect. This is in contrast with the mathematical notation that uses the equality sign and does not indicate the direction of the causality.

5 DISCUSSION

This research focused on gifted high-school students who participated after regular school hours in an elective 3-year computational-science course (for which they earned credit reflected in their matriculation diploma). They combined physics, mathematics, and computer science in order to learn computational models and computational methods. We investigated the relationships between the computational environment, CS and the physics conceptual knowledge that the students gained.

Programming physical phenomena is a complex activity. On the one hand, it puts an extra load on the students. It may confuse the students and prevent them from focusing on aspects of physics. Evidence for this claim was included in the descriptions of both case studies presented in the paper and was found in other cases that we analyzed. Using improper names for programming variables, lack of debugging skills, and more were found to prevent the students from achieving some physical insights.

On the other hand, programming forces the students to unfold the physical meanings and relationships expressed in the formulas. It motivates students to deal with difficult physics knowledge and causes them to feel obligated to design correct simulations. Moreover, programming simulations provides context-rich problems similar to real-life situations. Students' conceptual knowledge in physics was found to develop even regarding concepts that are known in the literature to be difficult. Within the limitations of an exploratory qualitative study, it is reasonable to attribute this learning to the computational science course and the unique learning scenarios it has enabled.

Physics and computational-science instructors face a dilemma when considering the inclusion of programming sessions in physics classes. Our observations lead us to hypothesize that one of the major problems students face when combining these three disciplines is related to cognitive load [6]. Three types of cognitive load are described in the literature: intrinsic, extraneous [6], and germane [44]. Intrinsic load is the level of difficulty inherent to the learning task, extraneous load is generated by the manner in which the information is presented to the learner, and germane load is the load devoted to the processing, construction and automation of schemata. Thus, intrinsic and extraneous are the "bad" loads and germane is the "good" one, since instructional effort should be put in creating schemata of information to make the learning efficient.

Learning within multiple disciplines (CS, physics and mathematics) may cause intrinsic load, since each discipline is difficult by itself. Moreover, it may cause extraneous load as well, due to the instruction of three different disciplines at the same time.

One of the possible ways to reduce the extraneous load would be to provide the students with more intensive physics training. Some of the training may take place apart from programming, so that students would have a chance to understand the physics aspects before they mix it with programming. This recommendation is compatible with the teacher's claim that the students lacked proper physics knowledge when programming the simulations. Similarly, the students should be taught CS strategies separately, possibly after learning some of the physics content.

The research presented in this paper explored the evolution of conceptual knowledge in physics during the programming of computational models. Other studies explored *what* elements in the computational-science environment affected this evolution (removed for anonymity). We did not, however, refer to other factors that may have been related to the students' learning. Further research is needed to explore the possible relationships among other factors and the students' learning, such as them being gifted or learning in pairs.

Another important aspect that was not addressed here is the evolution of the students' CS learning. A large portion of the class time was spent on learning programming aspects. The question of the influence of the physics context on CS learning is one that would definitely interest CS educators and may affect the instruction of the discipline.

REFERENCES

[1] Abelson, Harold, Sussman, Gerald Jay, and Sussman, Julie. 1996. *Structure and interpretation of computer programs* (2nd Ed.). Cambridge, MA: MIT Press, Cambridge, MA, USA.

[2] Bagno, Esther, Berger, Hana, and Eylon, Bat Sheva. 2008. Meeting the challenge of students' understanding of formulae in high-school physics: a

learning tool. *Physics Education,* 43, 1 (Jan. 2008), 75-82.
[3] Bagno, Esther, and Eylon, Bat Sheva. 1997. From problem solving to a knowledge structure: An example from the domain of electromagnetism. *American Journal of Physics,* 65, 8 (Aug. 1997), 726-736.
[4] Chabay, Ruth, & Sherwood, Bruce. 2008. Computational physics in the introductory calculus-based course. *American Journal of Physics*, 76, 4 (Apr. 2008), 307–313.
[5] Chabay, Ruth. W., and Sherwood, Bruce. A. 1994. *Electric and magnetic interactions*. Wiley, New York.
[6] Chandler, Paul, and Sweller, John. 1991. Cognitive load theory and the format of instruction. *Cognition and Instruction,* 8, 4, 293–332.
[7] Chang, Huey-Por, Chen, Jun-Yi, Guo, Chorng-Jee, Chen, Chung-Chih, Chang, Ching-Yi, Lin, Shean-Huey, . . . Tseng, Yaw-Teng. 2007. Investigating primary and secondary students' learning of physics concepts in Taiwan. *International Journal of Science Education,* 29, 4, 465-482.
[8] Chi, Michelene T. H., Feltovich, Paul J., and Glaser, Robert. 1981. Categorization and representation of physics problems by experts and novices. *Cognitive Science,* 5, (Apr.-Jun. 1981), 121-152.
[9] Christian, Wolfgang, and Esquembre, Francisco. 2007. Modeling physics with easy Java simulations. *The Physics Teacher,* 45, 8 (Nov. 2007), 475-480.
[10] De Jong, Ton, and Van Joolingen, Wouter R. 1998. Scientific discovery learning with computer simulations of conceptual domains. *Review of Educational Research*, 68, 2 (Jun. 1998), 179–201.
[11] Détienne, Françoise. 1990. Expert programming knowledge: a schema-based approach. In J-M Hoc, T.R.G. Green, R. Samurcay, and D. Gilmore (Eds.), *Psychology of Programming*. Academic press, London, 205-222.
[12] Esquembre, Francisco. 2004. Easy Java Simulations: A software tool to create scientific simulations in Java. *Computer Physics Communications,* 156, 2 (Jan. 2004), 199-204.
[13] Eylon, Bat Sheva, and Ganiel, Uri. 1990. Macro- micro relationships: the missing link between electrostatics and electrodynamics in students' reasoning. *International Journal of Science Education,* 12, 1, 79-94.
[14] Galili, Igal. 1996. Students' conceptual change in geometrical optics. *International Journal of Science Education,* 18, 7, 847-868.
[15] Galili, Igal, and Hazan, Amnon. 2000. Learners' knowledge in optics: interpretation, structure and analysis. *International Journal of Science Education*, 22, 1, 57–88.
[16] Ghaffar, Munazzah Abdul., Iqbal, M. Ashraf., and Hashmi, Yasser. 2007. Meaningful learning of problem transformations for a grid graph. In *Proceedings of the International Conference on Engineering Education (*ICEE 2007).
[17] Guisasola, Jenaro, Almudi, Jose M., and Zuza, Kristina. 2013. University students' understanding of electromagnetic induction. *International Journal of Science Education*, 35, 16, 2692–2717.
[18] Gunstone, Richard F., and White, Richard. T. 1981. Understanding of gravity. *Science Education,* 65, 3 (Jul.. 1981), 291-299.
[19] Halloun, Ibrahim Abou, and Hestenes, David. 1985. The initial knowledge state of college physics students. *American Journal of Physics*, 53, 11 (Nov. 1985), 1043–1055.
[20] Hasemann, Klaus, and Mansfield, Helen. 1995. Concept mapping in research on mathematical knowledge development: backgrounds, methods, findings and conclusions. *Educational Studies in Mathematics* 29 (Jul.. 1995), 45-72.
[21] Jacobs-Lawson, Joy M. and Hershey, Douglas. A. 2002. Concept maps as an assessment tool in psychology courses. *Teaching of Psychology,* 29, 1 (Jan. 2002), 25-29.
[22] Jasien, Paul. G., and Oberem, Graham. E. 2002. Understanding of elementary concepts in heat and temperature among college students and K-12 teachers. *Journal of Chemical Education,* 79, 7 (Jul. 2002), 889.
[23] Linn, Marcia C., and Eylon, Bat Sheva. 2006. Science education: integrating views of learning and instruction. In *Handbook in Educational Psychology*, Patricia. A. Alexander and Philip. H. Winne (Eds.). Lawrence Erlbaum Associates, Mahwah, NJ, USA, 511-544.
[24] Linn, Marcia C., and Eylon, Bat Sheva. 2011. Science Learning and Instruction: Taking Advantage of Technology to Promote Knowledge Integration. Routledge, New York.
[25] Lomask, M., Baron, J.B., Greig, J., and Harrison, C. 1992. ConnMap: Connecticut's use of concept mapping to assess the structure of students' knowledge of science. Presented at the Annual Meeting of the National Association of Research in Teaching (March, 1992, Cambridge, MA, USA).
[26] McCloskey, Michael. 1983. Naive theories of motion. In *Mental models*, Dedre Gentner and Albert L. Stevens (Eds.). Lawrence Erlbaum Associates, Hillsdale, NJ, USA, 299–324.
[27] McClure, John R., Sonak, Brian, and Suen, Hoy. K. 1999. Concept map assessment of classroom learning: Reliability, validity, and logistical practicality. *Journal of Research in Science Teaching,* 36, 4 (Mar. 1999), 475-492.
[28] McDermott, Lillian. C. 1984. Research on conceptual understanding in mechanics. *Physics Today*, 37, 7 (Jul. 1984), 24–32.
[29] Novak, Joseph D., and Cañas, Alberto. J. 2008. The theory underlying concept maps and how to construct and use them. *Florida Institute for Human and Machine Cognition Pensacola.* Retreived August 29, 2017, from http://cmap.ihmc.us/docs/theory-of-concept-maps.
[30] Novak, Joseph D., and Gowin, D. Bob. 1984. *Learning how to learn*. Cambridge University, Cambridge, UK.
[31] Papert, Seymour. 1981. Mindstorms: children, computers, and powerful ideas. Basic Books, New York.
[32] Pearsall, N. Renne, Skipper, Jo. El J., and Mintzes, Joel. J. 1997. Knowledge restructuring in the life sciences: A longitudinal study of conceptual change in biology. *Science Education,* 81, 2 (Apr., 1997), 193-215.
[33] Peterson, Ray, and Treagust, David, F. 1995. Developing preservice teachers' pedagogical reasoning ability. *Research in Science Education,* 25 (Sep. 1995), 291-305.
[34] Peterson, Ray, and Treagust, David, F. 1998. Learning to teach primary science through problem-based learning. *Science Education,* 82, 2 (Dec. 1998), 215-237.
[35] Redish, Edward F., and Wilson, Jack M. 1993. Student programming in the introductory physics course: MUPPET. *American Journal of Physics*, 61, 3 (Mar. 1993), 222-232.
[36] Ruiz-Primo, Maria. 2000. On the use of concept maps as an assessment tool in science: What we have learned so far. *Revista Electrónica de Investigación Educativa,* 2, 1, 1-24.
[37] Rutten, Nico, van Joolingen, Wouter R., and van der Veen, Jan T. 2012. The learning effects of computer simulations in science education. *Computers & Education,* 58, 1 (Jan. 2012), 136-153.
[38] Savinainen, Antti, Mäkynen, Asko, Nieminen, Pasi, and Viiri, Jouni. 2012. An intervention using an Interaction Diagram for teaching Newton's third law in upper secondary school. In *Physics Alive. Proceedings of the GIREP-EPEC 2011 Conference*. (Aug., 2012), 123-128.
[39] Shaffer, Peter. S., and McDermott, Lillian. C. 1992. Research as a guide for curriculum development: An example from introductory electricity. Part II: Design of instructional strategies. *American Journal of Physics,* 60, 11 (Nov. 1992), 1003-1013.
[40] Shaffer, Peter S., and McDermott, Lillian C. 2005. A research-based approach to improving student understanding of the vector nature of kinematical concepts. *American Journal of Physics,* 73, 10 (Oct. 2005), 921-931.
[41] Sherin, Bruce. L. 2001. A comparison of programming languages and algebraic notation as expressive languages for physics. *International Journal of Computers for Mathematical Learning,* 6, 1 (May. 2001), 1-61.
[42] Sloot, Peter. (n.d.). Aims & scope of the Journal of Computational Science. Retreived August 29, 2017, from http://www.elsevier.com/wps/find/journaldescription.cws_home/721195/description.
[43] Stoddart, Trish, Abrams, Robert, Gasper, Erika, and Canaday, Dana. 2000. Concept maps as assessment in science inquiry learning - a report of methodology. *International Journal of Science Education,* 22, 12 (Dec. 2000), 1221-1246.
[44] Sweller, John, van Merriënboer, Jeroen J. G., and Paas, Fred. 1998 Cognitive architecture and instructional design. *Educational Psychology Review* 10, 3 (Sep. 1998), 251-296.
[45] Thomaz, Marília F., Malaquias, I. M., Valente, M. C., and Antunes, M. J. 1995. An attempt to overcome alternative conceptions related to heat and temperature. *Physics Education,* 30, 1 (Jan. 1995), 19-26.
[46] Trowbridge, David E., and McDermott, Lillian C. 1980. Investigation of student understanding of the concept of velocity in one dimension. *American Journal of Physics,* 48, 12 (Dec. 1980), 1020-1028.
[47] Yasar, Osman, and Landau, Rubin. 2003. Elements of computational science & engineering education. *SIAM Review* 45, 4, 787-805.

Automatic Feature Selection in Markov State Models Using Genetic Algorithm

Qihua Chen*
University of Illinois at Urbana-Champaign
Urbana, IL
Qihua.chen06@gmail.com

Jiangyan Feng*
University of Illinois at Urbana-Champaign
Urbana, IL
jf8@illinois.edu

Shriyaa Mittal
University of Illinois at Urbana-Champaign
Urbana, IL
smittal6@illinois.edu

Diwakar Shukla†
University of Illinois at Urbana-Champaign
Urbana, IL
diwakar@illinois.edu

ABSTRACT

Markov State Models (MSMs) are a powerful framework to reproduce the long-time conformational dynamics of biomolecules using a set of short Molecular Dynamics (MD) simulations. However, precise kinetics predictions of MSMs heavily rely on the features selected to describe the system. Despite the importance of feature selection for large system, determining an optimal set of features remains a difficult unsolved problem. Here, we introduce an automatic approach to optimize feature selection based on genetic algorithms (GA), which adaptively evolves the most fitted solution according to natural selection laws. The power of the GA-based method is illustrated on long atomistic folding simulations of four proteins, varying in length from 28 to 80 residues. Due to the diversity of tested proteins, we expect that our method will be extensible to other proteins and drive MSM building to a more objective protocol.

KEYWORDS

Genetic algorithm, feature selection, markov state model, molecular dynamics simulation, generalized matrix Rayleigh quotient

1 INTRODUCTION

Molecular Dynamics (MD) simulation, first introduced by Alder and Wainwright[2] in the late 1950's, has evolved into a major technique to study the detailed actions and mechanisms of proteins[10, 23, 26, 35, 39]. Based on Newton's equations of motion, MD simulations can describe protein dynamics in unprecedented spatial and temporal resolution. However, one of the major challenges for MD simulations are the analysis of high dimensional data and the incompatibility between timescales accessible to MD simulation and that are functionally relevant[22, 25, 45, 46, 50]. Markov State Models (MSMs)[20, 37, 44] have recently been used to address the aforementioned issues by predicting protein dynamics at long timescales from a pool of short MD simulations. The MSM itself is a "transition probability matrix"[6], describing mathematically the memoryless transitions between metastable states. To construct a MSM, raw MD trajectories are first transformed from their Cartesian coordinates to features, such as dihedral angles[18, 33] or pairwise contact distances of a protein. This step is often called "featurization". The dimensionality of these features may be further reduced through dimensionality reduction step. One commonly used method is time-structure independent components analysis (tICA), which creates linear combinations of input features by maximizing their decorrelation time[24, 27, 38, 41, 42]. With a properly constructed MSM, useful thermodynamic and kinetic properties of the dynamic process can be extracted. Despite the attractive feature of MSMs, the thermodynamics and kinetics predicted by MSMs are highly sensitive to which features are selected to discretize the configuration space[4, 10, 28]. Ideally, features should be chosen to capture the slowest motions of the protein, which are usually the most interesting or important processes. However, determining an optimal set of features remains a considerable challenge especially when a protein system is sufficiently complex.

Currently, there are two major ways of selecting features in terms of "contact featurization", where pairwise contact distances of a protein are used as features. One is using all pairwise contact distances of a protein as features. In principle, no important information about the system is missed out since all the contact distances are considered. However, it is costly to calculate all distances even for a small protein. For a protein system with R residues, the total number of distances among each other will be $R(R\text{-}1)/2$, which creates a heavy load of calculation on computers. In addition, irrelevant features that do not contribute to the dynamics process may lead to the poor generalization performance of the model. Thus, using all available features may degrade the performance of the MSM both in speed (due to high dimensionality) and accuracy (due to irrelevant information). Alternatively, the most commonly used method is choosing a subset of contact distances based on human intuition [20]. Consequently, the thermodynamics and kinetics extracted from MSMs can be biased by the manually chosen features. In summary, either way is not appropriate for the selection of features and a more convenient, accurate and automated method for feature selection is necessary. A variety of machine learning methods have been recently reported for dimensionality reduction and/or feature

*Qihua Chen and Jiangyan Feng contributed equally to this work.
†Correspondence and requests for materials should be addressed to D.S. (email: diwakar.shukla@shuklagroup.org)

https://doi.org/10.22369/issn.2153-4136/9/2/2

selection for molecular dynamics datasets[33]. However, the use of these ideas for automatic feature selection in building MSMs has not been explored.

Here, we present a genetic-algorithm based method to select an optimal set of residue pair distances for contact featurization. Genetic algorithm (GA) is one of the advanced methods to help with dealing feature selection problems in data science. First proposed by John H. Holland[13, 15], GA is a heuristic and adaptive simulation algorithm that evolves the most fitted solution to a problem based on Darwinian natural selection laws. GA has been broadly applied to help with function optimization[48], protein folding prediction[21, 49], multiple sequence alignment[14] and more scientific investigations[16]. In nature, useful traits in genes tend to be preserved in offspring because of a higher survival probability. Like the real cases in nature, better solutions to a problem can be derived by GA according to this principal. In our case, each "gene" represents the alpha carbon distance between a residue pair, and "chromosomes" are combinations of residue pair distances. To seed the whole process, we randomly select one residue pair distance as the starting point of the GA. The adaptability of each "chromosome" (a set of residue pair distances) is quantitatively expressed as fitness scores in GA. In this study, we use generalized matrix Rayleigh quotient (GMRQ) score as the fitness score. GMRQ was recently introduced to quantitatively evaluate MSMs based on its distance from a theoretical upper limit[19, 30, 36]. The higher the GMRQ score is, the more prominent the MSM is to capture the slow underlying dynamical motions while a low GMRQ score indicates that the MSM is not able to reveal the slow dynamics of the system. Therefore, the goal of our method is optimizing a set of residue pairs that gives the highest GMRQ score. The framework of our GA-based method is adapted from the "Optimal Probes" method proposed by Mittal and Shukla[32]. In their study, an optimal choice of residue pairs, capturing the slow conformational dynamics, is successfully predicted for double electron-election resonance spectroscopy, an experimental technique capable of detecting conformational changes by monitoring the distance between electron spins.

In this method, we (1) perform contact featurization for each set of residue pair alpha carbon distances, (2) use tICA to further reduce the dimensionality of the data, (3) construct MSMs based on the reduced dimensionality, and (4) calculate GMRQ for each set of residue pair distances to evaluate the MSMs. Based on the GMRQ score, the combination of residue pair distances will be updated. The algorithm will then go back to step (1) to repeat the whole process until reaching user specified number of iterations. In the end, the set of distances with the maximum GMRQ score is chosen as an optimal set of residues for the construction of the "best MSM". To evaluate of our method, we test the GA-based method on four folding proteins with the size ranging from 28 residues to 80 residues. Our experimental results show that the method yields comparable and even better accuracy compared with using all available features. To our knowledge, this is the first attempt to automatically select proper MSM features for analysis. The GA-based method described here can be extended to larger proteins undergoing conformational changes.

2 THEORY AND METHODS

Molecular Dynamics (MD) Simulation Dataset. MD simulation datasets of the four folding proteins for analysis were generated by Lindorff-Larsen *et al*[26]. The four proteins (BBA, Villin, WW domain and λ-repressor) vary in length from 28 to 80 amino acids. More details of the simulations are summarized in Table 1. For the analysis, we retain all the trajectory frames. Three small proteins (BBA, Villin and WW domain) are chosen to evaluate the proposed method and the best GMRQ achieved using all contact distances serves as the benchmark. The 80-amino-acid λ-repressor is used to test the feasibility of the method on large proteins, as using all distances is impractical.

Table 1: Protein and Trajectory Information.

Protein	PDB	Residues	Total simulation time (μs)
BBA	1FME	28	325
Villin	2F4K	47	429
WW domain	2F21	35	1137
λ-repressor	1LMB	80	643

Markov State Models (MSMs). In this study, the goal is optimizing a set of residue pair distances to build the best MSM based GMRQ. MSMs are kinetic models that reveal the dynamics of a system[6, 17, 37, 39, 40]. An MSM describes a network of metastable conformational states and reveals the probabilities of each state performing jumps from one to another over an appropriate time resolution (τ, also called lag time). The jumps are memoryless, which means the probability to transit to the present state is not dependent on the previous ones. Such information is presented in a "transition probability matrix" by MSM, where an $n \times n$ square matrix depicts the transitions among n states[6]. The probability of each jump can be expressed according to the equation below:

$$p_j(t+\tau) = \sum_{i=1}^{n} p_i(t) T_{ij}(\tau) \tag{1}$$

The equation can also be expressed in a matrix form:

$$p^T(t+\tau) = p^T(t) T(\tau) \tag{2}$$

where $p_i(t)$ is a population vector whose elements show the probability at time t, $p_j(t+\tau)$ is a population vector after time τ, $T_{ij}(\tau)$ is the probability to jump from state i to state j and $T(\tau)$ is the transition probability matrix that $T(\tau) \in \mathbb{R}^{n \times n}$. Further details of the transition matrix can be found in literatures[6, 44].

