Improving the Portability and Performance of jViz.RNA – A Dynamic RNA Visualization Software

Boris Shabash, Kay Wiese, and Edward Glen

Simon Fraser University, School of Computing Science
8888 University Drive, Burnaby, BC, Canada
http://www.sfu.ca

Abstract. In this paper, four methods were explored for improving the performance of jViz.RNA's structure drawing algorithm when dealing with large sequences; First, the approximation based Barnes-Hut algorithm was explored. Second, the effects of using multithreading were measured. additionally, dynamic C libraries, which integrate C code into the Java™environment, were investigated. Finally, a technique termed structure recall was examined.

The results demonstrated that the use of the Barnes-Hut algorithm produced the most drastic improvements in run-time, but distorts the structure if too crude of an approximation is used. Multithreading and integration of C code proved to be favorable approaches since these improved the speed at which calculations are done, without distorting the structures.

jViz.RNA is available to download from http://jviz.cs.sfu.ca/.

1 Introduction

jViz.RNA is an RNA visualization software developed at Simon Fraser University [5,12,13]. jViz.RNA draws its strength from three major components. First, since it is written in Java™, it is platform independent and can run on any machine that has the Java Virtual Machine (JVM) installed. Second, unlike other RNA visualization tools [4,6,7], jViz.RNA offers a dynamic model that users can interact with. This gives users the ability to modify the layout generated by jViz.RNA if they feel they need to. Third, jViz.RNA offers the ability to inspect several different aspects of the RNA molecule explored, as well as the ability to compare two RNA structures. This aspect is particularly useful when working alongside algorithms designed to predict RNA structure.

In the research proposed in this paper, our aim was to improve the run-time performance of jViz.RNA's structure drawing algorithm, in order to allow faster rendering of large RNA sequences. During the course of this research, jViz.RNA's portability was also extended by incorporating the RNAML and FASTA file formats. However, the details of these file formats' integration are omitted for brevity purposes.

T. Shibuya et al. (Eds.): PRIB 2012, LNBI 7632, pp. 82–93, 2012.

Unlike other visualization tools that allow for manipulation of the resulting layout [2,3,9,8], the RNA drawing approach jViz.RNA employes relies on simulating the natural forces that are at work on RNA inside the cell, thus simulating its natural folding process[1]. Furthermore, jViz.RNA renders the folding while it calculates it, thus giving the user an interactive model. However, a major drawback of such an approach is that it is more time consuming than preparing an immutable layout discretely. The calculations involved propose a running time of, at worst, $O(n^2)$ for each iteration. This work aimed at reducing the run-time required for jViz.RNA's drawing algorithm by using both algorithmic methods aimed to try and approximate the interactions involved at each iteration in exchange for a faster theoretical run time, as well as software engineering approaches aimed at decreasing the run time of individual instructions.

The remainder of this article is organized as follows; Section 2 presents insight into the different approaches employed to improve jViz.RNA's drawing algorithm, as well as the results, Section 3 discusses the significance of the results and possible extensions, and finally Section 4 provides a concluding summary.

2 jViz.RNA's Drawing Algorithm and Potential Improvements

2.1 Approach

jViz.RNA aims at presenting the user with a dynamic, interactive, model. The model would behave according to the forces acting on each nucleotide, which were estimated by incorporating Newtonian mechanics and electrostatics into the simulation; Each nucleotide would experience attraction forces ($\boldsymbol{F}_{attraction}$) from nucleotides that it is bonded to (two nucleotides from the backbone and another one if it is involved in base pairing), as well as repulsion forces ($\boldsymbol{F}_{repulsion}$) from all other nucleotides in the simulation, simulating electrostatic repulsion. The structure would be in a stable conformation when the two forces for each nucleotide cancel out, i.e. when $\boldsymbol{F}_{attraction} = \boldsymbol{F}_{repulsion}$.