The transition probability matrix can be decomposed into eigenfunctions and eigenvalues shown below:

$$T(\tau) \circ \psi_i = \lambda_i \psi_i \tag{3}$$

where ψ_i is the eigenfunction and λ_i are the real eigenvalues that $\lambda_i \leq 1$, arranged in descending order.

Here, each step of the MSM building process used in this study is described in detail. All the hyperparameters (e.g. the number of tICA components, tICA lagtime, the number of clusters, the number of MSM timescales and MSM lagtime) are shown in Table 2.

(1) Featurization. To construct an MSM, the first step is to process the datasets that we plan to work on. In our case, we use the MD simulation data sets listed in Table 1. The datasets are given in the form of MD trajectories, which present series of motion of the protein atoms in a frame-wise arrangement. Because the simulated movements recorded in Cartesian coordinates are not ideal for analysis, and too much noise not relevant to our study may be included, it is better to interpret the data in other ways. As a result, a lot of reasonable metrics such as dihedral angle[18] and contact distances between residue pairs are used to featurize the data. The featurization method we choose here is contact distance analysis. By using such technique, more useful information can be extracted from the redundant MD trajectories. Again, our goal of this study is to optimize the choice of residue pairs for contact distance calculation, so that an MSM with more information and less noise can be found by this method. The method outlined in this study could be applied to any chosen set of features calculated using simulation data.
(2) Dimensionality reduction. We further processed our featurized data by tICA so as to reduce the dimensionality of the data. After featurization, the featurized data were projected onto linear subspaces of the slowest dynamics. The components of tICA are termed time structure-based independent components (tICs), which are linear combination of the input features (a set of contact distances in our case). Top tICs capture the slowest motion captured by tICA and usually represent the most interesting dynamics[24, 27, 38, 41, 42].
(3) Clustering. We perform mini-batch k-means clustering on the processed data. Clustering refers to the coarse graining analysis that groups certain datasets based on their similarities, so that macrostates can be formed to be better understood. Commonly used clustering algorithms, such as mini-batch k-means[6, 30, 34], mini-batch k-medoids[8, 12] and k-centers[5, 24], have shown similar performance when the data is preprocessed with tICA[27, 33, 38, 41].
(4) MSM construction. After the clustering, a MSM can be built based on the processed datasets. The process was implemented in a Python environment and the software involved to produce the analysis above include Numpy[3], MDTraj[28] and MSMBuilder3.8[4].

Generalized Rayleigh Quotient (GMRQ). In short, an ideal MSM should successfully identify the slowest dynamics of the protein. Because the state decomposition mentioned above reveals the dynamical processes in the system, the identification of true eigenfunction and eigenvalues become the major problem for scientists to solve. A more quantitative method is needed to help evaluate and find the true state decomposition, which is directly related to the choice of metrics in the featurization stage.

To help solve this problem, GMRQ was introduced as a quantitative way of evaluating the quality of an MSM[30, 30, 36]. GMRQ is derived from the variational principle that adds up the first m eigenvalues, which denote the slowest m dynamical processes in the system. The variational principles set an upper boundary[19, 30, 36] for the total sum of real eigenvalues shown below:

$$GMRQ \equiv \sum_{i=1}^{m} \hat{\lambda}_i \leq \sum_{i=1}^{m} \lambda_i \tag{4}$$

where the $\hat{\lambda}_i$ is the estimated eigenvalue and the λ_i is the real eigenvalue. In this study, as we try to maximize GMRQ score to approach the upper boundary, the larger the GMRQ score we get, the closer we are to the slowest dynamics of the protein.

To help avoid overfitting, cross-validation must be applied to evaluate our GMRQ scores. The dataset from the MD simulation is split into a training set and a test set. The training set is first used to estimate the model parameters such as the eigenvalues, then the estimated model is applied to score its performance in the test set. In this way, the model will not be biased by overfitting the data onto the model. The process of deriving GMRQ scores is achieved by Osprey package[29] and the recruited parameters are shown in Table 2. Mean GMRQ of five cross-validation iterations are used for the analysis.

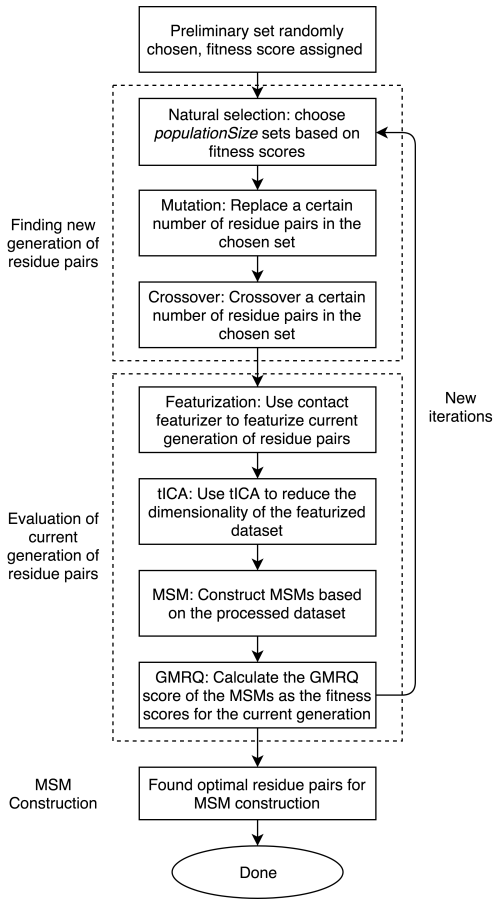


Figure 1: The flow chart showing the whole process of our GA-based method.

Genetic-algorithm-based Method for automatic feature selection in Markov State Models (MSMs). To simulate the natural selection process according to the Darwinian law, we must decide how the natural selection principles are implemented in our algorithm. In this section, we introduce our basic operators of GA, the framework that we follow to perform GA, and the protocol we adopt to finally generate optimal residue pairs. The construction of the GA is based on the work of Mittal and Shukla[32].

In the field of programming, operators refer to the actions to take during each step of execution of the algorithm. The basic operators in our study are composed of natural selection, mutation, and crossover. In the following section, we discuss our method to help predict an optimized set of residue pair distances for MSM construction using genetic algorithm. We also provide the series of steps as a flow chart shown in Figure 1. Some important parameters that are involved in these steps are: *populationSize*, *percentMutation* and *percentCrossover*. These parameters can be changed according to user's need.

(1) A set of all possible residue pairs is identified. $R(R\text{-}1)/2$ residue pairs for a protein with R residues.
(2) *populationSize* preliminary sets of residue pair are randomly selected from the set of all possible residue pairs for the first iteration. Each set contains only one residue pair as the starting point for selection. These sets of residue pairs serve as the initial generation G_0 and are assigned fitness scores of 0.
(3) Natural selection is performed to choose the new generation of residue pairs according to their fitness scores. The natural selection operator corresponds to the reproductive process in nature, which selects genomes with ideal traits for breeding offspring. In our case, we define a parameter *populationSize* that describes the number of elements randomly chosen from the parental set for a new generation G_{new}.
(4) Mutation is performed to maintain diversity to the current generation of residue pair selections. The mutation operator corresponds to the mutation process in nature to increase genetic diversity. In our version of GA, we define a parameter *percentMutation* to maintain a ratio of mutation in our combination of residue pairs. During the mutation step, the number of residue pairs to be mutated are generated by (*percentMutation* × *populationSize*)/100 from G_{new} and those residue pairs are randomly replaced by other residue pairs that are excluded in the G_{new}.
(5) Crossover is performed to add more diversity to the current generation. The crossover operator corresponds to the natural recombination process of chromosomes. Here, we define another parameter *percentCrossover* as the percentage of crossover in our combination of residue pairs. The number of residue pairs to perform crossover is generated by (*percentCrossover* × *populationSize*)/100 from G_{new}. The residue pair distance sets will then be swapped according to the number calculated before to create a new combination of residue pair distances.
(6) Evaluations are performed to assign fitness scores to the newly generated residue pairs. MSMs are constructed based on contact featurization using the current generation of residue pairs, and GMRQ scores are calculated accordingly to serve as fitness scores.
(7) If more iterations are designed to be finished, the next iteration should restart at step (3) and use the current generation of residue pairs as G_0. As the iteration number increases, the fitness scores for the selection of residue pairs should show a convergence of fitness scores.

All the parameters used in this study are organized in Table 2.

Table 2: Model Hyperparameteres.

Featurization		
α-carbon contact distances		
Decomposition	Components	Lag time (ns)
tICA	5	0.2
Clustering	Clusters	
Mini-batch k-means	200	
Model fitting	N_timescales	Lag time (ns)
MSM	5	50
Scoring		
GMRQ		
Cross-validation	Iterations	Test set size
Shuffle & Split	5	0.5
Genetic algorithm		
Iterations	40	
populationSize	20%	
percentMutation	50%	
percentCrossover	20%	

3 RESULTS

In this section, we discuss the optimized set of residue pair distances obtained from our GA-based approach. As described in the method part, the unbiased and extensive MD simulation data (>100μs) simulating the folding process of the proteins is taken from literature[26]. Preliminary sets of residue pair distance are randomly selected from the set of all the possible residue pairs as the starting point of the genetic algorithms. These sets go through selection, mutation and crossover steps to provide a new generation of residue pair distances. In the setting of GA, we choose a population size of 10%, mutation percentage of 50% and crossover percentage of 20%. Next, the newly generated residue pair distances are used to build MSMs and assign new GMRQ scores (fitness scores) for evaluation. The next iteration will then go back to the selection step and select according to the newly assigned fitness scores. As the process goes through more iterations, the GMRQ scores will converge and a best GMRQ score can be found.

This method is applied to 4 proteins for demonstration of its functionality: BBA, Villin, WW domain and λ-repressor. Among the proteins, 3 proteins (BBA, Villin and WW domain) are small proteins, each of which has a residue number that smaller than 40 ($R < 40$). To examine the effectiveness of our method, we compare the GMRQ scores and implied timescales with their corresponding benchmark values (using all contact distances as features). In the end, we show the ability of our GA-based method to process larger

Table 3: Comparison of the best GMRQ scores generated and benchmark GMRQ scores from all contact featurization. The fraction of all residue pairs is the fraction of chosen residue pairs in all residue pairs. The best GMRQ refers to the highest GMRQ score that we obtain from MSMs using residue pair distance features given by our genetic algorithm approach, and the benchmark GMRQ score is the GMRQ provided by the MSM constructed with all contact featurization. The deviation column if the deviation of our best GMRQ score from the benchmark GMRQ score.

Protein	Residues	Number of chosen distances	Fraction of pairs (%)	Best GMRQ	Benchmark GMRQ	Deviation (%)
BBA (1FME)	28	47	12.43	4.445	4.239	+4.80
Villin (2F4K)	35	61	10.25	3.203	3.705	-13.5
WW domain (2F21)	35	4	0.67	4.198	4.111	+2.12
λ-repressor (1LMB)	80	60	1.90	4.956	N/A	N/A

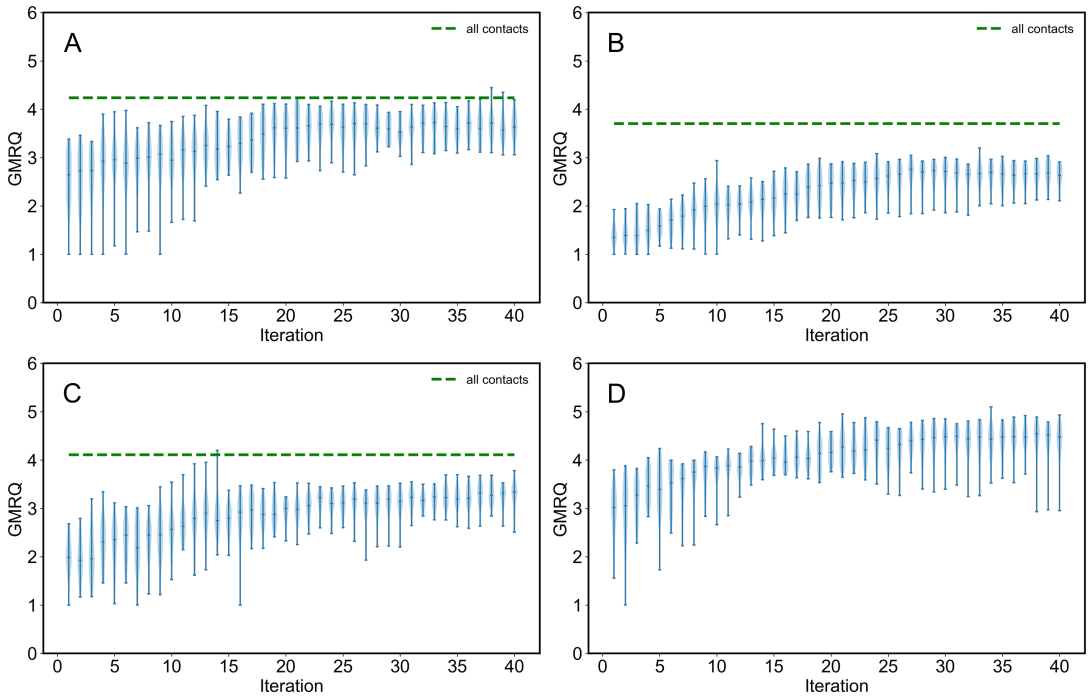


Figure 2: GMRQ scores reflecting the MSMs based on the GA-predicted residue pairs. (A) BBA (PDB ID: 1FME), (B) Villin (PDB ID: 2F4K), (C) WW domain (PDB ID: 2F21), (D) λ-repressor (PDB ID: 1LMB). Green, dashed lines indicate the best GMRQ score corresponding to MSMs based on all contact featurization. Each violin plot shows the increase of GMRQ scores over 40 iterations. In each set of data, the center dot shows the mean values and the vertical line shows the range of this GMRQ data set.

proteins such as λ-repressor, a protein with 80 residues, which cannot be featurized using all contact distances.

3.1 Our GA-based method proved effectiveness in generating GMRQ scores that are close to the highest possible values given by all contacts featurizer.

We featurize the small proteins (BBA, Villin, WW domain) using all contacts featurization to produce benchmark GMRQ scores for comparison. Benchmark GMRQ scores will serve as a comparable reference to evaluate the performance of our method of using GA to generate optimal residue pairs as featurization metrics. By comparing the best GMRQ scores from our GA-based method to the benchmark GMRQ scores, we are able to check whether our method successfully provides the residue pair sets that depict the slowest process of the protein dynamics. We also apply this method to λ-repressor, a medium sized protein with 80 residues, to show its ability to process larger proteins. The GMRQ scores are calculated by adding up the eigenvalues of the transition probability matrix provided by MSMs[30, 36]. The theoretical upper limit of GMRQ score is 6 in all cases[19, 30, 36], due to the fact that the number of MSM timescales is chosen to be 5 in the MSM settings. Therefore, in our case, high GMRQ score that approaches 6 usually

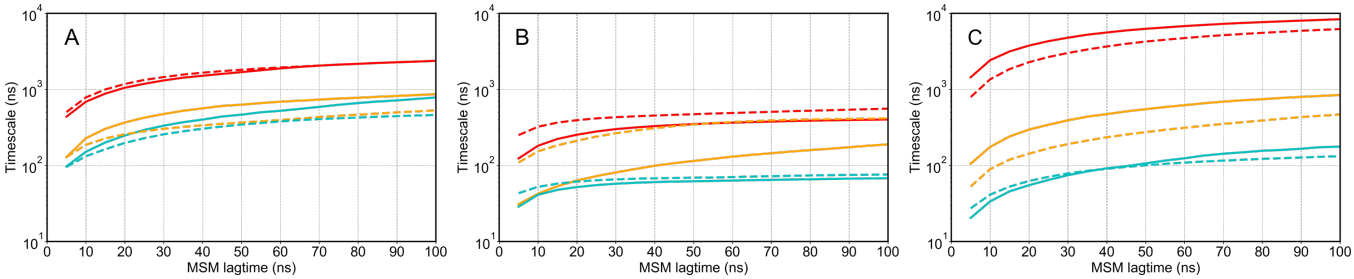


Figure 3: The first three slowest implied timescales as a function of MSM lag time. (A) BBA (PDB ID: 1FME), (B) Villin (PDB ID: 2F4K), (C) WW domain (PDB ID: 2F21). The red, yellow and cyan colored lines indicate the slowest, second slowest and third slowest implied timescales, respectively. Dashed lines correspond to the reference value given by the MSMs built on all contacts featurization. Solid lines correspond to the implied timescales given by the MSMs achieved by using the set of distances optimally chosen by our GA-based method.

suggests a better ability of an MSM to capture the slowest process, whereas low GMRQ score implies ineffective state decomposition during the MSM construction process. All information regarding the GMRQ scores and residue pair selection is summarized in Table 3. As shown in Figure 2, all GMRQ scores converged over 40 iterations. In Figure 2A, the highest GMRQ score for BBA is around 4.445, which is higher than the benchmark GMRQ score (4.239). Similar traits are shown by WW domain in Figure 2C that the best GMRQ from GA (4.198) is higher than the benchmark (4.111). However, one exception happens in Villin, shown in Figure 2B. In Figure 2B, the best predicted GMRQ (3.203) does not reach the benchmark (3.705). More iterations for Villin are needed to reach a best GMRQ score that is higher than the benchmark, but there exists a trade-off between the accuracy and computational resource needed. Overall, the percent variances between our predicted GMRQ score and the benchmark GMRQ score are +4.8% for BBA, -13.5% for Villin and +2.12% for WW domain, in which Villin has the highest difference compared to the other two proteins.

Similar analysis is applied to λ-repressor, except that λ-repressor lacks a benchmark GMRQ score due to its higher number of residues. Hence, there is no reference value to compare in this case. The best GMRQ score given over 40 iterations is around 4.956. Considering that the upper limit of the GMRQ score in this system is 6, we believe that a score of 4.956 is a relatively high GMRQ that effectively captures the slow dynamics of the protein folding mechanisms. Therefore, we can conclude that the method proves its ability to provide the optimal selection of residue pairs for the construction of the best MSM.

3.2 Implied timescale plots show that predicted optimal sets of residue pair distances are able to successfully capture the slowest dynamics in the proteins.

By plotting lag time dependent implied timescale plots, we can quantitatively visualize the slow modes of protein dynamics. Figure 3 shows the comparison between the converged slowest implied timescales provided by all contact featurization and our GA-based method. Again, the reference values are provided by utilizing all residue pair distances as features. Since λ-repressor is too big for all contacts featurization, there is no benchmark data available and its implied timescale is not shown. In Figure 3A, the slowest implied timescales (solid and dashed red lines) of BBA nearly overlap with each other, indicating that our method has chosen a set residue pair distances that captures the slowest process. In addition, the predicted second and third slowest implied timescales (yellow and cyan) are slower than the corresponding timescale for the benchmarks. In Figure 3B, the predicted implied timescales of Villin has a larger deviation. This inconsistency will be explained and justified in the next paragraph. In the case of WW domain (Figure 3C), we capture a slower timescale than the benchmarks. We find that inclusion of all residue pair distances can add noises to the model, and our GA-based method helps improve the MSM construction by excluding those irrelevant features.

3.3 Number of selected distances may reflect the degree of complexity of the protein folding mechanism.

Other than the GMRQ scores and implied timescale plots, more information can be obtained from the sets of residue pair distances. In Table 2, we collect and summarize the number of distances selected by GA and the actual residue numbers in each protein. One interesting thing is that the number of residues in a protein is not necessary correlated to the number of distances needed to capture its slowest dynamics. For example, it can be observed in Table 2 that although both Villin and WW domain have 35 residues in their sequences, WW domain only needs 4 distances of residue pairs while Villin requires 61 distances. This may be due to the complex folding mechanism of Villin. Though both proteins are fast-folding proteins with small numbers of residues, the secondary structure elements in Villin fold more independently without much interactions[31]. Such minimized interaction or minimal frustration makes the folding kinetics fast for Villin, according to the folding funnel theory[7]. Consequently, because the protein folds quickly, this phenomenon suggests a continuous reduction in energy in the folding funnel[7, 9], which implies multiple parallel pathways during the folding hypothesis[51]. On the other hand, WW domain folds much slower than Villin[6] and has more consistent folding

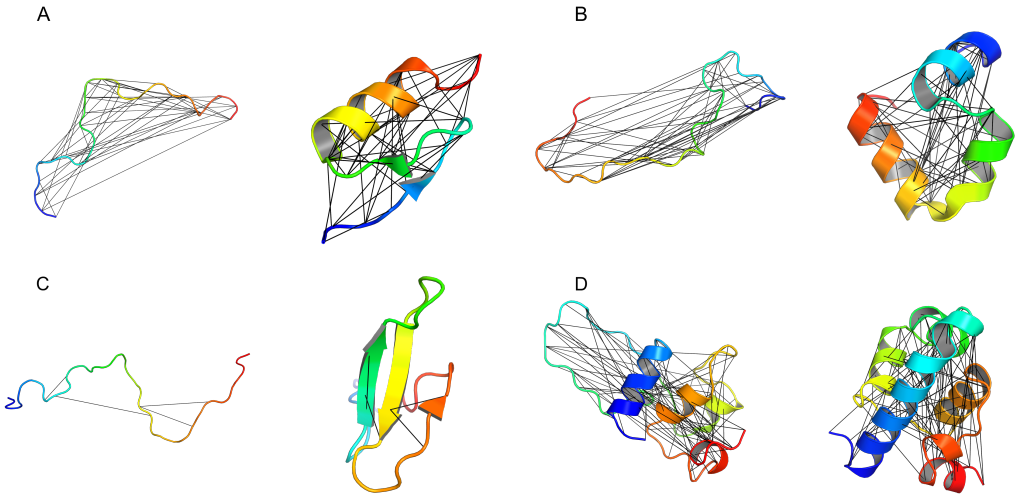


Figure 4: GA-chosen residue pairs visualized on the unfolded MD structures and folded crystal structures. (A) BBA (PDB ID: 1FME), (B) Villin (PDB ID: 2F4K), (C) WW domain (PDB ID: 2F21), (D) λ-repressor (PDB ID: 1LMB). The black lines specify the distances between residue pairs chosen by our GA-based method, which capture the slowest dynamics of the proteins.

pathways[1]. The independent features in Villin make it hard for GA to fully capture its slowest dynamics.

In a previous study, Feng and Shukla[11] utilized evolution couplings (ECs) as functional features to capture protein folding and conformational dynamics, which gives the similar results for Villin and WW domain. Their work identified that Villin needs 73 ECs and WW domain only needs 5 ECs to fully describe the protein dynamics. They stated that more ECs are needed if the ECs has low correlation. Here, our results show the same trait that Villin requires more features for identification of its slowest dynamic processes, which is reasonable due to the folding complexity of Villin comparing to other fast-folding small proteins. To fully capture the slow dynamics of proteins like Villin, a large number of features should be included from its whole dataset. This is a different scenario comparing to capturing the dynamics of the proteins that needs small numbers of features, which is a problem easier for GA to solve. For proteins like Villin, other methods needs to be explored for a more efficient way to capture the slowest dynamics. Although our method results in some degrees of deviations from the benchmarks (shown in Figure 2B and 3B), it still shows effectiveness in dealing with proteins with complicated kinetics.

To present our predicted results in a more understandable way, we visualize the optimal sets of residue pairs for all four proteins in Figure 4. Each section (A, B, C and D) of Figure 4 consists of two parts, representing the unfolded and folded structure of the protein respectively. It is easy to notice that the residue pair distances chosen by our method spread out in the protein to capture the complex dynamics of protein folding.

4 CONCLUSIONS

Feature selection of MSM construction determines the accuracy of predicted kinetics properties. Currently, the selection of features is done using trial and error. The utilization of GMRQ score enables a quantitative description of the accuracy of MSMs in representing the molecular dynamics observed in a simulation dataset. Using GMRQ score as fitness score, we introduce a GA-based method in order to optimize a set of residue pair distances that produce superior MSMs. In this study, we have shown that our method can provide an automatic, efficient and accurate way to choose the optimal residue pair distances as features for MSMs construction. This significantly improves the efficiency in the overall process of building MSMs while still guarantees the quality of MSMs to capture the slowest protein dynamics. Due to the diversity of tested proteins, our method can be widely applied to other proteins to help with the feature selection process and we anticipate that this method will shift MSM building one step closer to a systemic and objective protocol. It is important to be aware that the underlying assumption of this approach is that the slowest dynamic processes correspond to the process of interest. However, this assumption can be challenged in the case of insufficient sampling or inaccurate force field.

However, the method also has some limitations. The proposed method belongs to the class of wrapper methods for feature selection that find the “optimal" feature subset by iteratively selecting features based on the classifier performance. The performance of these methods drops significantly for datasets with large number of important but uncorrelated features. Our method also does not perform well on systems with complex dynamics that requires a large number of features to capture the underlying dynamics. In other words, the effectiveness partially depends on the complexity of the conformational changes in the protein, which is shown in the discussion of Villin. As the folding complexity increases, more pathways are available for the protein, so the selection of residue pairs may not fully depict the slowest dynamics of the protein. However, a large number of biologically relevant dynamic processes have been

shown to involve only a few important features[11, 22, 25, 43, 45–47, 50]. In addition, sequence information and crystal structure of the protein should be known, and sufficient amount of MD simulation data should be generated to apply our method. In conclusion, the proposed algorithm, can help identify essential residue pair distances for featurization and exclude noises for MSM construction with high efficiency.

5 REFLECTION

The year-long Blue Waters Internship enriched my experience in many aspects. This opportunity was rare and precious, especially because I can utilize one of the leading-edge petascale computational resources on the Blue Waters Supercomputer. I was excited to be offered the opportunity to meet other interns to study and practice computational skills together. Starting last summer, I have been involved in a variety of activities, including a two-week educational workshop at University of Illinois at Urbana-Champaign, regular webinars, monthly reports and preparing a manuscript. Majoring in Material Science, I joined the internship with limited computational experience. However, I quickly gained essential skills and became adept with the help of internship coordinators, my research advisor and mentors in the lab. In addition, working on the projects helped me to be familiar with the life in a research group and be better prepared for the graduate school. My presentation skills were improved through attending group meetings and poster sessions. I also practiced my writing skills through regular progress reports and writing this manuscript. Overall, the past year was a busy year, but it has became a unique experience in my undergraduate studies.

ACKNOWLEDGMENTS

The authors thank D. E. Shaw Research for MD simulation trajectories (BBA, Villin, WW domain and λ-repressor). Q.C. would like to thank the support of the Shodor Education Foundation and the Blue Waters Student Internship Program. J.F. would like to thank Chia-chen Chu fellowship for support. S.M. is supported by the CSE Fellows Program, Computational Science and Engineering at University of Illinois, Urbana-Champaign, Urbana, IL. D. S. acknowledges support from the Foundation for Food and Agriculture Research via the new innovator program.

REFERENCES

[1] Silvio a Beccara, Tatjana Škrbić, Roberto Covino, and Pietro Faccioli. 2012. Dominant folding pathways of a WW domain. *Proc. Natl. Acad. Sci. U.S.A* 109, 7 (2012), 2330–2335.

[2] BJ Alder and TEf Wainwright. 1957. Phase transition for a hard sphere system. *J. Chem. Phys.* 27, 5 (1957), 1208–1209.

[3] David Ascher, Paul F Dubois, Konrad Hinsen, Jim Hugunin, Travis Oliphant, et al. 2001. Numerical python.

[4] Kyle A Beauchamp, Gregory R Bowman, Thomas J Lane, Lutz Maibaum, Imran S Haque, and Vijay S Pande. 2011. MSMBuilder2: modeling conformational dynamics on the picosecond to millisecond scale. *J. Chem. Theory. Comput.* 7, 10 (2011), 3412–3419.

[5] Kyle A Beauchamp, Daniel L Ensign, Rhiju Das, and Vijay S Pande. 2011. Quantitative comparison of villin headpiece subdomain simulations and triplet–triplet energy transfer experiments. *Proc. Natl. Acad. Sci. U.S.A* 108, 31 (2011), 12734–12739.

[6] Gregory R Bowman, Vijay S Pande, and Frank Noé. 2014. Introduction and overview of this book. In *An introduction to Markov state models and their application to long timescale molecular simulation*. Springer, 1–6.

[7] Joseph D Bryngelson, Jose Nelson Onuchic, Nicholas D Socci, and Peter G Wolynes. 1995. Funnels, pathways, and the energy landscape of protein folding: a synthesis. *Proteins: Struct., Funct., Bioinf.* 21, 3 (1995), 167–195.

[8] John D Chodera, Nina Singhal, Vijay S Pande, Ken A Dill, and William C Swope. 2007. Automatic discovery of metastable states for the construction of Markov models of macromolecular conformational dynamics. *J. Chem. Phys.* 126, 15 (2007), 04B616.

[9] Ken A Dill and Hue Sun Chan. 1997. From Levinthal to pathways to funnels. *Nat. Struct. Mol. Biol.* 4, 1 (1997), 10.

[10] S Doerr, MJ Harvey, Frank Noé, and G De Fabritiis. 2016. HTMD: high-throughput molecular dynamics for molecular discovery. *J. Chem. Theory. Comput.* 12, 4 (2016), 1845–1852.

[11] Jiangyan Feng and Diwakar Shukla. 2018. Characterizing Conformational Dynamics of Proteins Using Evolutionary Couplings. *J. Phys. Chem. B* (2018).

[12] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. 2001. *The elements of statistical learning*. Vol. 1. Springer Series in Statistics New York.

[13] David E Goldberg and John H Holland. 1988. Genetic algorithms and machine learning. *Mach. Learn.* 3, 2 (1988), 95–99.

[14] Cedric Gondro and Brian P Kinghorn. 2007. A simple genetic algorithm for multiple sequence alignment. *Genet. Mol. Res.* 6, 4 (2007), 964–982.

[15] John Henry Holland. 1992. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press.

[16] Md Tamjidul Hoque and Sumaiya Iqbal. 2017. Genetic algorithm-based improved sampling for protein structure prediction. *Int. J. Bio. Inspir. Com.* 9, 3 (2017), 129–141.

[17] Gerhard Hummer and Attila Szabo. 2014. Optimal dimensionality reduction of multistate kinetic and Markov-state models. *J. Phys. Chem. B* 119, 29 (2014), 9029–9037.

[18] Brooke E Husic, Robert T McGibbon, Mohammad M Sultan, and Vijay S Pande. 2016. Optimized parameter selection reveals trends in Markov state models for protein folding. *J. Chem. Phys.* 145, 19 (2016), 194103.

[19] Brooke E Husic and Vijay S Pande. 2017. Note: MSM lag time cannot be used for variational model selection. *J. Chem. Phys.* 147, 17 (2017), 176101.

[20] Brooke E Husic and Vijay S Pande. 2018. Markov state models: From an art to a science. *J. Am. Chem. Soc.* 140, 7 (2018), 2386–2396.

[21] M Kenneth Jr, Scott M LeGrand, et al. 2012. *The protein folding problem and tertiary structure prediction*. Springer Science & Business Media.

[22] Kai J Kohlhoff, Diwakar Shukla, Morgan Lawrenz, Gregory R Bowman, David E Konerding, Dan Belov, Russ B Altman, and Vijay S Pande. 2014. Cloud-based simulations on Google Exacycle reveal ligand modulation of GPCR activation pathways. *Nature chemistry* 6, 1 (2014), 15.

[23] Thomas J Lane, Diwakar Shukla, Kyle A Beauchamp, and Vijay S Pande. 2013. To milliseconds and beyond: challenges in the simulation of protein folding. *Curr. Opin. Struct. Biol.* 23, 1 (2013), 58–65.

[24] Lisa J Lapidus, Srabasti Acharya, Christian R Schwantes, Ling Wu, Diwakar Shukla, Michael King, Stephen J DeCamp, and Vijay S Pande. 2014. Complex pathways in folding of protein G explored by simulation and experiment. *Biophys. J.* 107, 4 (2014), 947–955.

[25] Morgan Lawrenz, Diwakar Shukla, and Vijay S Pande. 2015. Cloud computing approaches for prediction of ligand binding poses and pathways. *Scientific reports* 5 (2015), 7918.

[26] Kresten Lindorff-Larsen, Stefano Piana, Ron O Dror, and David E Shaw. 2011. How fast-folding proteins fold. *Science* 334, 6055 (2011), 517–520.

[27] Mohammad M. Sultan and Vijay S Pande. 2017. tICA-Metadynamics: Accelerating Metadynamics by using kinetically selected collective variables. *J. Chem. Theory. Comput.* 13, 6 (2017), 2440–2447.

[28] Robert T McGibbon, Kyle A Beauchamp, Matthew P Harrigan, Christoph Klein, Jason M Swails, Carlos X Hernández, Christian R Schwantes, Lee-Ping Wang, Thomas J Lane, and Vijay S Pande. 2015. MDTraj: a modern open library for the analysis of molecular dynamics trajectories. *Biophys. J.* 109, 8 (2015), 1528–1532.

[29] Robert T McGibbon, Carlos X Hernández, Matthew P Harrigan, Steven Kearnes, Mohammad M Sultan, Stanislaw Jastrzebski, Brooke E Husic, and Vijay S Pande. 2016. Osprey: Hyperparameter optimization for machine learning. *J. O. S. S* 1 (2016), 00034.

[30] Robert T McGibbon and Vijay S Pande. 2015. Variational cross-validation of slow dynamical modes in molecular kinetics. *J. Chem. Phys.* 142, 12 (2015), 03B621_1.

[31] James C McKnight, Don S Doering, Paul T Matsudaira, and Peter S Kim. 1996. A thermostable 35-residue subdomain within villin headpiece. *Mol. Biol.* (1996).

[32] Shriyaa Mittal and Diwakar Shukla. 2017. Predicting Optimal DEER Label Positions to Study Protein Conformational Heterogeneity. *J. Phys. Chem. B* 121, 42 (2017), 9761–9770.

[33] Shriyaa Mittal and Diwakar Shukla. 2018. Recruiting machine learning methods for molecular simulations of proteins. *Mol. Simul.* 44, 11 (2018), 891–904.

[34] Alexander S Moffett, Kyle W Bender, Steven C Huber, and Diwakar Shukla. 2017. Allosteric Control of a Plant Receptor Kinase through S-Glutathionylation. *Biophys. J.* 113, 11 (2017), 2354–2363.

[35] Alexander S Moffett and Diwakar Shukla. 2018. Using molecular simulation to explore the nanoscale dynamics of the plant kinome. *Biochem J* 475, 5 (2018), 905–921.

[36] Frank Noé and Feliks Nuske. 2013. A variational approach to modeling slow processes in stochastic dynamical systems. *Multiscale Model Simul.* 11, 2 (2013), 635–655.
[37] Vijay S Pande, Kyle Beauchamp, and Gregory R Bowman. 2010. Everything you wanted to know about Markov State Models but were afraid to ask. *Methods* 52, 1 (2010), 99–105.
[38] Guillermo Pérez-Hernández, Fabian Paul, Toni Giorgino, Gianni De Fabritiis, and Frank Noé. 2013. Identification of slow molecular order parameters for Markov model construction. *J. Chem. Phys.* 139, 1 (2013), 015102.
[39] Nuria Plattner, Stefan Doerr, Gianni De Fabritiis, and Frank Noé. 2017. Complete protein–protein association kinetics in atomic detail revealed by molecular dynamics simulations and Markov modelling. *Nat. Chem.* 9, 10 (2017), 1005.
[40] Jan-Hendrik Prinz, Hao Wu, Marco Sarich, Bettina Keller, Martin Senne, Martin Held, John D Chodera, Christof Schütte, and Frank Noé. 2011. Markov models of molecular kinetics: Generation and validation. *J. Chem. Phys.* 134, 17 (2011), 174105.
[41] Christian R Schwantes and Vijay S Pande. 2013. Improvements in Markov state model construction reveal many non-native interactions in the folding of NTL9. *J. Chem. Theory. Comput.* 9, 4 (2013), 2000–2009.
[42] Christian R Schwantes, Diwakar Shukla, and Vijay S Pande. 2016. Markov state models and tICA reveal a nonnative folding nucleus in simulations of NUG2. *Biophysical journal* 110, 8 (2016), 1716–1719.
[43] Zahra Shamsi, Alexander S Moffett, and Diwakar Shukla. 2017. Enhanced unbiased sampling of protein dynamics using evolutionary coupling information. *Scientific Reports* 7, 1 (2017), 12700.
[44] Diwakar Shukla, Carlos X Hernández, Jeffrey K Weber, and Vijay S Pande. 2015. Markov state models provide insights into dynamic modulation of protein function. *Acc. Chem. Res.* 48, 2 (2015), 414–422.
[45] Diwakar Shukla, Yilin Meng, Benoît Roux, and Vijay S Pande. 2014. Activation pathway of Src kinase reveals intermediate states as targets for drug design. *Nat. Commun.* 5 (2014), 3397.
[46] Diwakar Shukla, Ariana Peck, and Vijay S Pande. 2016. Conformational heterogeneity of the calmodulin binding interface. *Nat. Commun.* 7 (2016), 10910.
[47] Mohammad M Sultan, Gert Kiss, Diwakar Shukla, and Vijay S Pande. 2014. Automatic selection of order parameters in the analysis of large scale molecular dynamics simulations. *J. Chem. Theory. Comput.* 10, 12 (2014), 5217–5223.
[48] Mohammad Taherdangkoo, Mahsa Paziresh, Mehran Yazdi, and Mohammad Hadi Bagheri. 2013. An efficient algorithm for function optimization: modified stem cells algorithm. *Cent. Eur. J. Eng.* 1, 3 (2013), 36–50.
[49] Ron Unger and John Moult. 1993. Genetic algorithms for protein folding simulations. *Biochem Mol Biol J.* 231, 1 (1993), 75–81.
[50] Dan K Vanatta, Diwakar Shukla, Morgan Lawrenz, and Vijay S Pande. 2015. A network of molecular switches controls the activation of the two-component response regulator NtrC. *Nat. Commun.* 6 (2015), 7283.
[51] Li Zhu, Kingshuk Ghosh, Michael King, Troy Cellmer, Olgica Bakajin, and Lisa J Lapidus. 2011. Evidence of multiple folding pathways for the villin headpiece subdomain. *J. Phys. Chem. B* 115, 43 (2011), 12632–12637.

Teaching and Learning Graph Algorithms Using Animation

Y. Daniel Liang
Department of Computer Science
Georgia Southern University
Savannah Campus, GA 31419
y.daniel.liang@gmail.com

ABSTRACT
Graph algorithms have many applications. Many real-world problems can be solved using graph algorithms. Graph algorithms are commonly taught in the data structures, algorithms, and discrete mathematics courses. We have created two animations to visually demonstrate the graph algorithms. The first animation is for depth-first search, breadth-first search, shortest paths, connected components, finding bipartite sets, and Hamiltonian path/cycle on unweighted graphs. The second animation is for the minimum spanning trees, shortest paths, travelling salesman problems on weighted graphs. The animations are developed using HTML, CSS, and JavaScript and are platform independent. They can be viewed from a browser on any device. The animations are useful tools for teaching and learning graph algorithms. This paper presents these animations.

Keywords
Algorithms, animation, data structures, discrete mathematics, graphs

1. INTRODUCTION
Graphs are simple mathematical structures. A graph consists of a set of vertices and a set of edges for connecting vertices. In a weighted graph, each edge is assigned with a value, called a weight. Graphs have many important applications. For example, a map can be modelled using a graph. The cities are the vertices and the roads connecting the cities are the edges, and the distances are the weights on the edges. The problem of finding the shortest distance between two cities can be solved by finding a shortest path between the two vertices in the graph. Many algorithms have been developed to solve a variety of graph problems. The common graph problems for unweighted graphs covered in the data structures, algorithms and discrete mathematics courses are depth-first search, breadth-first search, shortest paths, connected components, finding bipartite sets, and Hamiltonian path/cycle. For weighted graphs, the common problems are the minimum spanning trees, shortest paths, travelling salesman problems. We have created animations for helping instructors and students to teach and learn these algorithms.

The animations are freely accessible from https://yongdanielliang.github.io/animation/animation.html. The animations enable instructors and students to create graphs dynamically, apply the graph algorithms on graphs, and immediately see the results. The animations are effective tools for teaching and learning graph algorithms. The animations have been integrated in the Pearson's interactive REVEL™ ebooks [5, 6, 7], which have received positive reviews [9, 10]. This paper presents the graph algorithm animations for unweighted graphs and for weighted graphs, respectively.

2. SURVEYS OF GRAPH ALOGRITHM ANIMATIONS
Several graph algorithm animations are available on the Web. The most popular graph animations are accessible from http://jhave.org/ [8], https://visualgo.net/en [12] and https://www.cs.usfca.edu/~galles/visualization/Algorithms.html [3]. The first tool [8] is written in Java. Due to security restrictions on Java running on the browser, this tool cannot run from a Web browser. Neither of the tools allows you to create graphs interactively. Because the graphs are pre-created in these tools, the instructor cannot create their own graphs for class demonstration. Our graph algorithm animation tools enable instructors and students to create custom graphs dynamically and easily. Additionally, our tools combine all algorithms for unweighted graphs in one unified animation and all algorithms for weighted graphs in the other unified animation. As a result, it is simple and easy to use our graph algorithm animation tools.

3. ALGORITHM ANIMATION FOR UNWEIGHTED GRAPHS
The unweighted graph algorithm animation tool can be accessed for free from https://yongdanielliang.github.io/animation/web/GraphLearningTool.html, as shown in Figure 3.1. The process of creating a graph is simple. You can add a vertex by clicking the primary button in an open area. You can remove a vertex by clicking the vertex using the secondary button. You can add an edge between two vertices by dragging from one vertex to the other. You can also move a vertex by dragging the vertex while pressing the CTRL button.

After creating a graph, you can apply an algorithm on the graph and see the result of applying the algorithm interactively. To display a

depth-first search tree or breadth-first search tree, specify a starting vertex and click the DFS or the BFS button to see the search tree. For example, as shown in Figure 3.2, a DFS tree is displayed starting from vertex 4. A BFS tree is displayed starting from vertex 4 in Figure 3.3.

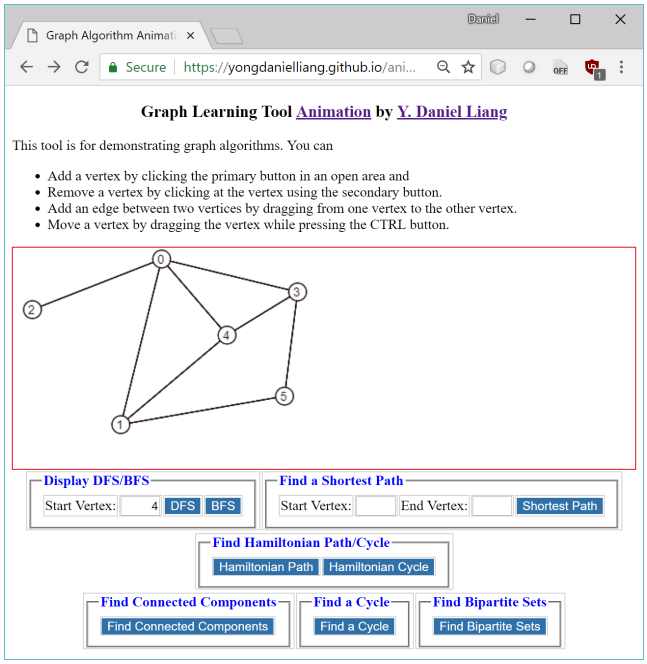


Figure 3.1: The graph algorithm animation for unweighted graphs.