To simulate the forces acting on each nucleotide, a spring based model was developed. The attraction force between two nucleotides is given by

$$\boldsymbol{F}_{attraction} = k\Delta d \tag{1}$$

where k is a spring coefficient and Δd is the distance between two nucleotides. The repulsion force between two nucleotides is given by

$$\boldsymbol{F}_{repulsion} = \frac{q}{\Delta d^2} \tag{2}$$

where q is the repulsion coefficient and Δd is again the distance between the two nucleotides. To draw the RNA structure's layout, the sequence is first laid in a

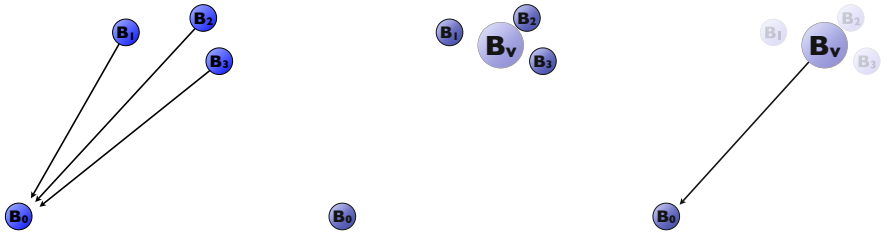
[1] This is not to imply jViz.RNA predicts folding, it simply simulates it after other software have established the correct base pairing of the RNA molecule.

circle, and the forces acting on the nucleotides are calculated iteratively until $\boldsymbol{F}_{attraction} = \boldsymbol{F}_{repulsion}$ for all nucleotides. The calculation of attracting forces for all nucleotides at a given iteration is in order of $O(n)$, since each nucleotide is bonded to at most three other nucleotides. However, calculation of repulsive forces can be in order of $O(n^2)$ if all interactions are considered.

The most recent iteration of jViz.RNA employed the idea of a "neighborhood" of nucleotides. The neighborhood of each nucleotide increased throughout the iterations, until eventually, for every nucleotide, all nucleotides were considered for calculating repulsion forces. However, for large sequences of size 1,000+ nucleotides (nt), this approach would still lag and large structures would take up to half an hour to collapse into a stable conformation.

The Barnes-Hut Algorithm. The Barnes-Hut algorithm [1] was originally employed in astrophysical simulations to simulate the movements of planets within galaxies. However, it showed potential to be useful in the context of bioinformatics as well.

The main premise of the Barnes-Hut algorithm is that if a group of bodies (nucleotides, in the case of jViz.RNA), e.g. B_1, B_2 and B_3, are far enough away from a body B_0 and close enough together, then the forces they exert on B_0 can be approximated by creating a virtual body, B_v, which combines all of their properties and lies in the center of the body group. In essence, if the group of bodies mentioned exerts repulsive forces over B_0 by Equation 2 described above, then instead of calculating $\boldsymbol{F}_{repulsion_{1\to0}} + \boldsymbol{F}_{repulsion_{2\to0}} + \boldsymbol{F}_{repulsion_{3\to0}}$, one can calculate $\boldsymbol{F}_{repulsion_{v\to0}}$. The virtual body B_v combines the properties of B_1, B_2 and B_3 into one body, and so fewer calculation need to be made (Figure 1).



(a) B_0 experiences repulsion forces by B_1, B_2, and B_3

(b) B_1, B_2 and B_3 can be approximated by a virtual body, B_v which encompasses all their properties and lies in the centre of the body group

(c) The virtual body B_v can be used to approximate the repulsion forces exerted on B_0, thus reducing calculations

Fig. 1. The basic premise for the Barnes-Hut algorithm is that if a group of bodies is close enough together, and far enough from another body, B_0, then their effects on it can be approximated by a virtual body

The algorithm accomplishes this simplification by first constructing a quad-tree (or oct-tree for 3D) where all the leaves are either bodies in the simulation or empty spaces and all internal nodes are groups of bodies represented by a virtual body.

Then each body in the simulation traces through the quad-tree and calculates which internal nodes simulate a group of bodies far enough away from it and close enough together such that their forces can be approximated. In theory, this results in a cruder approximation of the forces acting in the simulation but performs calculations in the order of $O(n \log n)$ rather than $O(n^2)$.

Multithreading. Multi-threading can be employed if there are independent calculations that can be distributed. In the case of force calculation in jViz.RNA, the set of force calculations can be divided by assigning each thread a set of nucleotides and having it calculate the forces acting on each nucleotide in the set.

For our purposes we chose to implement multithreading such that each of the m threads receives a set of $\frac{n}{m}$ nucleotides under its care, where n is the number of nucleotides in the structure in total. Then, each of these threads would calculate the forces acting on each of its nucleotides by all other $n-1$ nucleotides. In theory, the running time of the algorithm when multithreading is employed should be in the order of $O(\frac{n^2}{m})$.

There is the possibility that for smaller structures the time involved in managing the threads may add an increase in run-time greater than any time gained by splitting the calculations. In addition, larger structures may benefit from a different number of threads than smaller structures, and as a result, we have tested several different structures of varying lengths and saw how they behaved under an increasing number of threads.

The goal of this research was to deliver a proof of principle regarding the advantages multithreading can provide to jViz.RNA. As such, we tested multithreading on the brute-force approach (without the use of a neighborhood), and compared it with jViz.RNA's neighborhood algorithm. If multithreading shows promising results, it can then be implemented in jViz.RNA's neighborhood based algorithm.

Native C Code. Java™offers the advantage of platform independence due to the presence of the JVM. However, sometimes the use of the JVM can cause slowdowns since code needs to be interpreted through the JVM, adding an extra step for execution. To compensate, Java™offers the potential to integrate C code through dynamic libraries. However, unlike the Java™code, C code is not platform independent and so a library must be compiled for each separate architecture the code is to be executed on. In addition, since calling a C library from the Java™environment introduces an overhead, the integration of C libraries into Java™is most fruitful when large sets of computations are executed by the C code.

In this research as well, our purpose was to provide a proof of principle that the C code would work faster than the Java™code when dealing with medium-size and large-size sequences. If the improvement found would be great enough, it will be safe to assume that the use of native C code can be scaled to jViz.RNA's neighborhood based algorithm.

Structure Recall. The final approach for improving the performance of jViz.RNA is a concept we denote as 'structure recall'. In this setup, jViz.RNA would recall structures it has previously seen and lay them out exactly as they were last viewed. Since structures are often viewed more than once, it would save researchers and users the time spent waiting for the structure to collapse into a stable conformation.

In order to achieve this form of 'memory', jViz.RNA has to keep the information of each RNA structure previously viewed in files. The main decision that motivated the design of these files was whether the data should be stored in a file containing binary data (flat files) or files that contain the (x, y) coordinates of each nucleotide (structured files). Flat files offer the advantage of quick access time, since once the data would be read, simply a reference to the right type of data would have to be assigned to it. Alternatively, structured files would take longer to access the data, but could be employed by other applications.

Since users and programmers would not be able to visualize how an RNA molecule is laid out given the coordinates, and other applications may display an RNA molecule in a different way, it was decided to use the flat file format.

2.2 Results – Implementing and Testing Run-Time Optimization Methods

For the experiments in this section, the following 16 sequences were used to test the performance of different methods (Table 1). The sequences were selected to represent a wide spectrum of sizes that could be of interest to researchers in the natural sciences.

The experiments involving these 16 sequences were all performed on a MacBook Pro with a 2.4 GHz Intel Core 2 Duo and 2GB of RAM running Mac OS X Version 10.5.8.

The Barnes-Hut Algorithm. The first step in implementing the Barnes-Hut algorithm was defining a quad-tree and how it should be built. The tree was built anew at every iteration since updating it could be more costly in computation time than building it. At every iteration, the maximum and minimum x and y coordinates were found in order to establish the enclosing area of the quad-tree. Each internal node in the tree would represent a space, and would store its center, as well as its center of mass (which need not be the same as the center).