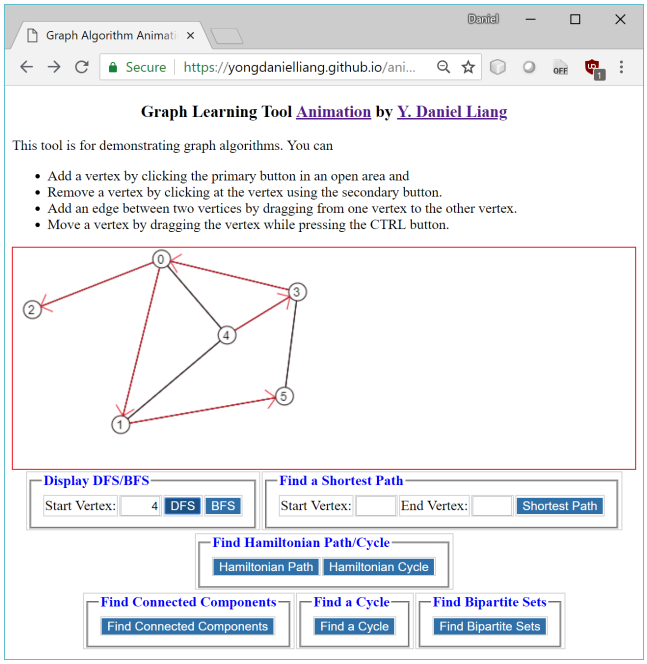


Figure 3.2: A DFS tree starting from vertex 4 is displayed.

The shortest path between two vertices in an unweighted graph can be obtained using the breadth-first search from a vertex. As shown in Figure 3.4, the user enters the starting vertex 2 and the ending vertex 5 and clicks the Shortest Path button to display a shortest path from 2 to 5.

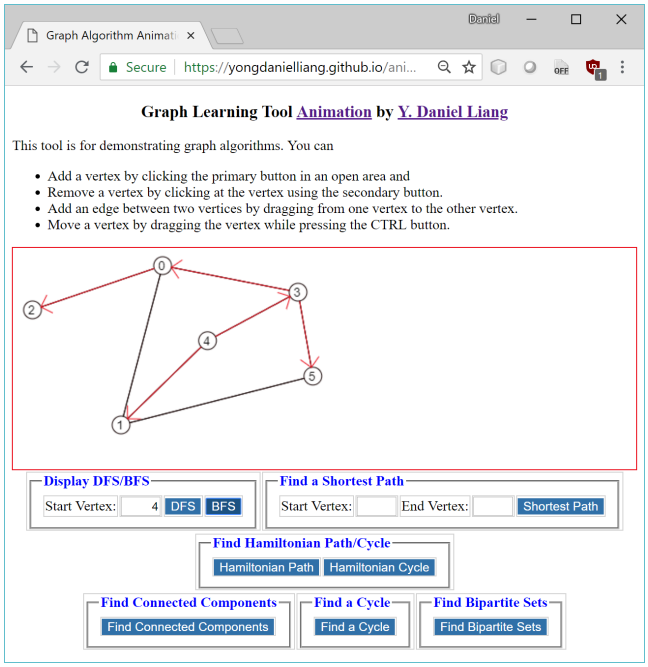


Figure 3.3: A BFS tree starting from vertex 4 is displayed.

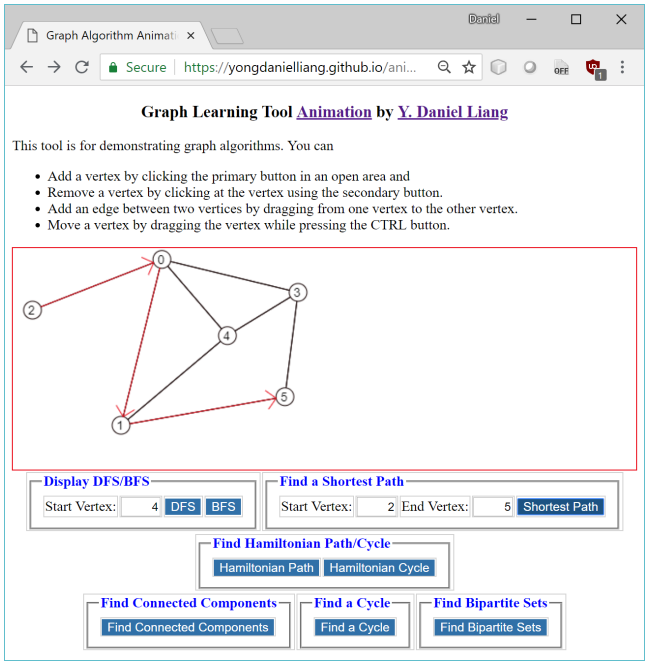


Figure 3.4: The shortest path from vertex 2 to vertex 5 is displayed.

The Hamiltonian path is a path that traverses all vertices in the graph exactly once. As shown in Figure 3.5, clicking the Hamiltonian Path button displays a Hamiltonian path. The Hamiltonian cycle is a Hamiltonian path in which the starting vertex and the ending vertex are connected. As shown in Figure 3.6, clicking the Hamiltonian Cycle button displays a Hamiltonian cycle.

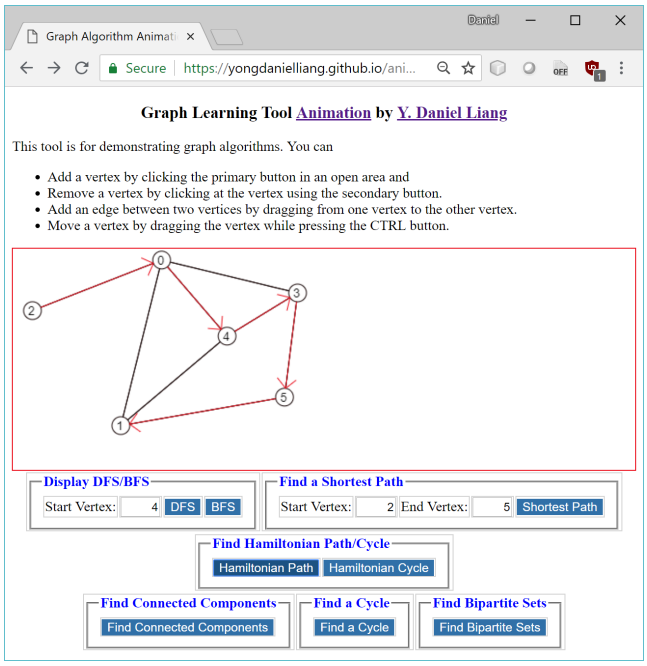


Figure 3.5: The animation displays a Hamiltonian path.

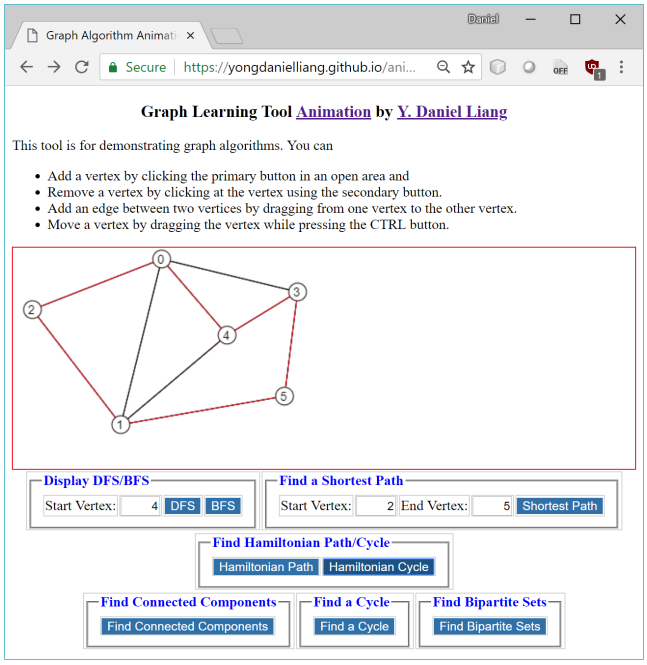


Figure 3.6: The animation displays a Hamiltonian cycle.

A connected component is a maximal connected subgraph in which every two vertices is connected by a path. You can find all connected components in the graph by clicking the Find Connected Components button as shown in Figure 3.7. Two connected components [0, 1, 4, 3, 2] and [5, 7, 6] are displayed for the graph in Figure 3.7.

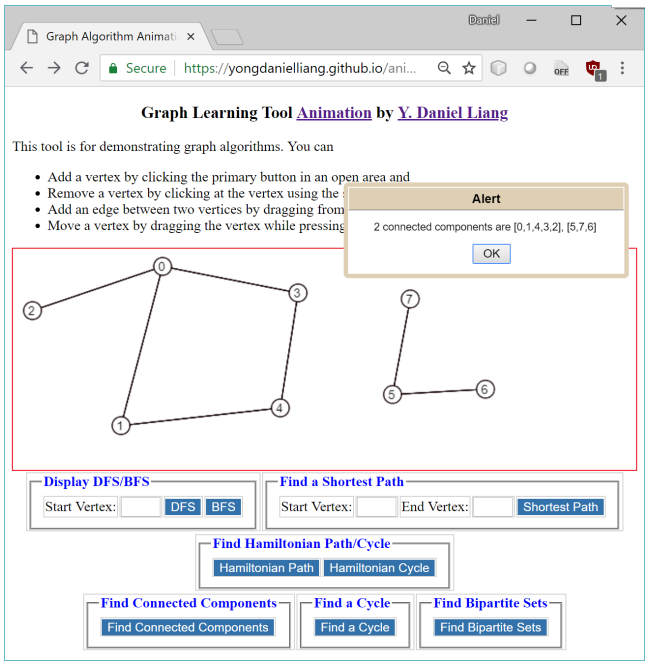


Figure 3.7: The connected components are displayed in the dialog box.

You can find a cycle in the graph by clicking the Find a Cycle button, as shown in Figure 3.8. A cycle 0, 1, 4, 3 is displayed in the dialog box.

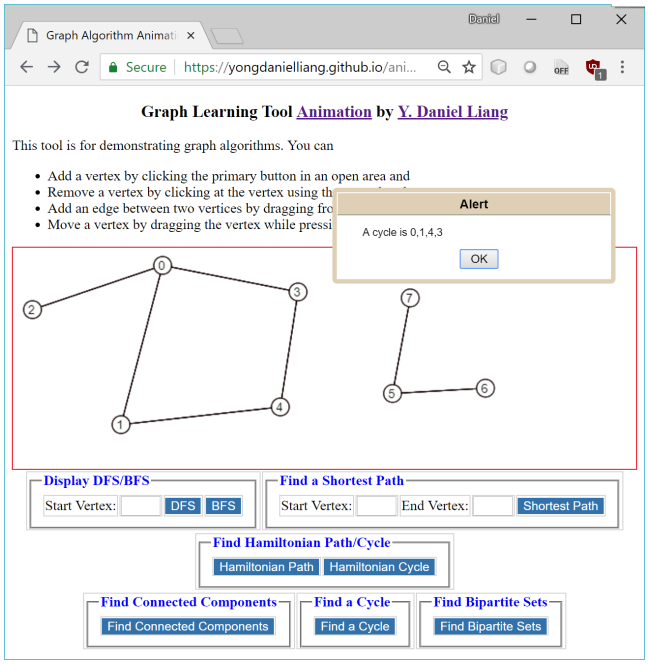


Figure 3.8: A cycle is displayed.

The bipartite sets are the two sets of vertices obtained from the graph such that no vertices in a set is connected. This type of the graph is called a bipartite graph. When you click the Find Bipartite Sets button for the graph in Figure 3.9, the animation displays that the graph is not bipartite, because no bipartite sets can be found for the graph.

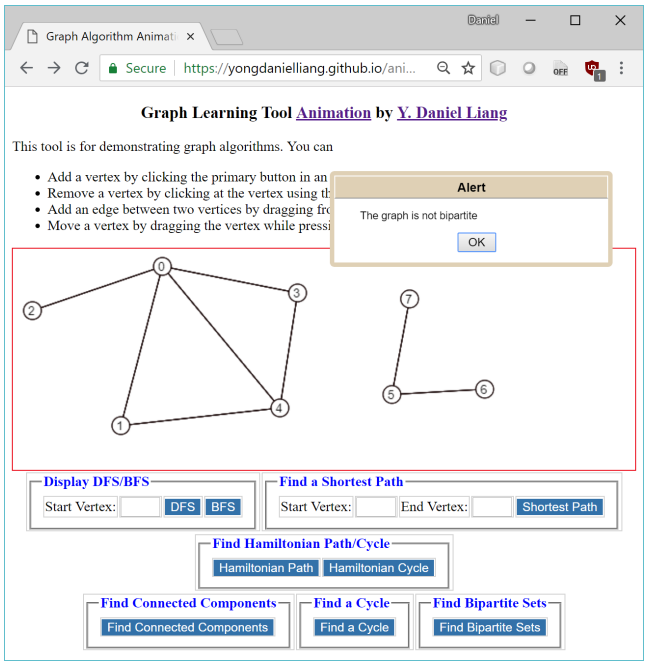


Figure 3.9: The graph is not bipartite.

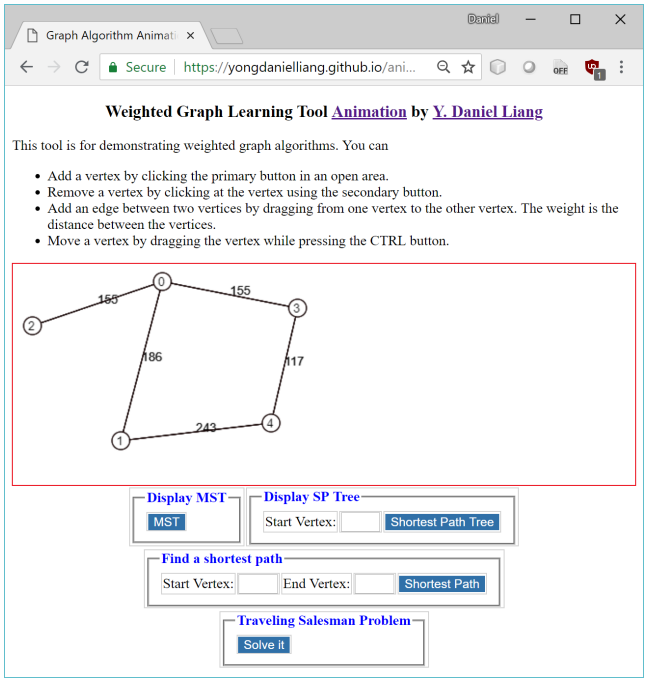


Figure 4.1: The graph algorithm animation for weighted graphs.

4. ALGORITHM ANIMATION FOR WEIGHTED GRAPHS

The weighted graph algorithm animation can be accessed from https://yongdanielliang.github.io/animation/web/WeightedGraphLearningTool.html, as shown in Figure 4.1. The process of creating a weighted graph is similar to creating an unweighted graph. You can add/remove a vertex in the same way as in the unweighted graph algorithm animation. You can add an edge by dragging from one vertex to the other. The weight of the edge is the distance between the two vertices.

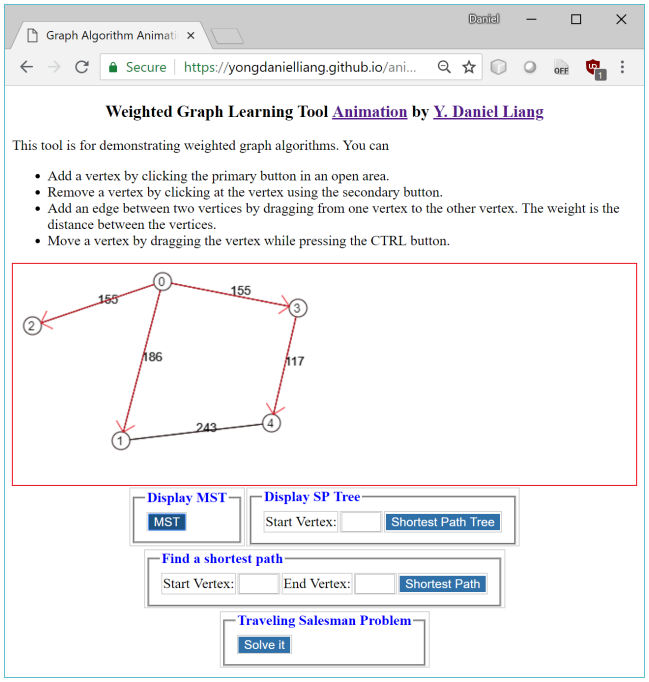


Figure 4.2: A minimum spanning tree is displayed.

A spanning tree of a graph is a connected subgraph that contains all the vertices in the graph and the subgraph is a tree. A minimum spanning tree of a graph is a spanning tree with the minimum total weights. You can obtain the minimum spanning tree by clicking the MST button, as shown in Figure 4.2.

A shortest path tree can be found using the Dijkstra's algorithm. The tree represents a single source all shortest paths. For example, Figure 4.3 shows the shortest path tree starting from vertex 1.

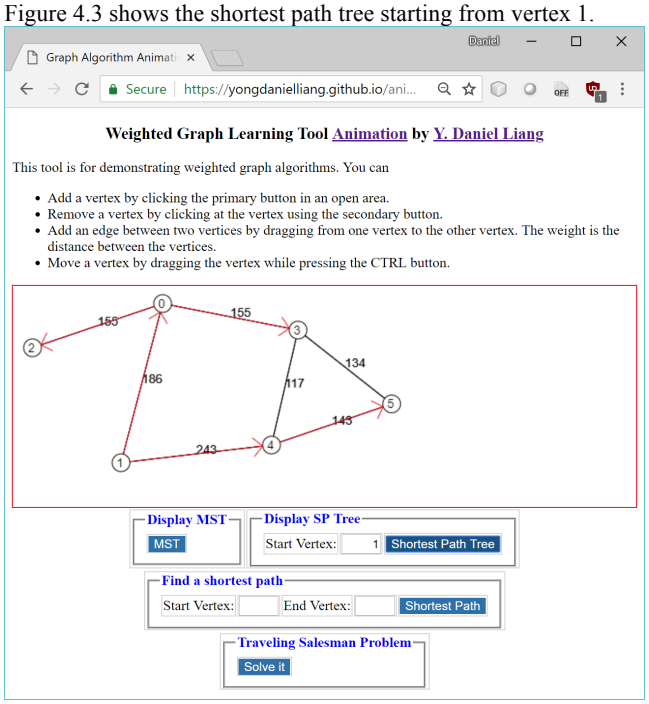


Figure 4.3: A shortest path tree starts from vertex 1.

A shortest path between two vertices can be found after a shortest path tree is constructed. To find a shortest path from vertex u to v, first create a shortest path tree starting from vertex u. A path from u to v in the tree is the shortest path from u to v. For example, Figure 4.4 shows the shortest path from vertex 1 to vertex 5.

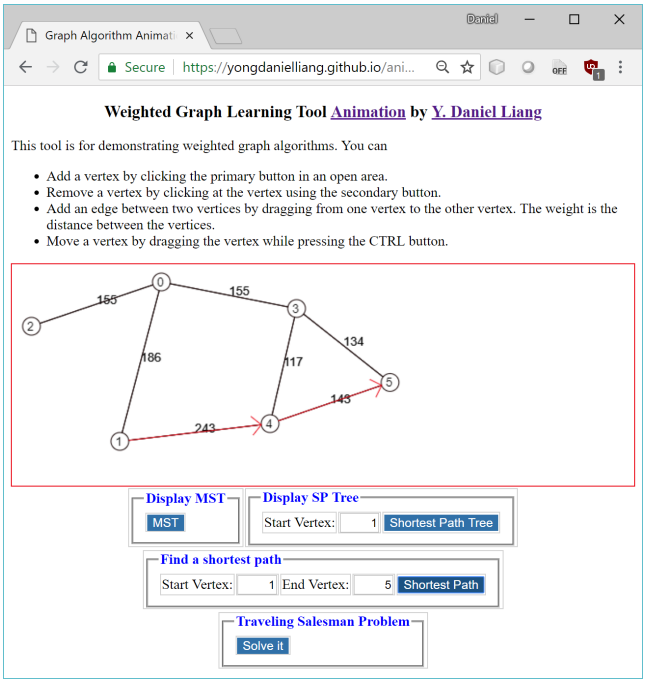


Figure 4.4: A shortest path from vertex 1 to vertex 5 is displayed.