Table 1. The 16 sequences used for the experiments and their corresponding accession numbers

Structure Name	Accession Number	Size (nt)
Bacillus stearothermophilus 5S ribosomal RNA	AJ251080	117
Saccharomyces cerevisiae 5S ribosomal RNA	X67579	118
Agrobacterium tumefaciens 5S ribosomal RNA	X02627	120
Arthrobacter globiformis 5S ribosomal RNA	M16173	122
Deinococcus radiodurans 5S ribosomal RNA	AE000513:254392-254515	124
Metarhizium anisopliae var. anisopliae strain 33 28S ribosomal RNA group IB intron	AF197122	436
Tetrahymena thermophila 26S ribosomal RNA fragment (with intron)	V01416	517
Acomys cahirinus mitochondrial 12S ribosomal RNA	X84387	940
Xenopus laevis mitochondrial 12S ribosomal RNA	M27605	945
Homo sapiens mitochondrial 16S ribosomal RNA	J01415:648- 1601	954
Ailurus fulgens mitochondrial 16S ribosomal RNA	Y08511	964
Sulfolobus acidocaldarius 16S ribosomal RNA	D14876	2080
Aureoumbra lagunensis 18S ribosomal RNA	U40258	2236
Hildenbrandia rubra 18S ribosomal RNA (with intron)	L19345	2283
Porphyra leucosticta 18S ribosmal RNA (with intron)	AF342746	2404
Chlorella saccharophila 18S ribosomal RNA	AB058310	2510

Each node would also store a `mass` attributes denoting the mass of the body it represents. Leafs would have a `mass` of either 0 (for empty spaces), or 1 (nucleotides). Internal nodes would have their `mass` set equal to the sum of their four children's masses, and the center of mass for each node would be the weighted average of the centers of mass of its children. The nodes also each had a `size` attribute which indicated their circumference.

With the nodes defined, the tree itself was relatively straightforward to define; the only crucial parameter to define for the tree was the ratio parameter θ. When a nucleotide traverses the tree it inspects every internal node for its size, s, and the distance between the nucleotide and the node's center, Δd. If Equation 3 holds true,

$$\frac{s}{\Delta d} < \theta \tag{3}$$

then that internal node can be used as a virtual body to approximate the entire subtree beneath it. Otherwise, the nucleotide must examine each of the node's children and so on.

For the purposes of our experiments, we have experimented with θ values of 0.5, 1.0 and 5.0, as well as a set of experiments where the value of θ varied according to the sequence length. The run-time results can be seen in Figure 2.

It may seem as though the Barnes-Hut algorithm performs better as the value of θ increases. When using a θ value of 5.0, the running time of jViz.RNA dropped over 90%. However, it was important to also explore the visual effects that the Barnes-Hut algorithm delivered. Figure 3 demonstrates the results for

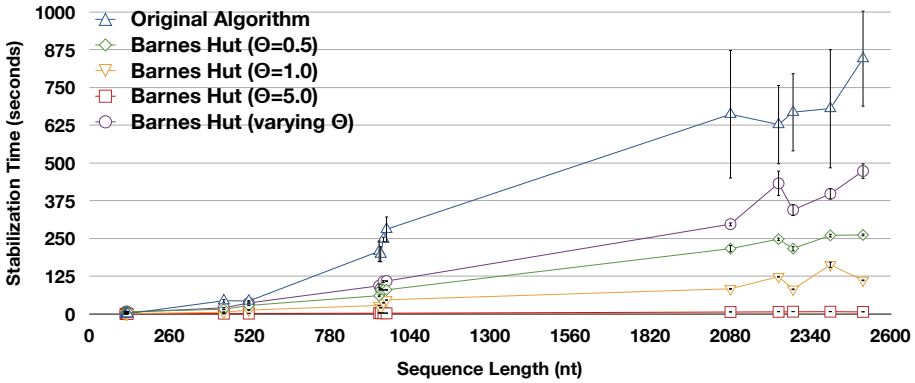


Fig. 2. The stabilization times (in seconds) for the different RNA sequences (as the number of nucleotides increases) using the Barnes-Hut algorithm under different θ values. 99% confidence intervals are used as error bars