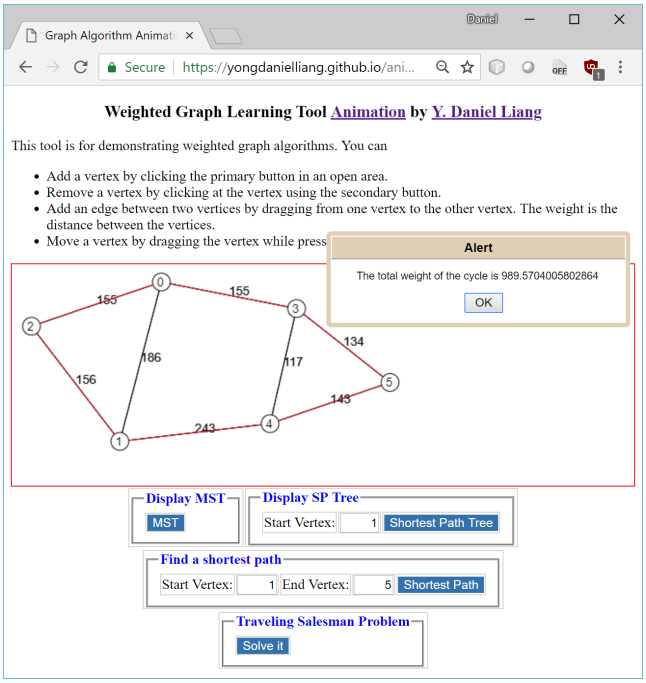


Figure 4.5: A solution for the travelling salesman problem is found with the total weights displayed in the dialog box.

The travelling salesman problem is to find a shortest path that starts from a vertex and visits each vertex exactly once and returns back to the original vertex. Figure 4.5 shows a solution to the problem for the graph in the figure.

5. BENEFITS OF USING GRAPH ALGORITHM ANIMATION

Here are the major benefits for instructors and students to use our tool.

Benefit 1: In a typical lecture for the graph algorithms, the instructor draws various types of graphs on the board and shows the result of applying the algorithms by hand. This is a tedious and time-consuming process. This tool enables the instructor to create a graph dynamically and show the results of applying the algorithm spontaneously. The instructor can draw any type of graph using this tool. The instructor can create vertices anywhere on the screen and can move it to a new location after it is created. The vertices can also be deleted. By dragging the mouse from one vertex to another, an edge between the two vertices can be created. All these interactive features also work on mobile devices.

Benefit 2: Once a graph is created, the tool can show the result of applying the algorithm on the graph interactively. After a graph is modified, with a click of button, the tool can show the new result of applying the algorithm on the new graph. This is tremendously helpful to show students different scenarios.

Benefit 3: A picture is worth a thousand words. An interactive animation is worth more than pictures. The interactive animation not only catches student attention in the class, it also engages the student with visual interaction. Students can use the tool to study before and after the lectures to see how an algorithm works on different graphs.

Benefit 4: Our animation also serves as examples for students to write their own programs to visualize the graph algorithms. This gives students the opportunity to get deeper into the algorithms and see how the algorithms work in their own animation. In our data structures and algorithms courses, we assign projects for students to write their own code for graph algorithm animation. Students like the algorithm animation projects. As supported in [11], students learn better, when they actually implement the algorithms using animation.

6. EVALUATION

Many algorithm animation tools are available. It is safe to say that algorithm animation assists instruction, but whether it helps students to learn is a mixed bag. Some experiments show positive student outcome [1, 8], while others say there are no significant difference to students whether animations are used or not [2]. An experiment conducted at George Washington University [4] showed that the students who used an interactive version of courseware spent more time and performed worse overall than those who used the non-interactive version of the courseware. The reason behind this is that the tools are ineffective and difficult to use. Our goal is to develop a simple tool that is effective and easy to use. First, our tool is free and directly accessible on the Web and

can run on any device from a Web browser. There is no need to install any software. Second, our tool is intuitive. It has only four lines of instructions on how to use it. Third, we combined all the algorithms for unweighted graphs into one application and all the algorithms for the weighted graph into another application, rather to have a separate application for each algorithm. Once a graph is created, the user can apply different algorithms on the same graph.

We use the animation in our data structures and algorithm course in Java. The course covers recursion, Java generics, use of Java collections framework for array lists, linked lists, stacks, queues, priority queues, sets, and maps, implementation of array lists, linked lists, stacks, queues, and priority queues, binary search trees, AVL trees, hashing, and graphs applications for unweighted graphs and weighted graphs. The graph algorithms is a small part in the course, which is covered at the end of the semester. The course is offered every semester.

In the spring of 2015, we conducted a survey for a class of 26 students. The survey has many questions. Two questions related to the graph algorithm animation are the following:

1. Does the graph algorithm animation help you learn graph algorithms? 20 answered yes, 2 answered no, and 4 answered "not sure".
2. Is the graph algorithm animation intuitive and easy to use? All answered yes.

In the fall of 2015, we conducted a second survey for a class of 22 students. This time, we used a scale of 1 to 10 for answers, where 1 is poor and 10 is excellent. The result is as follows:

1. Does the graph algorithm animation help you learn graph algorithms? The average answer is 7.4.
2. Is the graph algorithm animation intuitive and easy to use? The average answer is 9.1.

The survey strongly suggests that the tool is easy to use and helps students learn graph algorithms.

7. IMPLEMENTATION OF THE ANIMATION

The animations are implemented using HTML, CSS, and JavaScript. The user interface is created using HTML. The style is defined in CSS. The user interaction and algorithms are implemented using JavaScript. We define the classes Graph and WeightedGraph to model unweighted graphs and weighted graphs. WeightedGraph is a subtype of Graph. The algorithms such as DFS, BFS, minimum spanning tree, and shortest path are implemented in these classes. The complete code for these classes can be obtained from https://yongdanielliang.github.io/animation/web/Graph.js and https://yongdanielliang.github.io/animation/web/WeightedGraph.js. When the user clicks the right mouse button on the canvas, a new vertex is created. The addVertex method in the Graph class is invoked to add the vertex to the graph. When the user clicks the left mouse button on a vertex, the vertex is removed. The removeVertex method in the Graph class is invoked to remove the vertex from the graph. When the user drags the mouse button from one vertex to another, an edge between the two vertices is created. The program invokes the addEdge method in the Graph class to add the edge to the graph. Each button on the user interface corresponds to an algorithm. For example, when the user clicks the DFS button in Figure 3.2, the program invokes the dfs method in the Graph class to find a depth-first search tree. The tree is then displayed on the canvas in the user interface.

The system is built using a modular approach. An animation for a new algorithm can be easily added by creating a button in the user interface and implementing the algorithm in the Graph class for unweighted graphs or in the WeightedGraph class for weighted graphs.

The source code (HTML, CSS, JavaScript) for the animations can be viewed using the "view page source" function in the browser. With the knowledge of HTML, CSS, JavaScript, and graph algorithms, one can modify the code to add new animations for custom algorithms.

8. LESSONS LEARNED

We started the project to develop the animations for graph algorithms in 2008. Over the years, we have created the animations for many graph algorithms and continuously improved the animation based on the feedback from the students and instructors. There are several lessons learned from developing the animations and from using the animations in classrooms.

- The first lesson learned is to make the animation easy to access. We initially developed the animation using Java applets. Due to security restrictions, many users cannot access the animation. We recreated the animation using HTML, CSS, and JavaScript. The animations now can be viewed anywhere from a browser on a computer and on a mobile device.

- The second lesson learned is to make the animation simple to deploy. We initially developed the animation for each algorithm. We had a total of thirteen animations: an animation for DFS, an animation for BFS, an animation for finding a shortest path, etc. With so many animations, it is difficult to deploy and the user has to click many links to access the animation. Later we combined all the animations into two animations: one for unweighted graph algorithms and the other for the weighted graph algorithms. Now we just need to deploy two animations rather than thirteen separate animations.

- The third lesson learned is to make the animation easy to use. In the early version, the animation lets the user enter the coordinates for each vertex and specify the edges in text boxes in order to create a graph. This proved to be difficult and time-consuming for the user to create a graph. Later we improved it by letting the user use the mouse gestures to add and remove vertices and create the edges. With the mouse gestures, the user can create a graph quickly and easily.

9. FUTURE WORK

We have improved the tool over the years. At present, the tool enables the user to create a graph, apply the algorithm on the graph,

and show the result of applying the algorithm. However, it does not show the intermediate steps to obtain the result. The future work is to expand the animation to show the user the step-by-step procedure for obtaining the results while still retaining the tool's simplicity.

10. CONCLUSIONS

This paper presented graph algorithm animation that is a useful tool for teaching and learning graph algorithms. It enables instructors and students to create custom graphs and see how the graph algorithms work. We developed two animations: one for the unweighted graphs and the other for the weighted graphs. For the unweighted graph, our algorithm animation supports the depth-first search, breadth-first search, shortest path, Hamiltonian path/cycle, finding connected components, finding a cycle, and finding bipartite sets. For the weighted graph, our animation supports minimum spanning trees, shortest path trees, shortest path, and travelling salesman problem. The animations are freely accessible from https://yongdanielliang.github.io/animation/animation.html.

11. ACKNOWLEDGEMENTS

Many thanks to Dr. Steven Gordon and the anonymous reviewers for their careful reading and constructive suggestions for improving the presentation of the paper.

12. REFERENCES

[1] Bazik, J., Tamassia, R., Reiss, S.P., van Dam A., "Software Visualization in Teaching at Brown University," in Software Visualization: Programming as a Multimedia Experience, The MIT Press, pp. 383-398, 1998.

[2] Byrne, M.D., Catrambone, R. and Stasko, J.T., "Do Algorithm Animations Aid Learning?", Tech. Rep. No. GIT-GVU-96-18, Georgia Tech Graphics, Visualization, and Usability Center, 1996.

[3] David Galles, Data Structure Visualizations, https://www.cs.usfca.edu/~galles/visualization/Algorithms.html. Computer Science Department, University of San Francisco.

[4] Jarc, D.J., M.B. Feldman, and R.S. Heller, "Accessing the Benefits of Interactive Prediction Using Web-based Algorithm Animation Courseware," in Proceedings of the ACM SIGCSE Technical Session, Austin, Texas, March 2000.

[5] Liang, Y. Daniel, REVEL™ for Introduction to Java Programming and Data Structures. ISBN-13: 978-0134167008. Pearson Education, 2016.

[6] Liang, Y. Daniel, REVEL™ for Introduction to C++ Programming and Data Structures. ISBN-13: 978-0134669854. Pearson Education, 2018.

[7] Liang, Y. Daniel, REVEL™ for Introduction to Python Programming and Data Structures. ISBN-13: 978-0135187753. Pearson Education, 2018.

[8] Naps, T., Eagan, J, and Norton, L. 2000. "JHAVÉ – An Environment to Actively Engage Students in Web-based Algorithm Visualizations", in Proceedings of the SIGCSE Session, ACM Meetings, Austin, Texas.Visualgo.net, Algorithm and Data Structure Animations, visualgo.net/en.

[9] REVEL™ educator study observes homework and exam grades at University of Louisiana, Spring 2016, http://www.pearsoned.com/results/revel-educator-study-observes-homework-exam-grades-university-louisiana/.

[10] REVEL educator study assesses quiz, exam, and final course grades at Central Michigan University, Fall 2015, http://www.pearsoned.com/results/revel-educator-study-assesses-quiz-exam-final-course-grades-central-michigan-university/.

[11] Stasko, John, "Using Student-Built Algorithm Animations as Learning Aids" in Proceedings of the ACM SIGCSE Technical Session, San Jose, CA., February, 1997.

[12] Visualgo.net, Algorithm and Data Structure Animations, visualgo.net/en.

Identification of Active Oligonucleotide Sequences Using Artificial Neural Network

Alex Luke[1], Sarah Fergione[1], Riley Wilson[1], Brady Gunn[1],
Missouri Western State University,
Department of Chemistry
4525 Downs Drive
St. Joseph, MO 64507
aluke2, sfergione, rwilson26, bgunn1,
@missouriwestern.edu

Stan Svojanovsky[2]
Missouri Western State University
Department of Chemistry
4525 Downs Drive
St. Joseph, MO 64507
ssvojano@missouriwestern.edu

ABSTRACT

In this project we designed an Artificial Neural Network (ANN) computational model to predict the activity of short oligonucleotide sequences (octamers) with important biological role as exonic splicing enhancers (ESE) motifs recognized by human SR protein SC35. Since only active sequences were available from the literature as our initial data set, we generated an additional set of complementary sequences to the original set. We used back-propagation neural network (BPNN) with MATLAB® Neural Network Toolbox™ on our research designated computer. In Stage I of our project we trained, validated and tested the BPNN prototype. We started with 20 samples in the training and 8 samples in the validation sets. Trained and validated BPNN prototype was then used to test the unique set of 10 octamer sequences with 5 active samples and their 5 complementary sequences. The test showed 2 classification errors, one false positive and the other false negative. We used the test data and moved into Stage II of the project. First, we analyzed the initial DNA numerical representation (DNR) and changed the scheme to achieve higher difference between the subsets of active and complementary sequences. We compared the BPNN results with different numbers of nodes in the second hidden layer to optimize model accuracy. To estimate future model performance we needed to test the classifier on newly collected data from another paper. This practical application included the testing of 41 published, non-repeating SC35 ESE motif octamers, together with 41 complementary sequences. The test showed high BPNN accuracy in the predictive power for both (active and inactive) categories.

This study shows the potential for using a BPNN to screen SC35 ESE motif candidates.

Categories and Subject Descriptors

J.3 [Life and Medical Science]: Biology and genetics.

DOI: https://doi.org/10.22369/issn.2153-4136/9/2/4

Keywords

Artificial neural network (ANN), back-propagation neural network (BPNN), nucleotide sequences, exonic splicing enhancers (ESE), DNA numerical representation (DNR).

1. INTRODUCTION

1.1 Artificial Neural Networks (ANN)

Over the past few decades, machine learning processes have become more sophisticated and useful in many different fields of theoretical and applied science, such as applied biology, biomedical research, medicine, and drug discoveries. These methods are based on pattern recognition capabilities [1, 2].

The new and more advanced applications of these models now achieved a major growing momentum.

They are now incorporated in text (spam filtering) and voice recognition (Alexa, Siri and Cortana), virtual video games, self-driving cars, economic forecasting, health related scans and images to reveal any abnormal patterns related to different symptoms and many other fields.

Among other computer-assisted approaches such as machine-learning Decision Trees and Nearest Neighbors algorithms, the ANN–based schemes have gained probably the most attention and are now widely applied.

The initial information (signal) is entering the network of 'neurons' called nodes that is programmed to react to this initial signal and passed the transformed signal to other cluster of nodes so that other signal transformation could be performed. Part of the ANN design is to assign a finite number of these clusters (layers) together with the number of nodes in each layer. The general process of turning the initial input into the output information is the result of ANN program and model design. So, the computer is actually allowed to 'learn' specific information by repeating the very same process, and adjusting the connections intensity between the nodes till the required output is reached. ANNs are then used to solve the problems that are too difficult for both: people and our digital computers. Since these models work on pattern recognition they do not need any underlying data distribution function that is usually required prior to any statistical data analysis and the requirement of data normality before hypotheses testing.

[1] Undergraduate Student

[2] Corresponding Author

1.2 Biological Aspects

Literature [3, 4] and personal communication are the sources of active oligonucleotide sequences (class=1) used in this project.

The authors used the SELEX [5] method to generate a set of sequences with 8 nucleotides (octamers) that were originally evaluated by calculated scores.

Only unique octamers, each with the non-repeating sequence pattern were used in our project. Nucleotide frequencies of a single position of each individual active sequence were then combined into score matrix resulting in an assembly of more general, biologically active SC35 motif [GGCCCCTG] called consensus sequence that we also incorporated into our BPNN model.

These active octamers (also named SC35 ESE motifs) play a major biological role during the process of exon splicing process as exonic splicing enhancers (ESE) that are recognized by human SR protein SC35. This protein is responsible for splicing of another enzyme called pyruvate dehydrogenase (PDH). Any significant deficiency in the process of producing PDH complex is a major cause of lactic acidosis and mental retardation in childhood. SR proteins are involved in proper RNA splicing. They are named SR since this family of proteins is rather conserved and contains many repeats of serine (S) and arginine (R) amino acids [6].

1.3 Goals

The major objective of our project was to apply ANN concept and design the back-propagation neural network (BPNN) on available SC35 ESE motifs. DNA numerical representation (DNR) scheme was then applied to encode the nucleotide bases into numerical values representing each sequence. The set of signals was normalized and partitioned into two major subgroups:

1. training and validation (train+val) subsets
2. testing (test) subset

Both of these subsets contained only unique signals, i.e. none of the test sequences were included in train+val subsets and vice versa.

If the ANN prototype shows high accuracy in sequence classification into active (1) and non-active (0) groups then it might be potentially used as the screening tool for SC35 ESE motifs.

2. METHODS

The very first step was to extract active unique sequences in their letter description format as shown in Tables 1 and 2. It means the sequences of 8 letters combination of A, C, G, and T described as SC35 ECE motif. The letter format represents different types of nucleotides based on their chemical structure and biochemical properties:

A = adenine
C = cytosine
G = guanine
T = thymine

Computer-assisted BPNN is usually considered at least 2-class pattern recognition system with one class representing active (1) feature vectors and the other class holding the non-active (0) feature vectors. In order to satisfy these criteria and make balanced model we generated the matrices with complementary sequences representing non-active output. It is based on the general biological rule that complementary sequences would not fit as SC35 ESE motifs. This process was a part of our BPNN script, so the complementary matrix was computationally generated according to the basic biology principles, where G is the complementary (or antisense) base of C and A is a complement to T.
The conversion could be expressed by G ↔ C and A ↔ T.

We started with 20 active sequences in training and 8 active octamers in the validation set with generated complementary non-active sequences as shown in Tables 1 and 2.

Table 1. Training set of 20 unique sequences

ID	Active (1)	Non-active (0)
1.	GATCCCCG	CTAGGGGC
2.	GGCTCGTG	CCGAGCAC
3.	GGCCGCAG	CCGGCGTC
4.	GGCCCACA	CCGGGTGT
5.	GGTTGGCG	CCAACCGC
6.	GTCCTCCG	CAGGAGGC
7.	GTCCCCTG	CAGGGGAC
8.	GTTCTGTA	CAAGACAT
9.	GAATACCG	CTTATGGC
10.	GGACCGTA	CCTGGCAT
11.	GTCTAACG	CAGATTGC
12.	AGCCTCAG	TCGGAGTC
13.	GGATGGAG	CCTACCTC
14.	GGACTGTA	CCTGACAT
15.	GGTTGTTG	CCAACAAC
16.	GAGCACTG	CTCGTGAC
17.	TGTTACTA	ACAATGAT
18.	GGCTCCAA	CCGAGGTT
19.	GGATCCGG	CCTAGGCC
20.	GACCTGCT	CTGGACGA

Table 2. Validation set of 8 unique sequences

ID	Active (1)	Non-active (0)
1.	GTTTCGAG	CAAAGCTC
2.	GGTCGCCG	CCAGCGGC
3.	GGTCAGTG	CCAGTCAC
4.	GGCTGATG	CCGACTAC

5.	CGCCCTTG	GCGGGAAC
6.	AGCTCCCA	TCGAGGGT
7.	GACCGGTG	CTGGCCAC
8.	GACTAGAA	CTGATCTT

In any machine learning process, DNA sequence are converted to numerical values for data representation and feature learning related to specific biological or biochemical application. The distinct nature of the DNA sequence being discrete in the 'amplitude' and 'time' offers multiple DNA numerical representation (DNR) techniques in the form of single or multidimensional array. Current DNR techniques could be divided into three main categories: single-value mapping, multidimensional sequence mapping, and cumulative sequence mapping [7].

Integer, real number, and measurement representations are still frequently used encoding schemes. In many scenarios of single-value mapping A, C, G, and T are assigned to a single indicator such as 1, 2, 3, and 4. This scheme (also called Galois field) is also feasible for a complementary encoding because it provides symmetric deviations between both groups. Also, it was used in the past for DNA barcode in large-scale screening of multiple genomic core databases. Other direct encoding schemes include Atomic representation, where each nucleotide is assigned its atomic number (i.e. number of protons) [C=58, T=66, A=70, and G=70]. Calculated electron energies for each nucleotide [C=0.1340, T=0.1335, A=0.1260 and G=0.0806] are the core of Electron-Ion Interaction Pseudopotential (EIIP) single-value scheme, while the Molecular Mass encoding is applied in mapping DNA sequences based on molecular mass of different nucleobases [C=110, T=125, A=134, and G=150] with atomic mass units.

Multidimensional sequence mapping include binary sequence indicators such as A=[00], C=[11], G=[10], and T=[01]; 4-bit representation with A=[1000], C=[0100], G=[0010], and T=[0001].

Cumulative representation include Z-curve, DNA walk and other more complex DNA encoding schemes.

Currently, no DNR is considered to be the 'gold standard' and the choice is usually driven by the applicable biological aspects and the specific goals of the machine learning project.

We selected direct, single-mapping Galois field encoding method because it provides uniform distance between active and non-active (complementary) sequences with symmetric deviations. Other advantage is to use simple barcode method to label each sequence for automated sequence screening. It also supports our biological goals of the project to separate the signals for active and non-active octamers. However, this structure might imply that pyrimidines (C and T) are in some respect 'greater than' purines (A and G), which is a disadvantage of this encoding method.

Table 3 represents 10 octamers that we used to test BPNN model. This is a data set of unique sequences with known activities. Five of them are active and five of them are from the non-active group. None of these sequences were previously used in training and validation subset. Active samples (1, 4, 5 and 10) are from the published article [3], sample 9 was added based on the private communication [Luke, personal communication].

Table 3. Testing subset of 10 unique sequences

ID	Designation	Class
1.	GATCGCTG	1
2.	AGTCGGAT	0
3.	CTCATTGC	0
4.	GGCCCCTG	1
5.	GGACGCTG	1
6.	CCGGGGAC	0
7.	CTAGCGAC	0
8.	CCTGCGAC	0
9.	TCAGCCTA	1
10.	GAGTAACG	1

Figure 1 shows the general ANN based on one-layer hidden units, where all nodes have the same number of weights (synapses) and all receive the input signal simultaneously.