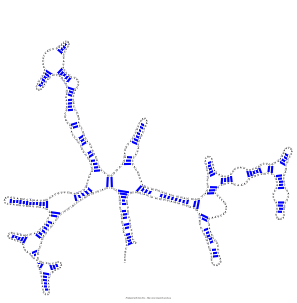
a particular sequence when using the Barnes-Hut algorithm, and similar results were observed for the other sequences. When the value of θ is 1.0, the resulting visualization becomes very crude. Especially for long stems of RNA base pairs, which seem to thin out as they extend (Figure 3(c)). These effects extend to even short helices as the value of θ increases to 5.0. However, when using a θ value of 0.5 (Figure 3(b)), the RNA sequence maintains a conformation similar to its original one (Figure 5(a)), while still demonstrating a run time drop of between 30% and 70%. The case where θ can be related to the sequence length was also considered, decreasing as the sequence length increases. Figure 3(d) demonstrates the results of employing such an approach, while the run time results can be seen in Figure 2. The value of θ was calculated using Equation 4[2];

$$\theta = 0.8936 \times n^{-0.1649} \tag{4}$$

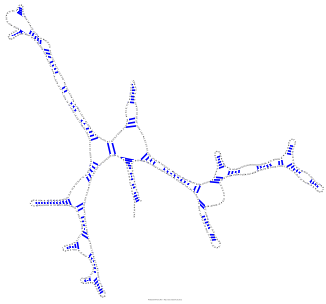
where n is the length of the given sequence.

In addition, future iterations of jViz.RNA could combine the Barnes-Hut algorithm with the original neighborhood based algorithm. Under this setup, once the sequence achieves force equilibrium under the Barnes-Hut algorithm, the neighborhood based algorithm would take over, and calculate the finer conformation of the sequence. This may result in diminished performance improvements compared to using only the Barnes-Hut algorithm, but overall giving aesthetically pleasing layouts for the RNA sequences in less time than the neighborhood based algorithm alone.

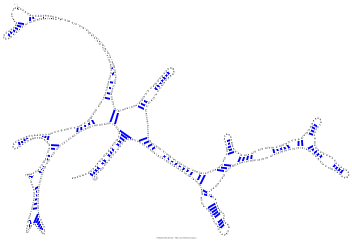
[2] The equation was developed using a regression that aimed to ensured that the θ value would be ≈ 0.41 for 117nt, and ≈ 0.25 for 2510nt.



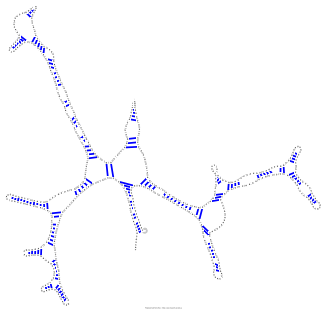
(a) The RNA sequence as drawn by the original algorithm



(b) The RNA sequence as drawn by the Barnes-Hut algorithm where $\theta = 0.5$



(c) The RNA sequence as drawn by the Barnes-Hut algorithm where $\theta = 1.0$

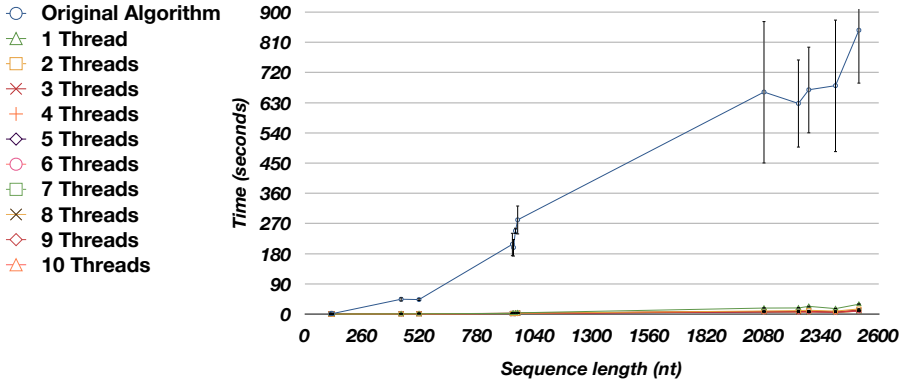


(d) The RNA sequence as drawn by the Barnes-Hut algorithm where θ varies ($\theta = 0.31$ for this sequence)