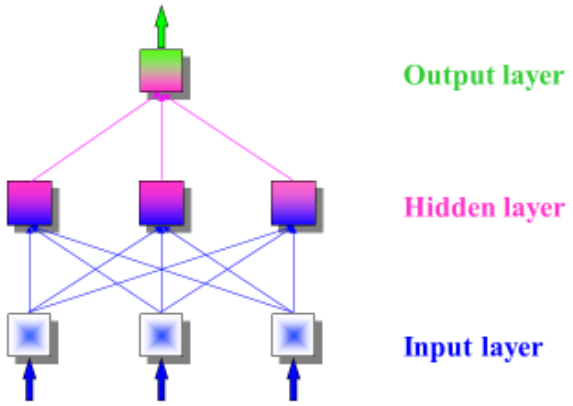
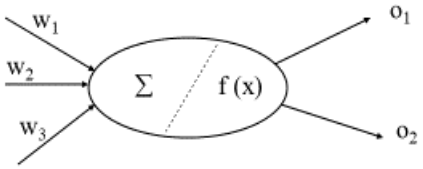


Figure 1. General assembly of neural network processing

Action of formal neuron (node) consists in summing all weighted inputs (w_i) transformed via activation function into output signals (o_j). BPNN default is the sigmoid function.



$$f(x) = 1/[1+\exp(-x)]$$

Figure 2. ANN node action with sigmoid transformation function

3. RESULTS AND DISCUSSION

3.1 Project Stage I

The initial step in BPNN design was to generate Galois field numerical encoding for A = 1, T = 2, C = 3, and G = 4.
Active sequences were added into BPNN MATLAB script with activity equal to 1. The next part of the script generated the complementary, non-active sequences that were used to balance the BPNN model. All data went through the normalization into [0, 1] interval across each feature matrix.

In this project we used a supervised training where both the input signal and the output activity are provided. The network transforms the inputs with connection weights through the nodes and layers and calculate the errors between the resulting and desired outputs. Errors are then propagated back through the network to adjust the weights which control the network assembly. During this learning process the training data set is processed many times as the connection weights are continually adjusted and finally refined.

Validation process that is parallel to training enables to validate the final model specification with the validation data set. The model is trained on the training set and the error is calculated on the validation set multiple times while adjusting the weights. It is used to analyze the value of parameters in the model which usually results in less error on validation set.

Testing provides then an unbiased evaluation of a final model fit on the training dataset.

For our BPNN model we used the seed for the random number generator applied for the initial weights to be equal 1.

BPNN component was applied with multiple variables:

- Convergence error (SSE): usually about 0.0001
- Number of iterations: 100

Samples not previously included in training process were used for the validation.
Finally, we tested the BPNN classifier with test data set (i.e. 10 unique octamers with known output 0 or 1) in specific model conditions with 8 nodes in the first layer and 6 nodes in the second hidden layer.

Table 4. BPNN classification of test sequences

ID	Letter Designation	Known classification	BPNN classification
1.	GATCGCTG	1	0.9935
2.	AGTCGGAT	**0**	**0.8843**
3.	CTCATTGC	0	0.0046
4.	GGCCCCTG	1	0.9934
5.	GGACGCTG	1	0.9949
6.	CCGGGGAC	0	0.0028
7.	CTAGCGAC	0	0.0043
8.	CCTGCGAC	0	0.0044
9.	TCAGCCTA	**1**	**0.0027**
10.	GAGTAACG	1	0.9958
Training error			0.0144
Validation1 error			0.9769
Validation0 error			0.9821
Testing error			2

Classification with the BPNN model under the specific conditions revealed 2 errors. Non-active sample 2 was predicted to be active (false positive), while the active sequence 9 was misplaced by BPNN model into the cluster of non-active sequences (false negative). We were not satisfied with model performance and moved into Stage II of the project.

3.2 Project Stage II

We started this stage with graphical interpretation of active and non-active feature vectors that provided the partial key to the problem. Our computer script generated the matrix of complementary (non-active) sequences based on the given instructions with the application of the existing biology rules. Our complementary sequences were generated with the absolute difference of 1 between nucleobases A and T and C and G.

$$| A - T | = | 1 - 2 | = | C - G | = | 3 - 4 | = 1$$

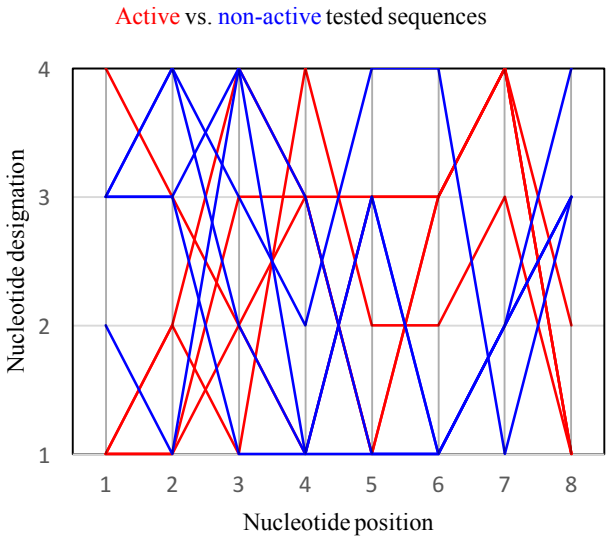
The data analysis of the initial train+val sequence subsets showed that the majority of the active sequences (86%) started with the first nucleotide G (C for complementary sequences). All tested sequences starting with G or C were then correctly classified by BPNN model. However, in the initial train+val subsets we also had total of 3 sequences starting with A nucleotide (2 active and 1 non-active). The active sequences started with AG and non-active AC dinucleotide. Size limitation of the training set could be the potential reason of the lower performance of our BPNN model resulting in misclassification of 2 tested sequences.

In attempt to reduce misclassification error we applied higher resolution between the active and non-active categories to further separate both of these subsets in their space. We went back and changed the initial single-value DNR scheme to achieve higher and constant difference between both groups.

If A = 2, T = 4, C = 3, and G = 1, then

$$| A - T | = | 2 - 4 | = \mathbf{2} \text{ and } | C - G | = | 3 - 1 | = \mathbf{2}$$

We also created a 2-D distribution chart to differentiate between active and non-active categories. Graph 1 displayed a complete overlap of both groups at position 3 and some partial overlaps at positions 2, 4, and 5, respectively.



Graph 1. Distribution of 10 tested sequences

In the following step of Stage II we tried different numbers of nodes in the second hidden layer in order to find an optimal estimate.
The results are summarized in Table 5 together with training, validation, and test errors.

Table 5. BPNN outputs for tested sequences with variable number of nodes in the second hidden layer

ID	Class	# nodes 1	# nodes 2	# nodes 3	# nodes 4
1.	1	0.9946	0.9964	0.9963	0.9970
2.	0	0.0589	0.1048	0.0371	0.0291
3.	0	0.0091	0.0064	0.0014	0.0033
4.	1	0.9948	0.9966	0.9970	0.9967
5.	1	0.9940	0.9963	0.9967	0.9964
6.	0	0.0075	0.0043	0.0013	0.0023
7.	0	0.0077	0.0050	0.0013	0.0028
8.	0	0.0084	0.0045	0.0013	0.0035
9.	1	**0.0094**	**0.0051**	**0.0014**	**0.0058**
10.	1	0.9930	0.9971	0.9967	0.9969
Training error		0.0292	0.0104	0.0125	0.0113
Validation1 error		0.0086	0.0054	0.0063	0.0048
Validation0 error		0.0163	0.0114	0.0020	0.0082
Testing error		1	1	1	1
ID	**Class**	**# nodes 5**	**# nodes 6**	**# nodes 7**	**# nodes 8**
1.	1	0.9962	0.9969	0.9891	0.9951
2.	0	0.3789	0.0129	0.4544	0.0792
3.	0	0.0070	0.0042	0.0048	0.0051
4.	1	0.9972	0.9968	0.9894	0.9955
5.	1	0.9966	0.9971	0.9894	0.9951
6.	0	0.0040	0.0043	0.0044	0.0053
7.	0	0.0042	0.0041	0.0045	0.0048
8.	0	0.0049	0.0044	0.0055	0.0059
9.	1	**0.0043**	**0.0047**	**0.0057**	**0.0070**
10.	1	0.9974	0.9963	0.9896	0.9948
Training error		0.0141	0.0084	0.0170	0.0216
Validation1 error		0.0062	0.0046	0.0125	0.0082
Validation0 error		0.0087	0.0078	0.0121	0.0141
Testing error		1	1	1	1

Based on calculated training, validation, testing errors and the BPNN overall performance, the optimal estimate is represented by 6 nodes in the second hidden layer.

Variables of the optimal BPNN prototype:

- Convergence error (SSE): **0.0001**
- Number of iterations: **100**
- Number of nodes in the first layer: **8**
- Number of nodes in second (hidden) layer: **6**

3.3 Testing larger database

We used Stage I test data to initiate Stage II and to optimize the number of nodes in the second hidden layer, so the test performance is likely an optimal estimate. To evaluate future performance, we needed to test the classifier on newly collected data from another paper [6]. The authors provided the list of 128 active SC35 ESE motif sequences specifically arranged by different tissues, genes, and selected organs. They proposed highly conserved SC35 motif between tissues, among different genes, and within the same chromosome. They showed a slight variation in the SC35 ESE sequence motif among human chromosomes, with the conserved G nucleotide at the very first position of all active sequences.

The set included multiple sequence duplicates as they occurred in several tissues and various genes, and chromosomes. Prior to the test we removed all duplicates (87 sequences) and used the total of 41 unique active sequences together with 41 complementary non-active sequences with our optimal BPNN classifier. Again, none of these tested sequences were included in our BPNN train+val sets.

Model classification, together with training, validation and test errors are summarized in Table 6.

Table 6. Prediction for 41 active and complementary sequences with the optimal BPNN model

ID	Class (1)	BPNN	Class (0)	BPNN
1.	GACCCCTG	0.9917	CTGGGGAC	0.0039
2.	GACCTCTG	0.9916	CTGGAGAC	0.0034
3.	GACCACTG	0.9917	CTGGTGAC	0.0027
4.	GATCACTG	0.9920	CTAGTGAC	0.0033
5.	GATCCCTG	0.9922	CTAGGGAC	0.0050
6.	GGCCCCTG	0.9922	CCGGGGAC	0.0053
7.	GGCTCCTG	0.9920	CCGAGGAC	0.0122
8.	GACTCCTG	0.9920	CTGAGGAC	0.0058
9.	GACTCCCG	0.9917	CTGAGGGC	0.0048
10.	GACCCCCG	0.9917	CTGGGGGC	0.0035

11.	GACCACCG	0.9922	CTGGTGGC	0.0025
12.	GGCCCCCG	0.9913	CCGGGGGC	0.0046
13.	GGCCTCTA	0.9921	CCGGAGAT	0.0032
14.	GGCCTCTG	0.9913	CCGGAGAC	0.0047
15.	GGCCTCCA	0.9921	CCGGAGGT	0.0029
16.	GGCCTCCG	0.9915	CCGGAGGC	0.0041
17.	GGCCCCTA	0.9907	CCGGGGAT	0.0036
18.	GTCTCCTG	0.9888	CAGAGGAC	0.0433
19.	GTCCCCTA	0.9923	CAGGGGAT	0.0090
20.	GGCTCCAG	0.9922	CCGAGGTC	0.0205
21.	GGCCCCAG	0.9915	CCGGGGTC	0.0068
22.	GGCCCCCA	0.9924	CCGGGGGT	0.0032
23.	GGCTACTG	0.9920	CCGATGAC	0.0121
24.	GGCTTCTG	0.9925	CCGAAGAC	0.0119
25.	GGCTGCTG	0.9922	CCGACGAC	0.0118
26.	GGCCACTG	0.9922	CCGGTGAC	0.0039
27.	GGCCGCTG	0.9922	CCGGCGAC	0.0042
28.	GGCTCCTA	0.9916	CCGAGGAT	0.0057
29.	GGCTCCCG	0.9923	CCGAGGGC	0.0093
30.	GGCTCCCA	0.9917	CCGAGGGT	0.0047
31.	GACTCCCA	0.9912	CTGAGGGT	0.0032
32.	GATTTCCG	0.9921	CTAAAGGC	0.0059
33.	GATTCCCG	0.9923	CTAAGGGC	0.0065
34.	GACTTCCG	0.9917	CTGAAGGC	0.0044
35.	GACCTCCG	0.9916	CTGGAGGC	0.0031
36.	GACCTCCA	0.9904	CTGGAGGT	0.0024
37.	GACCTCTA	0.9904	CTGGAGAT	0.0026
38.	GACCCCCA	0.9906	CTGGGGGT	0.0027
39.	GACCCCTA	0.9907	CTGGGGAT	0.0029
40.	GACTTCTG	0.9916	CTGAAGAC	0.0052
41.	GGCCTCAG	0.9920	CCGGAGTC	0.0063
Training error			0.0205	
Validation1 error			0.0107	
Validation0 error			0.8008	
Testing error			0	

The test confirmed that the BPNN prototype satisfactory distinguishes between all 41 proposed SC35 ESE active motifs and their compliments with high accuracy in BPNN classification performance.

4. CONCLUSION

In our research project we used ANN script to construct a functional back-propagation neural network (BPNN) model. We designed this model in order to classify the short oligonucleotide sequences with 8 nucleotide elements (octamers) into two categories: active (1) and non-active (0) clusters. The visual interpretation of the data (Graph 1) shows some partial overlaps of both groups on multiple feature vector elements, which supports our decision to apply neural network concept. Statistical data analysis requires a prior knowledge of data distribution, which could be very complex in case of any overlap. Also, all elements of the feature vector are discrete values in relatively small data set which will most likely require non-parametric statistical analysis.

We used single-value scheme to encode sequence letter description into numerical designation. The model was trained with 20 active sequences and validated with the set of 8 active sequences. In order to keep the model balanced the complementary, non-active sequences were generated. The initial virtual screen included 10 unique sequences from the testing data set (5 active and 5 non-active sequences) used to assess the model accuracy and overall performance. After the BPNN model update we tried different number of nodes (1-8) in the second hidden layer to determine the optimal model.

We tested our optimal BPNN prototype on larger data set of 82 unique (41 active and 41 non-active) sequences and the results of the data classification revealed high model accuracy for this data set.

5. FUTURE WORK

For future work we could test any proposed CS35 ESE motif candidate or use the BPNN prototype to screen any sequence database for a potential match. We might also draw random biological sequences that are not known to be SC35 ESE motif candidates and detect how many of them are classified by BPNN as active.

The initial published data were listed with their scores that were calculated using a score matrix. Another type of future work would be to incorporate this information into our model, i.e. not just to classify the data into active and non-active subsets but add some degree to the activity and answer the question: "If active, how much activity is predicted?"

Also, it would be beneficial to create and compare additional classification prototypes based on different DNA numerical representation (DNR) methods such as binary indicators and OneHot Encoder and additional classification procedures such as decision trees or k-nearest neighbor algorithm.

6. REFLECTIONS

The project described in this paper was the very first research project for all undergraduate students in my research group. They all actively participated on this project as each of them designed their own ANN model. The major attraction for all students was the introduction of artificial intelligence in the computer-assisted model and the practical application of the BPNN prototype on real SC35 ESE motif sequences.

This project provided the students with multiple opportunities to participate on each stage of the project, starting with the literature research, learning the basics of MATLAB computing together with Neural Network Toolbox, join the time consuming journey to design the proper ANN model through the training, validation, and testing procedures. They were all rather skeptical after the Stage I about the real possibility to enhance model 80% accuracy. The first run after model update in Stage II showing improved to

90% accuracy on small tested data was accepted with contagious joy and new motivation to continue and apply BPNN prototype on larger data set. I know that during this project all students learned many invaluable skills that they could apply to their future education or work. They all have a better understanding of the advantages of applied neural network models as well as the limitation of such models. Students also used this research opportunity and presented their work during all project stages in multiple forums, including poster and oral presentations at local, state and national conferences. Their poster was accepted for an oral presentation on ACS National Meeting & Exposition, as well as on ASBMB National Meeting.

7. ACKNOWLEDGMENTS

We would like to thank Dr. Swapan Chakrabarti, Department of Electrical Engineering and Computer Science, University of Kansas, Lawrence KS 66045 for his expertise and support.

This work was supported by PORTAL, The Program of Research, Teaching, and Applied Learning at Missouri Western State University, St. Joseph, MO 64507.

8. REFERENCES

[1] Chakrabarti S., Svojanovsky S., Slavik R., Georg G. I., Wilson G. S., and Smith P. G. 2009. Artificial Neural Network Based Analysis of High-Throughput Screening Data for Improved Prediction of Active Compounds. *J. Biomol. Screen* 2009 Dec; 14 (10):1236-44

DOI: 10.1177/1087057109351312

[2] Oyedotun O.K., and Khashman A., 2017 Prototype-Incorporated Emotional Neural Network. *IEEE Trans Neural Netw Learn Syst.* 2017 Aug 15. PMID: 28816677

DOI:10.1109/TNNLS.2017.2730179

[3] Liu, H-X., Chew S. L., Cartegni L., Zhang M. Q., and Krainer A. R. 2000. Exonic Splicing Enhancer Motif Recognized by Human SC35 under Splicing Conditions *Mol. Cel. Biol.* 20 (Feb 2000), 1063-1071. PMID: 10629063

[4] Siala, O., et al., 2014. Slight variations in the SC35 ESE sequence motif among human chromosomes: a computational approach, Gene (2014), http://dx.doi.org/10.1016/j.gene.2014.04.075

[5] Kim, Soyoun, Shi, Hua, Lee, Dong-kee, and Lis, John T. 2003. Specific SR protein-dependent splicing substrates identified through genomic SELEX. *Nucleic Acids Res*. 31, 7 (Feb. 2003), 1955-61

[6] Shepard, Peter J. and Hertel, Klemens J. 2009. The SR Protein Family. Genome Biol. 242, 10 (Oct. 2009), 242.1-242.9.

[7] Gerardo Mendizabal-Ruiz, Israel Román-Godínez, Sulema Torres-Ramos, Ricardo A. Salido-Ruiz, J. Alejandro Morales, 2017, 'On DNA numerical representations for genomic similarity computation', (2017) PLOS ONE, vol. 12, no. 3, p. e0173288 https://doi.org/10.1371/journal.pone.0173288

Parsing Next Generation Sequencing Data in Parallel Environments for Downstream Genetic Variation Analysis

Mariana Vasquez
Bioinformatics Program,
The University of Texas at El Paso,
El Paso, TX 79968
mvasquez16@miners.utep.edu

Jonathon Mohl
Border Biomedical Research Center
and Computational Science Program,
The University of Texas at El Paso,
El Paso, TX 79968
jemohl@utep.edu

Ming-Ying Leung
Bioinformatics Program,
Border Biomedical Research Center,
Computational Science Program, and
Department of Mathematical Sciences,
The University of Texas at El Paso,
El Paso, TX 79968
mleung@utep.edu

ABSTRACT

With the recent advances in next generation sequencing technology, analysis of prevalent DNA sequence variants from patients with a particular disease has become an important tool for understanding the associations between the disease and genetic mutations. A publicly accessible bioinformatics pipeline, called OncoMiner (http://oncominer.utep.edu), was implemented in 2016 to help biomedical researchers analyze large genomic datasets from patients with cancer. However, the current version of OncoMiner can only accept input files with a highly specific format for sequence variant description. In order to handle data from a broader range of sequencing platforms, a data preprocessing tool is necessary. We have therefore implemented the OncoMiner Preprocessing (OP) program for parsing data files in the popular FastQ and BAM formats to generate an OncoMiner input file. OP involves using the open source Bowtie2 and SAMtools software, followed by a python script we developed for genetic sequence variant identification. To preprocess very large datasets efficiently, the OP program has been parallelized on two local computers and the Blue Waters system at the National Center for Supercomputing Applications using a multiprocessing approach. Although reasonable parallelization efficiency has been obtained on the local computers, the OP program's speedup on Blue Waters has been limited, possibly due to I/O issues and individual node memory constraints. Despite these, Blue Waters has provided the necessary resources to process 35 datasets from patients with acute myeloid leukemia and demonstrated significant correlation of OP runtimes with the BAM input size and chromosome diversity.

Keywords

Next generation sequencing, Genetic sequence variants, Cancer, OncoMiner pipeline, Data preprocessing, Acute myeloid leukemia, High performance computing, Blue Waters, Multiprocessing, Python scripts.

1 INTRODUCTION

Many serious diseases, such as cancer [1], cystic fibrosis [2] and multiple sclerosis [3], are often linked to genetic mutations in the human genome. Identification of mutations prevalent in individuals with a given illness helps establish associations between the mutations and the disease. This is exemplified in acute myeloid leukemia (AML) with the genes DNMT3A, ASKL1, TET2, IDH1 and IDH2 linked to early disease progression [1]. With the recent advances in next generation sequencing (NGS), genomic sequences of an increasing number of cases have been made available. These data have enabled scientists to perform detailed data analyses to look for associations between diseases and DNA mutations, called genetic sequence variants (GSVs).

OncoMiner [4] (http://oncominer.utep.edu) is a publicly accessible bioinformatics pipeline developed at the University of Texas at El Paso (UTEP) for analyzing large genomic datasets from patients with cancer. OncoMiner's functionalities include linking GSVs with published research literature, visualization of their chromosomal locations, and performing statistical comparisons of their occurrence frequencies among different groups of subjects. As OncoMiner was originally developed for analyzing the GSVs from NGS and GSV identification services provided by Otogenetics Corporation, the input files for the OncoMiner pipeline were restricted to a particular format with a set of specific terms describing the type and location of each GSV. In order to utilize OncoMiner on more general datasets coming from other NGS platforms (e.g., the Illumina NextSeq sequencer in the UTEP Genomic Analysis Core Facility), data preprocessing needs to be performed in order to provide an input file that can be passed to OncoMiner for GSV analysis.

The purpose of this project was to develop an efficient program to preprocess NGS data, extract the necessary information and write it in a suitable format to be inputted to OncoMiner for further analysis. NGS data files are typically large, in the order of tens of gigabytes (GBs) and downstream analysis in OncoMiner usually involves multiple samples from the cancer group and the control group. Serial preprocessing of such datasets on our local computers would take excessive amount of time to complete all the tasks. A high performance computing system such as Blue

Waters that allow multiple samples to be processed simultaneously would be essential.

In this paper, we describe the implementation of a program to preprocess NGS data files in the popular FastQ and BAM formats, converting them to files in csv format that can be inputted to the OncoMiner pipeline on three parallel computing platforms. Some background information about NGS data and the OncoMiner input requirements is given in the next section. The Methods section describes the steps taken to complete the data preprocessing using a multiprocessing approach. This preprocessing program is tested with a collection of 35 datasets from patients with AML. The resulting runtimes, speedup, and efficiencies are presented in Section 4, where we also discuss various issues encountered during the parallelization process. The conclusion and our ongoing investigations are given in section 5.

2 BACKGROUND

Human genetic information is stored in DNA molecules contained inside 23 pairs of chromosomes, designated chromosomes 1, 2,…, 22, and X. Each chromosome contains a DNA molecule represented by a very long string of letters from the four-letter alphabet denoting the nucleotide bases adenine (A), cytosine (C), guanine (G), and thymine (T). The lengths of human DNA molecules are in the range of approximately 48 million - 250 million nucleotides. DNA has a double stranded, antiparallel structure. One end of each strand is labeled 5' and the other labeled 3'. Genetic information in DNA can be found on either strand and is always read from the 5' end to 3' end.

There are three common types of point mutations that cause genetic information changes: (1) substitution occurs when a nucleotide is substituted by another; (2) insertions are "extra" nucleotides inserted into the sequence; (3) deletions are the removal of nucleotides. An example of each type is shown in Figure 1. The top line is the reference sequence. A substitution of nucleotide "C" by "A" occurs at position 7 of line 2. In line 3, the nucleotide "A" is inserted after position 25. The four nucleotides "AACC" at position 13 – 16 are deleted in line 4.

```
ATCGGGCCAAAAAACCCCCGCGCGCGAAAAATTTTT  Ref Sequence
ATCGGGACAAAAAACCCCCGCGCGCGAAAAATTTTT  Substitution
ATCGGGCCAAAAAACCCCCGCGCGACGAAAAATTTTT Insertion
ATCGGGCCAAAA____CCCGCGCGCGAAAAATTTTT  Deletion
```

Figure 1. Three types of point mutations in DNA: substitution, insertion, and deletion.

A gene comprises multiple segments of DNA that are necessary to transcribe and translate encoded genetic information into a protein. DNA is transcribed into RNA containing both exons and introns initially. During the process of RNA splicing, introns are removed and the exons are joined to form a continuous, mature mRNA. Except for the small stretch of nucleotides at the 5' end and the 3' end of the mRNA, the rest of the transcript form the coding sequence (CDS) that will be translated into a protein. A description of the organization of DNA transcriptional elements can be found at http://www.scfbio-iitd.res.in/research/orf.html. GSVs within the CDS of a gene can either be synonymous or non-synonymous. Synonymous variants do not change the resulting protein, but non-synonymous variants do. Non-synonymous GSVs can directly affect biological functions and are generally of greater biomedical concern.

NGS is a high-throughput technology for DNA sequencing. It allows for large amounts of sequences to be obtained much faster and cheaper than the traditional Sanger sequencing procedure that produces one sequence at a time. Some NGS platforms can generate up to 20 billion reads or 6 TB of data per run. The results are often stored in FastQ format (Figure 2A), which contains nucleotide sequences and their respective sequencing quality scores. Programs like Bowtie2 [5] and Burrows-Wheeler Aligner (BWA) [6] align the sequences obtained by NGS to a reference human genome and store the results in a sequence alignment map (SAM) file or its binary equivalent BAM file. The SAM format (Figure 2B) contains information about the location and nature of the differences from the reference sequence, and quality scores among other things.

Currently, a number of open-source programs are available for GSV identification and analysis. For example, ANNOVAR [7] is a tool for ANNotation Of genetic VARiants. VEP (Variant Effect Predictor) uses different scoring schemes to evaluate

A

```
@NS500477:4:HHFYGBGX2:1:11101:7339:1083 1:N:0:9
GTTACNATCATCTTGTTTCTTGAGATAGAGCATATGGGCCTATAAAAAAAATTTAAAAAATAAAATTCCACTATTC
+
AAAAA#EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEAEEEEEEEEEEAEEEEEEEAEEEEAEEEEEEEEA
@NS500477:4:HHFYGBGX2:1:11101:3189:1084 1:N:0:9
CGGATNCCTTCTCACTGTACACAGCTGCCTGGCTCCCTCTGGTCCCTGCTTCAGAGGAGGCCTGGGCCTGTGGCC
+
A/AAA#EEEEEEEE/EEEEEE<AEEAEEAE/EAE<EEAEEEAEEEEEEAAEEA/E//EAEAEEE/<EEEEA/EEE
@NS500477:4:HHFYGBGX2:1:11101:9780:1084 1:N:0:9
```

B

```
K00211:113:H52VJBBXX:8:2115:30107:30204   83   chr1 21521 1   100M =    21502 -119 TCCTGTCTTGAGTAATCCGTGGGCCCTAACTCACTCATCCCAACTCTTCACTCACTGCCC
K00211:113:H52VJBBXX:8:2113:22252:2809    147  chr1 21574 1   100M =    21508 -166 ACTGCCCTGCCCCACACCCTGCCAGGGAGCCTCCCGTGGCACCGTGGGGACACAAAGGAA
K00211:113:H52VJBBXX:8:1114:4827:28147    99   chr1 21679 0   92M5D8M   =    21755 181  AGGCTCACGGAAAGTCAGCCTCTCATGCCCCGAGAGCTGAGTGCAAGGGAGAGG
K00211:113:H52VJBBXX:8:1109:30401:16682   163  chr1 21740 1   100M =    21774 134  GTCTGTGCTTCCCATGCAGAAGCACCCCCCTCCCACCCCTGTGCAGTCCGGCCTTCGCGG
K00211:113:H52VJBBXX:8:1123:14478:41739   163  chr1 21750 1   100M =    21845 195  CCCATGCAGAAGCACCCCCCTCCCACCCCTGTGCAGTCCGGCCTTCGCGGCAGACCACCA
K00211:113:H52VJBBXX:8:1114:4827:28147    147  chr1 21755 0   16M5D84M  =    21679 -181 GCAGAAGCACCCCCCTCCCTGTGCAGGCCGGCCTTCGCGGCAGACCACCATACAC
K00211:113:H52VJBBXX:8:1109:30401:16682   83   chr1 21774 1   100M =    21740 -134 ACCCCTGTGCAGTCCGGCCTTCGCGGCAGACCACCATACACCACGTTCCAAGCCACACTG
K00211:113:H52VJBBXX:8:1215:18578:23417   99   chr1 21783 1   100M =    21872 189  CAGGCCGGCCTTCGCGGCAGACCACCATACACCACGTTCCAAGCCACACTGAGGCCTCCC
```

Figure 2. Examples of FastQ (panel A) and SAM (panel B) input files.

var_index	chrom	gene_name	left	right	ref_seq	var_seq1	count	var_score	where_in_transcript	change_type1
11173	chr1	FAM87B	820507	820508	C	G	7	34.1428571	3' untranscribed	
2070956	chr6	TBP	170569705	170569706	T	C	54	31.3888889	CDS	Non-synonymous
349657	chr2	AGBL5	27074846	27074847	G	C	12	32.5833333	5' untranscribed	
1380229	chr7	POLR2J2	102667109	102667110	A	A	8	28.25	CDS	Non-synonymous
1380256	chr7	POLR2J2	102667105	102667106	C	C	6	29.8333333	CDS	Non-synonymous
1380215	chr7	POLR2J2	102667104	102667105	C	C	6	30.8333333	CDS	Non-synonymous
1446619	chr2	POLR1B	112550939	112550940	T	A	65	31.2	CDS	Synonymous

Figure 3. OncoMiner Input (OMI) file example

consequences of genetic mutations. SNPeff [8] is a set of tools for annotating and predicting the effects of GSVs on genes. MuSiC [9] is a package that provides statistical methods to identify significantly mutated genes. Each of these program packages contains their own preprocessing procedures for converting NGS data files to the required format for downstream analysis.

The OncoMiner pipeline [4] was originally developed in support of researchers in the UTEP Border Biomedical Research Center for the investigation of GSVs from a group of patients with leukemia in the El Paso Children's Hospital. The current OncoMiner pipeline provides the necessary tools for literature search, visualization, and statistical analysis with control of false discovery rates in one package, but it can only accept input files in a specific format that comes from the NGS and GSV identification services provided by Otogenetics Corporation. However, as the scope of the study expands and more genome sequencing is now being done at different locations by different sequencers, an additional data preprocessing step is needed to obtain the required information to generate an OncoMiner input (OMI) file.

At a minimum, an OMI file requires 11 data items for each variant as shown in Figure 3. These include a unique numeric identifier (var_index), its position on the reference genome (chrom, left and right), gene name (gene_name), the nucleotides involved (ref_seq and var_seq1), the number of sequences obtained (count) and an averaged sequencing quality score over those sequences (var_score). Additionally, if a variant is within a transcript, the where_in_transcript field states whether the variant falls on a CDS, an intron, or an untranslated region. The field change_type1 tells whether the variant is synonymous or nonsynonymous if it is within a CDS. In the next section, we will describe the steps involved to get the above information from an NGS dataset of an individual in FastQ or BAM format and prepare the OMI file.

3 METHODS

3.1 Overall Workflow

The OncoMiner Preprocessing (OP) program has three components. First, the open-source Bowtie2 aligner [5] is used to align the sequence data in a FastQ file to a reference human genome to produce a BAM file. Second, we use the SAMtools software toolkit [10], along with our file-splitting awk script to sort the aligned data and separate them by chromosomes. Finally, we have developed a mutation-calling (MC) python script to identify GSVs and generate the OMI file for input into OncoMiner. The overall workflow is displayed in Figure 4 and it has been implemented on two Linux-based local computers as well as the Blue Waters system at the National Center of Supercomputing Applications (NCSA).

3.2 Alignment by Bowtie2

An NGS data file, in FastQ format (Figure 2A), would first go through Bowtie2 [5], which aligns sequences in the FastQ file to the reference human genome version GRCh38 (Genome Reference Consortium human build 38), available at the

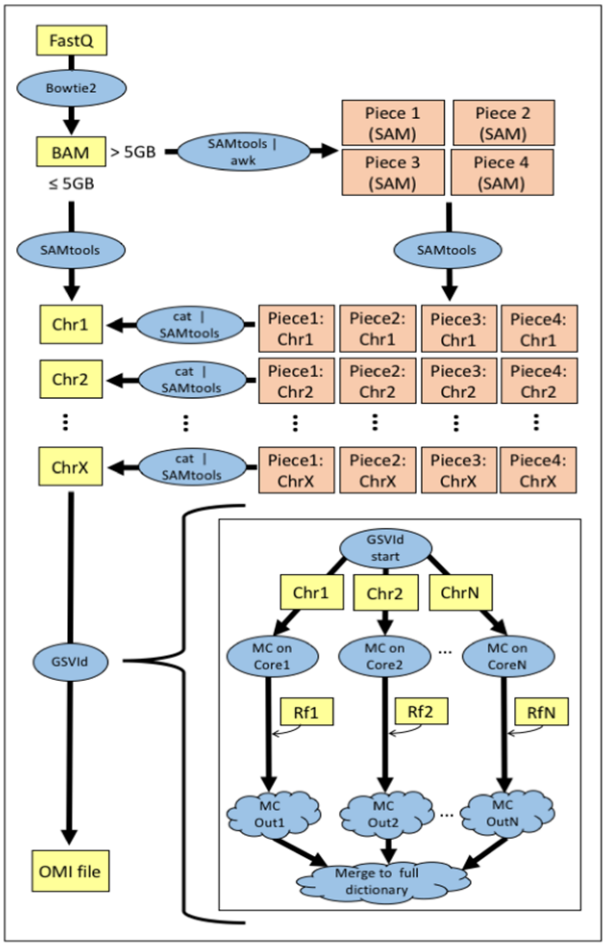


Figure 4. Workflow through which a FastQ or BAM file is processed to become an OncoMiner Input (OMI) file.

```
DDX11L1    NR_046018  chr1  +    11873 14409 14409 14409 3    11873,12612,13220,    12227,12721,14409,
WASH7P     NR_024540  chr1  -    14361 29370 29370 29370 11   14361,14969,15795,16606,16857,17232,17605,17914,18267,24737,29320,    14829,15038,
MIR6859-1  NR_106918  chr1  -    17368 17436 17436 17436 1    17368,     17436,
MIR6859-2  NR_107062  chr1  -    17368 17436 17436 17436 1    17368,     17436,
MIR6859-3  NR_107063  chr1  -    17368 17436 17436 17436 1    17368,     17436,
MIR6859-4  NR_128720  chr1  -    17368 17436 17436 17436 1    17368,     17436,
MIR1302-2  NR_036051  chr1  +    30365 30503 30503 30503 1    30365,     30503,
MIR1302-9  NR_036266  chr1  +    30365 30503 30503 30503 1    30365,     30503,
```

Figure 5. Example of a refflat file.

University of California at Santa Cruz (UCSC) Genome Browser [11]. These alignments are stored as a sequence alignment map (SAM) file as shown in Figure 2B or its binary equivalent BAM file. The SAM file contains readable text with the DNA sequence fragments as well as descriptions that include read lengths, ASCII-encoded quality scores of each nucleotide, chromosomal positions and mutation information in comparison to the reference sequence. The corresponding BAM file contains the same information in binary format, and has a much smaller file size.

3.3 File Sorting using SAMtools

The SAMtools sort and index functions are used on the BAM file generated by the alignment step above in order to group the GSVs on the same chromosome together and sort them by their positions on the chromosome. At this point, the SAMtools view function is used to extract information from the sorted BAM file and write them to 23 readable SAM files each containing the sorted, chromosome specific GSVs. These 23 chromosome specific SAM files are named Chr1, Chr2,…, Chr22, and ChrX in Figure 4, where C*n* stands for chromosome *n* for $n = 1,\ldots, 22$, and ChrX the sex chromosome (generally only chromosome X is sequenced even for a male subject as the chromosome Y is considered sufficiently similar to a portion of chromosome X).

When handling large BAM files on computers with limited memory (e.g., 32 GB), out-of-memory (OOM) errors sometimes occur when running the SAMtools sort function. This problem is circumvented by using an awk script to split a BAM file with size exceeding a threshold (e.g., 5 GB) into four pieces, each of which is then sorted by chromosome as described above. The threshold can be adjusted by the user according to the available amount of memory in the particular machine running the program. The four files associated with each chromosome are then joined back together by the cat command before feeding into the next step for GSV identification.

3.4 GSV Identification and Mutation-Calling

To identify GSVs and build the OMI file for input into OncoMiner, we have developed the mutation-calling (MC) script to parse the information contained in the chromosome specific SAM files from the previous sorting step. Information for most of the required fields in the OMI file, such as the GSV location, nucleotides involved, gene name, can be parsed directly from the SAM files. However, classifying the genomic region type for each variant is not as straightforward. GSVs have to be classified based on information obtained from the UCSC Genome Browser in the form of a refflat file (Figure 5) [12], which contains reference information for the start and end positions of various genes, introns, exons, and untranscribed regions.

Using this information, each GSV can be classified according to the decision tree shown in Figure 6. Each unique GSV is given an identifier consisting of the chromosome number and the position of the GSV within the chromosome. It is then added to the GSV dictionary. For each sequence containing the GSV, the site is counted and the sequence quality is tracked. Once all the sequences have been processed, GSVs that fail to meet minimum quality scores and sequence depth specified by the user are removed.

3.5 Parallelization

Bowtie2 is an open-source program that can run in parallel by simply specifying the number of processors in the command. In contrast, the SAMtools functions are designed to run in serial so we have not attempted any parallelization for them. However, the process of extracting GSV information from the sorted BAM file to produce the chromosome specific SAM files has been parallelized using different processors to extract the information for different chromosomes and write to different files.

For GSV identification, we have created the GSVId function that makes use of the Python multiprocessing module to run the MC script in parallel, distributing the chromosome specific files to run on different cores. Since the first two chromosomes contain the largest number of genes and are likely to take the longest to run, we start with assigning these two chromosomes to two cores first, and then the others to the remaining available cores.

3.6 Implementation and Testing

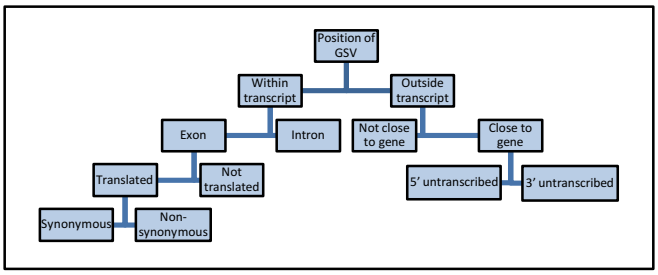


Figure 6. GSVId decision tree. The MC script classifies mutations according to their positions in relation to genetic transcripts. All decisions are based on gene information within the refflat file except for "Close to gene" which is a tunable parameter set as a default of 5000 nucleotides from either beginning or end of transcript.

The OP program has been implemented on two local computers at UTEP. The first one is BioTower, a Dell Precision 5810 containing an Intel Xeon E5-1650 with a 12-core processor and 32 GB memory. The second machine is BinfCompute, a more powerful Dell PowerEdge R730 with 32 cores (dual Intel Xeon E5-2667 processors with 16 cores) and 256 GB memory. Both computers have CentOS 7 as operating system and use the OS default version of Python (v2.7). The required Python modules, Bowtie2, SAMtools and reference files are locally available on these machines.

For initial testing of OP, we used a FastQ file generated by our local DNA sequencer in the Genomics Core Facility in the Border Biomedical Research Center at UTEP. The file was 7.3 GB in size containing 29 million 100-base long sequences spanning all 23 human chromosomes. OP was run on both machines using varying number of cores to check that parallelization of each step has been achieved. The final output file of OP was inputted to OncoMiner to check that a legitimate OMI file was produced.

For further performance testing, we have also implemented OP on the Blue Waters system, a Cray XE/XK hybrid machine composed of AMD 6276 Interlagos processors running the Cray Linux Environment. Our OP program implementation uses only one high memory XE node with 128 GB total memory to process the dataset from one individual. The Blue Waters Python software stack bwpy and PrgEnv-gnu modules have to be loaded in order to use Python and the GNU programming environment respectively.

A collection of BAM files containing the aligned sequence information of 35 patients with AML was obtained from The Cancer Genome Atlas (TCGA, http://cancergenome.nih.gov/) for testing [13]. These file sizes range from 15 to 54 GB. In all the test runs, runtimes were determined using the Linux time command and internally using Python's time module. Internal memory usage was determined using Python's getrusage() function.

We have assessed the performance of the parallelization on the local machines by calculating the efficiency of core usage as $T(1)/[pT(p)]$ where $T(p)$ is the measured runtime using p cores with varying p = 1, 2, 4, 8, 16, and 24. Statistical analysis of efficiencies and runtimes were performed using the functions for t-tests and linear models in the statistical software package R [14].

4 Results and Discussion

4.1 The OP program

The OP program has been successfully implemented on both of our local machines BioTower and BinfCompute. It can take FastQ files as input and produce OMI files as output in the correct format that can then be fed into OncoMiner for downstream analysis. Because many datasets in public databases such as TCGA are already stored in the form of BAM files, our OP program has been set up to also take input in BAM format.

To set a baseline for parallelization performance assessment, we first conducted runtime measurements of OP using the locally generated 7.3 GB FastQ file on a single core in the two local machines. The runtimes of the various steps on BinfCompute are displayed in Table 1 (the run time distribution on BioTower is similar). These results show that about 80% of the total OP runtime is taken by Bowtie2 in the alignment step. Fortunately, Bowtie2 is designed to run in parallel, and the speedup using multiple cores seems quite substantial. While the SAMtools sort and index functions must run serially, parallelization of the extraction process to produce chromosome specific SAM files by the SAMtools view function has reduced the runtime somewhat.

Table 1. Runtimes (in minutes) of different steps of OP on BinfCompute using 1, 8, and 24 cores.

Function	1 Core	8 Cores	24 Cores
Alignment (Bowtie2)	145.16	22.39	10.08
File sorting (SAMtools)	13.12	6.33	5.85
GSVId (MC script)	19.17	3.30	2.56
Others	0.32	0.29	0.29
Total	**177.76**	**32.31**	**18.78**

As the GSV identification part of the OP program was developed entirely by our group, we examined the MC script performance more carefully. Figure 7 displays the runtimes for our test dataset on BioTower and BinfCompute, showing substantial speedups as the number of cores increases from 1 to 8 on both machines. For BioTower with only 32 GB of RAM, the execution speed deteriorated sharply beyond 8 cores as swap memory started to be used. We therefore stopped the BioTower runtime measurements at that point, but continued the measurements on BinfCompute using higher number of cores. Overall, the best average MC script speedup achieved on BioTower was 3.32 using 6 cores, and on BinfCompute was 9.29 using 16 cores.