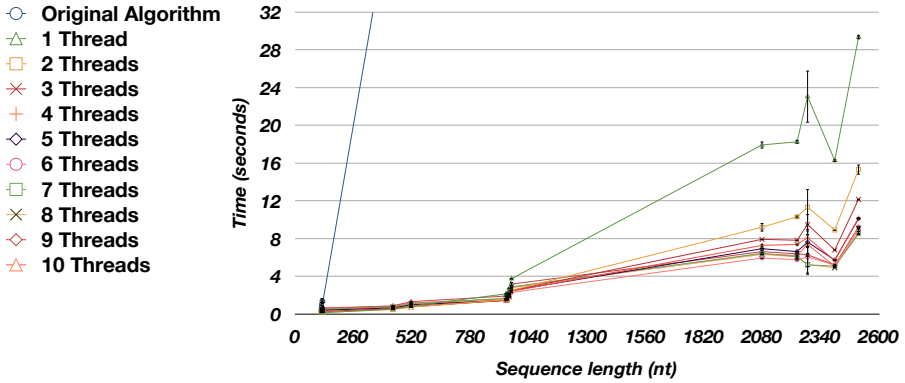
Fig. 3. The RNA sequence for *Tetrahymena thermophila* 26S ribosomal RNA fragment (with intron) (V01416) as drawn by jViz.RNA's original algorithm and the Barnes-Hut algorithm

Multithreading. Multithreading in jViz.RNA was implemented as a `Searcher` class using the `Runnable` interface Java™offers [11]. The `Runnable` interface allows a class to run on its own thread. The `Searcher` class would search through a subset of the nucleotides and check their interactions with all other nucleotides.

The results in Figure 4 show that as the number of threads increases, calculations for large sequences become faster and faster. However, even for large sequences there is a diminishing return as the number of threads increases. Furthermore, for medium and small size sequences the ideal number of threads is not the same as for large ones. For smaller sequences, an increase in the number of threads past four or five can actually cause a slowdown in the run-time. The reason for these observations is that although an increasing number of threads can allow CPU cores to process more data in parallel, they also require additional management. In addition, the number of available CPU cores would greatly effect these results since more cores would be able to handle more threads in parallel.



(a) The running times of the original, neighborhood based, algorithm versus multithreading