Aside from the locally generated 7.3 GB FastQ file, we also selected 9 datasets from our AML collection obtained from TCGA to run on the two local machines. These data files were already in BAM format but were much larger than our test dataset. OOM errors were encountered on BioTower during the execution of the SAMTools sort function. Such problems did not occur on BinfCompute though. Given that BioTower has only 32 GB of memory while BinfCompute has 256 GB, this is not surprising. It has, however, suggested that memory requirement is an issue at this point of the OP program.

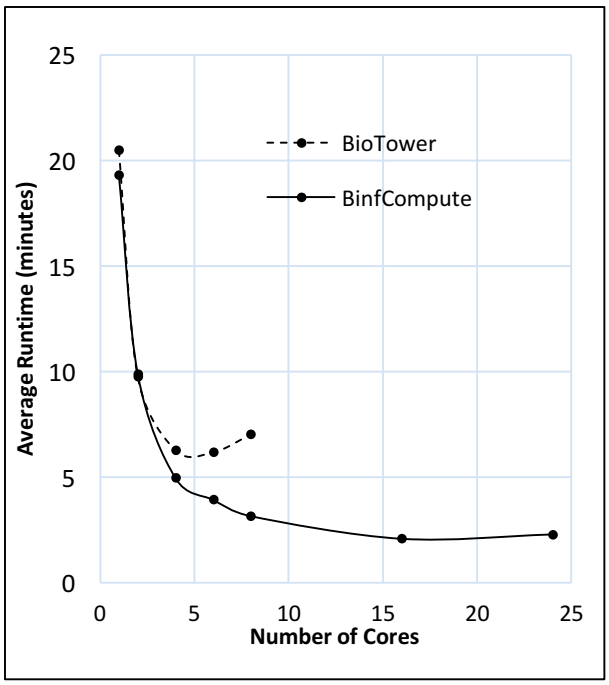


Figure 7. Runtimes for test dataset on two local computers: BioTower and BinfCompute.

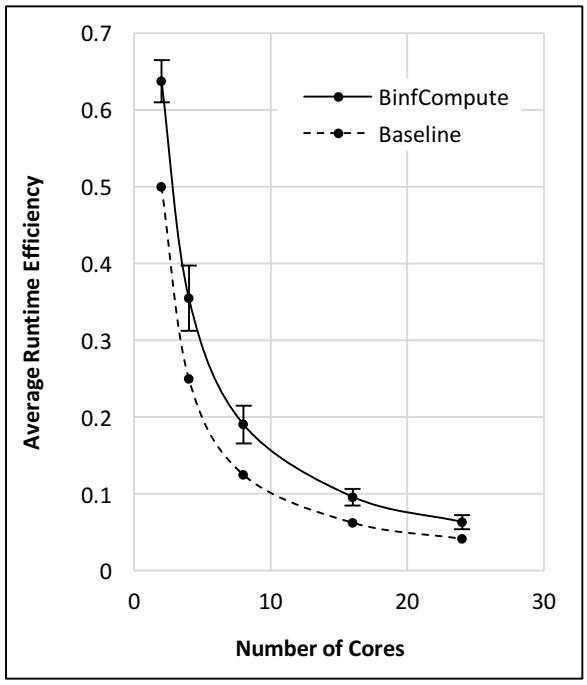


Figure 8. Average efficiencies ± standard deviation for 2, 4, 8, 16, 24 cores on BinfCompute. Dashed curve indicates baseline efficiencies.

With the expectation that the OP program might need to be run on other computers without large amounts of memory, we used awk to split the large BAM files exceeding a cutoff file size into four pieces and let each piece be sorted one at a time. The cutoff file size can be set by the user with consideration to the available memory of the specific computer running OP. For BioTower, we found that a 5 GB cutoff worked well. The splitting slowed down the file sorting step substantially, but OOM errors were avoided.

We further examined the overall parallelization efficiency of the OP program on BinfCompute for the 9 selected AML dataset. If no speedup were achieved at all by the parallelization, the efficiency would have a baseline value of 1/p. Figure 8 shows the average efficiency of the OP program on BinfCompute with p = 2, 4, 8, 16, 24. In each case, a t-test confirmed statistically that the average efficiency was at least 20% higher than the baseline values at significance level $\alpha = 0.05$.

In the test runs of these 9 files, the runtimes required ranged from around 15 minutes to almost 9 hours. It seemed not practical to run the OP program for all the TCGA datasets on our local computers as each run would tie up the computer for a substantial amount of time and prevent others from running their jobs on these machines that were used heavily. We therefore turned to Blue Waters to utilize the allocated resources to complete this project.

4.2 Running OP on Blue Waters

The OP program was installed on Blue Waters using one high memory XE node with 128 GB memory to process each of the 35 datasets from the AML collection. While Blue Waters allows multiple nodes to be used at one time, we decided to process a dataset in a single node, as the shared memory within the same node made it easier to construct the dictionary of GSVs and kept communication time to a minimum. However, we were able to use multiple nodes to independently process multiple datasets simultaneously.

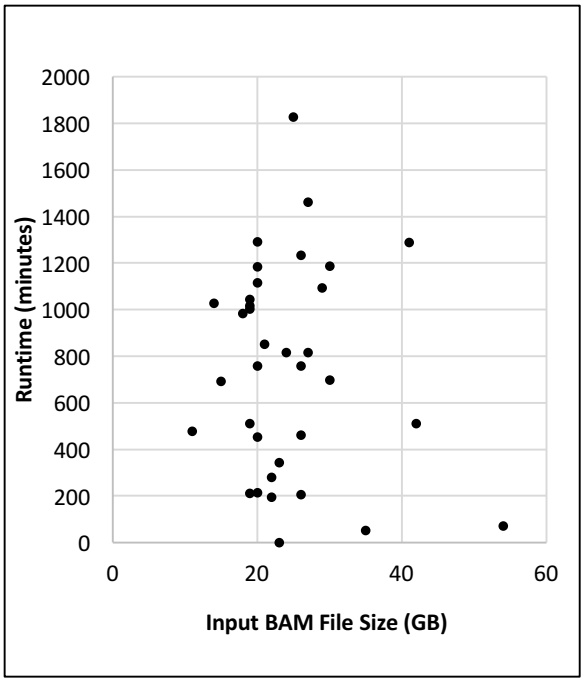


Figure 9. OP runtimes versus input file size for 35 datasets in AML collection

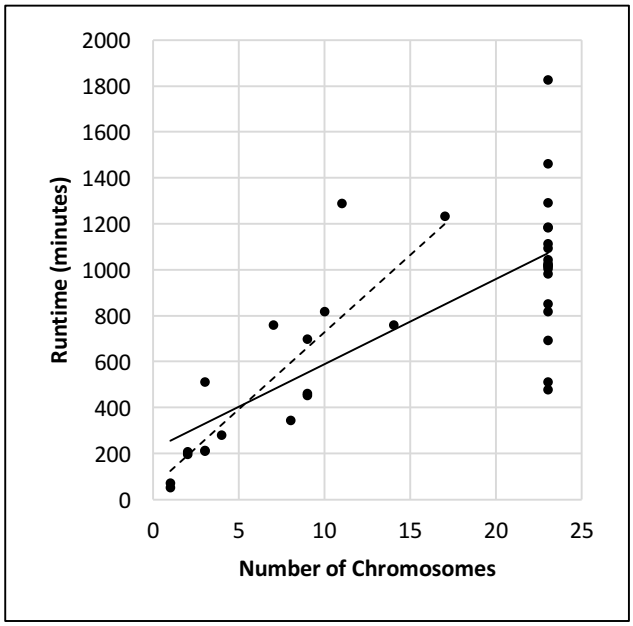


Figure 10. OP runtimes versus the number of chromosomes contained within the files. The solid regression line was determined with all data points while the dashed line used only the data points with no more than 22 chromosomes.

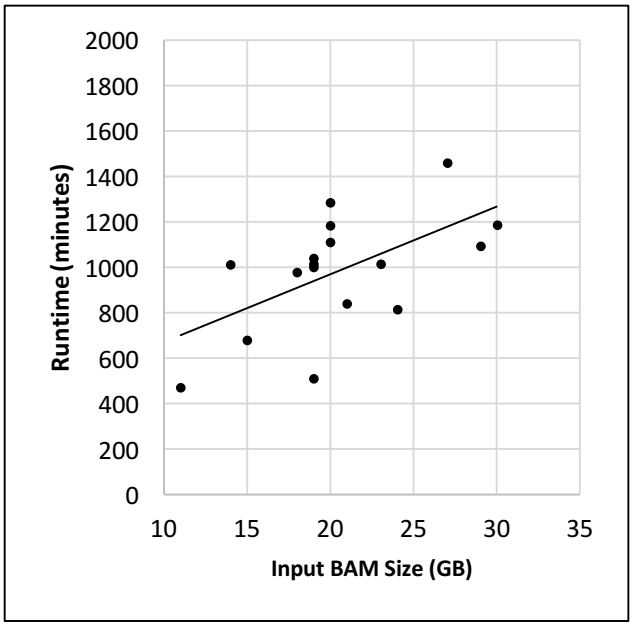


Figure 11. OP runtimes versus input file size for datasets with 23 chromosomes

OOM errors occurred in the MC script on Blue Waters for a few datasets when the node could not provide sufficient memory for processing all the chromosome files at once. In those cases, we had to reduce the number of cores used so that fewer chromosomes were processed at one time. On hindsight after understanding the shared memory constraints, a better approach to parallelize the OP program on Blue Waters would be to couple MPI to our script and allocate a node to process each of the four pieces of one chromosome-specific file, and then join the four dictionaries afterwards. We plan to implement this approach in the next version of OP.

Furthermore, we observed that using more than one core in the node produced no speedup in runtime. After monitoring the memory usage on Blue Waters, the reading and writing (I/O) of the utilized files was believed to be the cause of this lack in speedup because of the data transfer to and from the compute node. This would not have readily been seen on the BinfCompute and BioTower machines using local storage of the data but is a known issue for HPC systems analyzing large datasets [15].

Despite these issues, Blue Waters was the only platform that provided us with sufficient resources to process our complete AML dataset collection. From the recorded runtimes on Blue Waters, we were also able to investigate which characteristics of the datasets would influence the runtimes significantly, as described below.

4.3 Runtime correlates with input file size and chromosome diversity

The runtimes for our 35 AML datasets using one core varied from 53 minutes to over 30 hours. One would expect larger input files to require longer runtime. Surprisingly, a simple linear regression analysis indicated that the correlation r = 0.152 was not significant (p value = 0.385). Looking at the scatter plot in Figure 9, we noticed a few outliers that might have contributed to the unexpected result. For example, the largest input file (54 GB) ran very quickly. On closer examination, we found that this dataset contains GSVs from only one chromosome of the patient.

This prompted us to look into how the number of chromosomes in the input file might affect the OP runtime (Figure 10). This time, regression analysis showed a highly significant linear correlation (r = 0.762, p value = 1.07e-07). The correlation was even stronger (r = 0.858) when only the datasets with no more than 22 chromosomes were considered. This result suggested that chromosome diversity could be an important factor that influenced the OP runtime. The I/O involved in the analysis of individual chromosome files is believed to drive the major difference in runtimes of files with different numbers of chromosomes. Having multiple cores trying to access the larger files from a network drive at the same time could create a bottleneck in the accessibility of the data and thus causing the individual processes to slow down. This would not be as much of a factor when pulling data from a local hard drive.

When the OP runtimes for only those files with complete sets of 23 chromosomes are analyzed, we then see a significant positive

correlation with input file size (see Figure 11, r = 0.509, p value = 0.031). This implies that a strong relationship between input file size and runtime indeed exists once the number of chromosomes is fixed.

To complete the runtime analysis, a multiple regression model was fit to our runtime data with the number of chromosomes and input file size as covariates, producing the regression equation:

$$\text{Runtime} = 94.8 + 20.3 * (\#\text{ chromosomes}) + 22.3 * (\text{file size})$$

with coefficient of determination 0.749, implying that almost 75% of the variations in runtime can be explained by input file size and chromosome diversity together. This regression equation will allow OP runtimes to be estimated when we process new data files in the future.

5. CONCLUSION AND FUTURE WORK

We have implemented the OP program, which comprises the open source Bowtie2 and SAMtools programs, as well as our GSVId function, on two local computers at UTEP and on Blue Waters at the NCSA. The OP program can preprocess NGS data stored in either FastQ or BAM format and obtain the necessary information to produce an OMI file to be inputted to OncoMiner for downstream genetic variation analysis.

We have demonstrated that our multiprocessing parallelization approach in GSVId for the MC script works with reasonable efficiencies on our local computers. However, the same parallelization using multiple cores on one node in Blue Waters did not produce any substantial speedup of the OP program. We have identified possible factors involving memory constraints and I/O issues that limit the OP program's performance on Blue Waters and will continue to develop a better approach using MPI to distribute the analysis of a single dataset to multiple nodes. We also plan to test the script on other high performance platforms with more memory in a single node and internal solid state drives for the I/O intensive portions of the code.

Despite the memory and I/O issues encountered, Blue Waters provided the necessary resources for us to process our entire collection of datasets from 35 patients with AML and showed that OP runtimes were correlated not only with the input data file size, but also with chromosomes diversity.

Aside from the most popular FastQ and BAM formats, genome sequence variant data may also come in other file formats such as the variant call format (VCF). The OP program framework set up in this project now allows us to adapt and extend our code relatively easily to process files in other formats to produce OMI files for input to OncoMiner.

6. REFLECTIONS

The Blue Waters Student Internship Program (BWSIP) allowed me to learn about parallel computing, which I had not been previously introduced to. I am glad to have attended the two-week Petascale Institute 2016 workshop. What I learned during the workshop and internship will be useful in the future for performing bioinformatics analysis. The NCSA and Shodor staff was very helpful and explained the concepts of parallel computing clearly. I had an eye-opening experience in the BW Symposium in May of 2017 as I got to see how HPC was used in so many different fields with methodologies that I had not encountered in bioinformatics related projects.

Due to parallelization of the MC script having been effective on the local machines, memory usage was a suspected cause of static runtime on Blue Waters. The process of ruling out possible causes for the lack of speedup was an invaluable lesson. Looking at the memory usage of the MC script helped me better understand memory related issues in parallel programming and taught me how to monitor the usage at different points in the script. Although the runtime remained static, I now have a better feel for how to check to see if our current suspect, I/O, is behind it. In this, the future work will revolve around machines with higher memory and designed for I/O intensive programs.

The research experience also allowed for a local collaboration at UTEP. While working with my group, I learned about version control and improved on communication with other group members. I learned the value in clarifying tasks at the onset and in keeping track of changes to allow easier modifications during debugging. In conclusion, the internship has not only made me more knowledgeable in the use of HPC systems, but also trained me to become a better researcher.

7. ACKNOWLEDGMENTS

This research is funded in part by the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation (awards OCI-0725070 and ACI-1238993) and the state of Illinois. Blue Waters is a joint effort of the University of Illinois at Urbana-Champaign and its National Center for Supercomputing Applications. This project was also in part supported by NIH Grant #5G12RR007592 from the National Center for Research Resources (NCRR)/NIH to the Border Biomedical Research Center at The University of Texas at El Paso. The results published here are in whole or part based upon data generated by the TCGA Research Network: http://cancergenome.nih.gov/.

8. REFERENCES

[1] Bullinger, L., Dohner, K. and Dohner, H. Genomics of Acute Myeloid Leukemia Diagnosis and Pathways. *J Clin Oncol*, 35, 9 (Mar 20 2017), 934-946.

[2] Corvol, H., Blackman, S. M., Boelle, P. Y., Gallins, P. J., Pace, R. G., Stonebraker, J. R., Accurso, F. J., Clement, A., Collaco, J. M., Dang, H., Dang, A. T., Franca, A., Gong, J., Guillot, L., Keenan, K., Li, W., Lin, F., Patrone, M. V., Raraigh, K. S., Sun, L., Zhou, Y. H., O'Neal, W. K., Sontag, M. K., Levy, H., Durie, P. R., Rommens, J. M., Drumm, M. L., Wright, F. A., Strug, L. J., Cutting, G. R. and Knowles, M. R. Genome-wide association meta-analysis identifies five modifier loci of lung disease

severity in cystic fibrosis. *Nat Commun*, 6 (Sep 29 2015), 8382.

[3] Hilven, K., Vandebergh, M., Smets, I., Mallants, K., Goris, A. and Dubois, B. Genetic basis for relapse rate in multiple sclerosis: Association with LRP2 genetic variation. *Mult Scler* (Jan 1 2018), 1352458517749894.

[4] Leung, M.-Y., Knapka, J. A., Wagler, A. E., Rodriguez, G. and Kirken, R. A. *OncoMiner: A Pipeline for Bioinformatics Analysis of Exonic Sequence Variants in Cancer*. In: Big Data Analytics in Genomics, Wong, K.C. (Ed.), Springer, New York, USA (2016), 373-396.

[5] Langmead, B. and Salzberg, S. L. Fast gapped-read alignment with Bowtie 2. *Nat Methods*, 9, 4 (Mar 4 2012), 357-359.

[6] Li, H. and Durbin, R. Fast and accurate short read alignment with Burrows–Wheeler transform. *Bioinformatics*, 25, 14 (2009), 1754-1760.

[7] Wang, K., Li, M. and Hakonarson, H. ANNOVAR: functional annotation of genetic variants from high-throughput sequencing data. *Nucleic Acids Res*, 38, 16 (Sep 2010), e164.

[8] Cingolani, P., Platts, A., Wang le, L., Coon, M., Nguyen, T., Wang, L., Land, S. J., Lu, X. and Ruden, D. M. A program for annotating and predicting the effects of single nucleotide polymorphisms, SnpEff: SNPs in the genome of Drosophila melanogaster strain w1118; iso-2; iso-3. *Fly (Austin)*, 6, 2 (Apr-Jun 2012), 80-92.

[9] McKenna, A., Hanna, M., Banks, E., Sivachenko, A., Cibulskis, K., Kernytsky, A., Garimella, K., Altshuler, D., Gabriel, S., Daly, M. and DePristo, M. A. The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data. *Genome Res*, 20, 9 (Sep 2010), 1297-1303.

[10] Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Abecasis, G. and Durbin, R. The Sequence Alignment/Map format and SAMtools. *Bioinformatics*, 25, 16 (Aug 15 2009), 2078-2079.

[11] Karolchik, D., Baertsch, R., Diekhans, M., Furey, T. S., Hinrichs, A., Lu, Y. T., Roskin, K. M., Schwartz, M., Sugnet, C. W., Thomas, D. J., Weber, R. J., Haussler, D. and Kent, W. J. The UCSC Genome Browser Database. *Nucleic Acids Res*, 31, 1 (Jan 1 2003), 51-54.

[12] Tyner, C., Barber, G. P., Casper, J., Clawson, H., Diekhans, M., Eisenhart, C., Fischer, C. M., Gibson, D., Gonzalez, J. N., Guruvadoo, L., Haeussler, M., Heitner, S., Hinrichs, A. S., Karolchik, D., Lee, B. T., Lee, C. M., Nejad, P., Raney, B. J., Rosenbloom, K. R., Speir, M. L., Villarreal, C., Vivian, J., Zweig, A. S., Haussler, D., Kuhn, R. M. and Kent, W. J. The UCSC Genome Browser database: 2017 update. *Nucleic Acids Res*, 45, D1 (Jan 4 2017), D626-d634.

[13] Ley, T. J., Miller, C., Ding, L., Raphael, B. J., Mungall, A. J., Robertson, A., Hoadley, K., Triche, T. J., Jr., Laird, P. W., Baty, J. D., Fulton, L. L., Fulton, R., Heath, S. E., Kalicki-Veizer, J., Kandoth, C., Klco, J. M., Koboldt, D. C., Kanchi, K. L., Kulkarni, S., Lamprecht, T. L., Larson, D. E., Lin, L., Lu, C., McLellan, M. D., McMichael, J. F., Payton, J., Schmidt, H., Spencer, D. H., Tomasson, M. H., Wallis, J. W., Wartman, L. D., Watson, M. A., Welch, J., Wendl, M. C., Ally, A., Balasundaram, M., Birol, I., Butterfield, Y., Chiu, R., Chu, A., Chuah, E., Chun, H. J., Corbett, R., Dhalla, N., Guin, R., He, A., Hirst, C., Hirst, M., Holt, R. A., Jones, S., Karsan, A., Lee, D., Li, H. I., Marra, M. A., Mayo, M., Moore, R. A., Mungall, K., Parker, J., Pleasance, E., Plettner, P., Schein, J., Stoll, D., Swanson, L., Tam, A., Thiessen, N., Varhol, R., Wye, N., Zhao, Y., Gabriel, S., Getz, G., Sougnez, C., Zou, L., Leiserson, M. D., Vandin, F., Wu, H. T., Applebaum, F., Baylin, S. B., Akbani, R., Broom, B. M., Chen, K., Motter, T. C., Nguyen, K., Weinstein, J. N., Zhang, N., Ferguson, M. L., Adams, C., Black, A., Bowen, J., Gastier-Foster, J., Grossman, T., Lichtenberg, T., Wise, L., Davidsen, T., Demchok, J. A., Shaw, K. R., Sheth, M., Sofia, H. J., Yang, L., Downing, J. R. and Eley, G. Genomic and epigenomic landscapes of adult de novo acute myeloid leukemia. *N Engl J Med*, 368, 22 (May 30 2013), 2059-2074.

[14] R Core Team. *R: A language and environment for statistical computing.* R Foundation for Statistical Computing, Vienna, Austria. (2017).

[15] Reed, D. A. and Dongarra, J. Exascale computing and big data. *Communications of the ACM*, 58, 7 (2015), 56-68.

December 2018

Volume 9 Issue 2

TABLE OF CONTENTS

ISSN 2153-4136 (online)