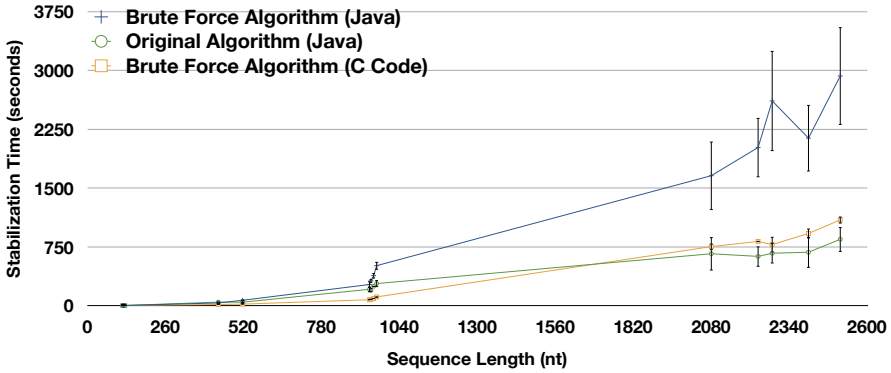


(b) The running times of the algorithm under different thread numbers

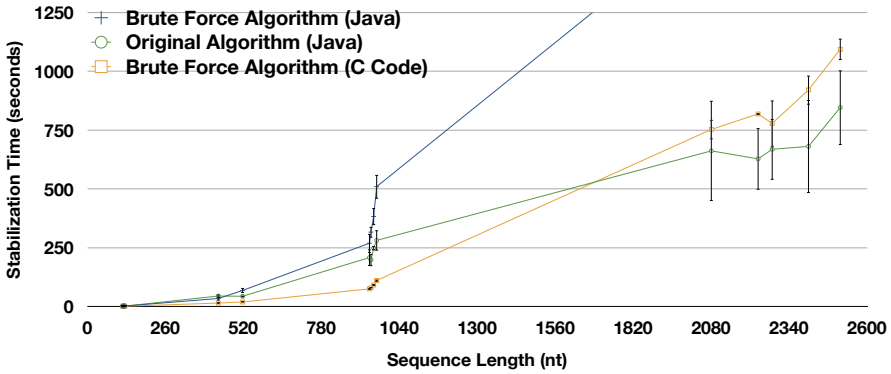
Fig. 4. The stabilization times (in seconds) for the different RNA sequences (as the number of nucleotides increases) using multithreading and the brute-force algorithm with different thread numbers versus the original algorithm emloyed by jViz.RNA. 99% confidence intervals are used as error bars

What is even more interesting is that even the brute force implementation, when employing multithreading, far outperforms jViz.RNA's current algorithm. There could be many factors accounting for this difference. However, it appears that dedicating more than one CPU core for the processing of the structure's layout drastically improves jViz.RNA's performance. With these results in mind, it is still worth noticing that the size of the sequence greatly effects the number of threads that allows for optimal performance insofar as run-time goes.

Native C Code. Keeping the overhead involved in calling C code from a Java^{TM}environment in mind, it was very interesting to find that even for the short sequences, the running times were faster when C based code was employed



(a) The running times of the original, neighborhood based, algorithm versus the brute-force algorithm using both Java and C code



(b) The running times of the original, neighborhood based, algorithm versus the brute-force algorithm using C code

Fig. 5. The stabilization times (in seconds) for the different RNA sequences (as the number of nucleotides increases) using the original algorithm, and the brute force algorithm implemented in both C and Java™code. 99% confidence intervals are used as error bars

than Java™code, as Figure 5 demonstrates. It isn't surprising to see that the brute-force C algorithm is outperformed by jViz.RNA's original algorithm as the sequences tested get larger. However, the fact that the C algorithm performs better over the smaller sequences shows that the overhead involved in calling C code does not contribute to a slowdown in performance. Had the overhead for calling C code accounted for a major slowdown, one would expect to see the C code perform slower than the Java™code over the smaller sequences, since the amount of time required for calculations is smaller, but the overhead involved in sending the data is fairly consistent. The results seen here, however, are just the opposite. For smaller sequences, the C code outperforms even the use of the

neighborhood based approach in Java™. Only when the number of nucleotides increases, and the use of the more sophisticated neighborhood approach gives the Java™code a competitive edge, is the Java™code performing better than the C based code.

Furthermore, the C code used to perform the brute force calculations consistently outperforms the Java™code for the brute force approach. The C code can be modified to implement the use of a neighborhood, and the results seen in Figure 5 demonstrate that the use of jViz.RNA's current algorithm in a C environment can improve performance for many, if not all, RNA sequences.

Additionally, an important point to make is that multithreading and the integration of C code do not distort the structures in any way since no approximation is entailed.

The main drawback of this method is the fact that for each different architecture, the dynamic library needs to be recompiled. However, this does not present a major problem since the calls to load the library and execute the native method can be put in blocks of defensive programming (`try/catch` blocks) and the original Java™code be used as default.

Structure Recall. Structure recall is a simple method for improving run-time, but has proven very effective. It is reminiscent of RNAViz's use of 'skeleton files' [4], but differs in its interaction with users since structure recall files are loaded automatically, if they are available. Structure recall employes flat files to store coordinate information regarding a sequence's nucleotides' positions.

Often groups of researchers are interested in a particular group of structures and would view a small set of structures many times. It is therefore only logical for jViz.RNA to avoid redundant calculations regarding structure conformation.

3 Discussion

This paper described several extensions and improvements for jViz.RNA, All of which provided promising results. The Barnes-Hut algorithm proved to improve the performance greatly, but some of the improvement needs to be traded in favor of aesthetic and clear layouts of the RNA sequences. On the other hand, software engineering approaches such as multithreading and C based execution of repetitive code, yielded very promising results both run-time wise and display wise, without the use of any approximation.

Furthermore, these approaches lend themselves to integration such that future iterations of jViz.RNA would use the Barnes-Hut algorithm, traversed in parallel by several threads, in a C environment for all force calculations, as well as integrate structure recall for sequences that have been previously viewed. Moreover, since multithreading benefits from having more cores, jViz.RNA could explore the possibility of General Purpose GPU (GPGPU) programming. GPUs are designed to perform calculations in parallel for large arrays of data, and are becoming increasingly popular with NVIDIA's introduction of the C based CUDA programming API [10], which would be ideal for jViz.RNA's required calculations

4 Conclusion

This paper describes four extensions to jViz.RNA which show potential for further experimentation and future inclusion in jViz.RNA. With these extensions, jViz.RNA overcomes the performance limitation for larger sequences. Additionally, as was mentioned in Section 1, jViz.RNA was also extended by the addition of functionality to process FASTA and RNAML files, allowing it to interact better with the input and output files of other programs and contributing to the standardization of RNAML.

References

1. Barnes, J., Hut, P.: A hierarchical O(N log N) force-calculation algorithm. Nature 324(4), 446–449 (1986)
2. Broccoleri, R.E., Heinrich, G.: An Improved Algorithm for Nucleic Acid Secondary Structure Display. Bioinformatics 4(1), 167–173 (1988)
3. Darty, K., Denise, A., Ponty, Y.: Varna: Interactive drawing and editing of the rna secondary structure. Bioinformatics 25(15) (2009)
4. De Risjk, P., De Wachter, R.: Rnaviz, a program for the visualisation of rna secondary structure. Nucleic Acids Research 25(22), 4679–4684 (1997)
5. Glen, E.: JVIZ.RNA - A Tool for Visual Comparison and Analysis of RNA Secondary Structures. Master's thesis, Simon Fraser University (2007)
6. Han, K., Byun, Y.: PseudoViewer3: generating planar drawings of large-scale RNA structures with pseudoknots. Bioinformatics 25(11), 1435–1437 (2009)
7. Hofacker, I.L.: Vienna RNA secondary structure server. Ivo L. Hofacker 31(13), 3429–3431 (2003)
8. Jossinet, F., Ludwig, T.E., Westhof, E.: Assemble: an interactive graphical tool to analyze and build RNA architectures at the 2D and 3D levels. Bioinformatics 26(16), 2057–2059 (2010)
9. Jossinet, F., Westhof, E.: The RnamlView Project. Institut de biologie moleculaire et cellulaire du CNRS
10. NVIDIA®. CUDA™Parallel Programming Made Easy (2011), `http://www.nvidia.com/object/cuda_home_new.html`
11. Oracle. Interface Runnable, `http://download.oracle.com/javase/1.4.2/docs/api/java/lang/Runnable.html`
12. Wiese, K.C., Glen, E.: jViz.Rna -a java tool for RNA secondary structure visualization. IEEE Transactions on NanoBioscience 4(3), 212–218 (2005)
13. Wiese, K.C., Glen, E.: jViz.Rna - An Interactive Graphical Tool for Visualizing RNA Secondary Structure Including Pseudoknots. In: CBMS, pp. 659–664. IEEE Computer Society (2006)