Partition Sums documentation

Matthijs G. A. van Dorp

Mathematics Institute, Utrecht University, P.O. Box 80010, 3508 TA Utrecht, The Netherlands
Institute for Theoretical Physics, Utrecht University, Leuvenlaan 4, 3584 CE Utrecht, The Netherlands

Introduction

This document provides information about the structure and functionality of the source code of the computer programs *Paramfit*, *AlgTest* and *Flopcount*. These programs have been used to obtain the results presented in the paper *Efficient calculation of partition sums in DNA/RNA hybridization* by Matthijs G.A. van Dorp, Rob H. Bisseling, and Gerard T. Barkema. The source code is available under the conditions of the GNU Lesser General Public License at http://www.math.uu.nl/people/bisseling/software.html .

General note

It should be noted that the source code itself also contains a lot of comments. This document should be considered as supplementary material, intended to illuminate the large-scale structure of each program, and the philosophy and reasoning of particular choices made.

Contents

1 Paramfit

1.1 Purpose

The Paramfit program aims to compute the parameters of the nearest-neighbor model. To accomplish this, it uses a guessed set of parameters to predict hybridization energies. The quality Q of the guess is given by a sum over squared errors. Varying the parameters, Paramfit determines the best fit by minimizing Q.

File name	function
paramfit.cpp	Contains the main functionality of the program. This includes the fitting procedure, and the function to determine Q.
sequence.h	Contains a class definition for DNA or RNA strands. This helps keep strand-related data organized.
sequence.cpp	Contains the experimental data from the literature.
partsum.h	Contains the algorithm to compute ΔG for any strand or pair of strands.
globdef.h	Contains some global definitions.

1.2 Model parameters to be fitted

There are a large number of parameters that may be varied. The program fits parameters for DNA/DNA, DNA/RNA and RNA/RNA hybridization. The parameters to be fitted are:

These parameters are given some rough initial values before the fit is run. The purpose and (bio-)physical meaning of the parameters will not be discussed here.

Parameter type	D/D	D/R	R/R	Total
Base pair doublet hybridization energy	10	16	10	36
Helix initiation energy	1	1	1	3
Penalty for terminal AU pairs	1	1	1	3
Penalty for internal AU pairs	1	1	1	3
Penalty for hairpins(*)	1	0	1	2
Total	14	19	14	47

Table 1: Number of parameters to be fitted. (*) It should be noted that the source code allows for many different hairpin parameters, for different sizes of the hairpin. However, because there was not much experimental data, the decision was made to use only two parameters, one for DNA and one for RNA hairpins. The source code, however, still contains artifacts of those extra parameters. These are not currently used anywhere in the program, and may be ignored.

1.3 Calculating Q repeatedly to fit the parameters

We may consider Q as a function of 47 parameters, whose value is obtained by summing the squared deviation for each data point. This deviation is the difference between the experimentally measured ΔG and the model-based prediction.

The approach is as follows. We may consider a small ball on a 47-dimensional landscape, rolling downhill because of a gradient. This gradient is found by numerically computing all 47 partial derivatives. The height of the landscape is equal to Q, and the minimum of Q is the lowest point.

By assigning a velocity to the ball, we aid it in finding the minimum quickly, in a dynamic way. That is to say, if the gradient is more or less the same several subsequent iterations, our step size gets larger and larger, which hopefully saves many different iterations. A drawback of the velocity is the ability of the ball to overshoot a valley of the landscape, and roll up the opposite hill. To prevent this, whenever such behavior is detected, the ball is returned to its former position, and the velocity is set to zero, so the ball can slowly start moving anew.

When the ball starts at zero velocity, but fails to improve at the first iteration, this algorithm breaks down. To help limit this danger, the time step *delta* is reduced in such a case, before trying again. This helps the fitting procedure cope with narrow canyon-like valleys. On the other side, if the landscape is very flat, it may take a long time for the ball to gain sufficient velocity. To reduce this, *delta* may be increased, such that we traverse the landscape a little faster. It turns out that those two situations are both encountered repeatedly when doing a fit.

1.4 Calculating ΔG

The value of ΔG for each experimental data point is predicted, using the parameters specified, by an algorithm implementing a many-state nearest-neighbor model. The code is contained in the file `partsum.h`. The precise details and working of the algorithm is too complex to describe here, the interested reader is referred to the paper mentioned on the first page for an introduction.

There are a few minor differences with respect to the algorithm described in the paper. For instance, the inclusion of a few additional parameters for hairpin and unzipping contributions. Their implementation is straightforward,

however, and the implementation still closely follows the algorithm outlined in the paper.

1.5 Results and output

The fitting continues until it is decided that the fit is optimal. This may be due to a user-set limit for convergence (meaning that Q decreases extremely little, or that the time step *delta* has become extremely small), or because the program detects machine-precision problems (which results in iterations alternating between two states indefinitely). As the fit terminates, the results are saved to a file, as well as printed to the screen.

On a final note, it is possible to do fits at different temperatures. While the version that is available online, only does calculations at a temperature of 37 degrees Celsius, the program may easily be altered to calculate a range of temperatures.

2 AlgTest

2.1 Purpose

The AlgTest program provides timings for a number of possible implementations of the algorithm based on the many-state nearest-neighbor model. Also, the time required for preprocessing is determined, as the implementations are timed independent of the construction times for arrays used by the different implementations.

File name	function
algtest.cpp	This is the main file, which contains the implementations to be tested, and the code used for timing their performance.
rng.h	This file contains an implementation of a Mersenne Twister-type random number generator, which is used to create random DNA strands.
globvar.h	Contains a few global variables.

2.2 Procedure and main parameters

The preprocessing steps are done repeatedly to determine the time required to carry out the necessary actions. Subsequently, all algorithms are run many times each, and the time required per run is determined. There are a few parameters available to tune this process:

Parameter	Role
REPBASE	The base number of repetitions. The number of times each algorithm is run, is determined by dividing this number by the cube of the strand length. Large numbers cause long calculations and accurate results, while small numbers are quick but are unreliable if the calculations are finished too quickly.
NUMSTR	The number of strands created for each data point (a data point corresponds to a certain strand length). Because some strands may be harder for the algorithm to cope with than others, it is good to use a decent number of strands, such that the results are not too strongly dependent on a single outlier.
MAXLEN	No lengths larger than this value will be tested. Very large lengths will cause the number of repetitions to decrease below unity. To counter this effect, it is required that for any timing, at least 10 repetitions are done. In general, lengths larger than about 800 will take a disproportionate amount of time to compute.
NUM_RUNS	To help reduce the effect of temporary decreases in processor performance, it is possible to do everything a number of times. This essentially runs the program multiple times, and averages the outcomes.

2.3 Results and output

The averaged timings are written to a text file in Mathematica-compatible array formatting, such that the results may easily be plotted. The output has the form $\{\{l_1, t_1\}, \ldots, \{l_n, t_n\}\}$, for length l and computation time t.

3 Flopcount

3.1 Purpose

The Flopcount program is a heavily modified version of the AlgTest program, intended to provide a good indication of the properties of the different implementations. To achieve this, it counts the number of multiplications between doubles, the number of writes and lookups into arrays, the number of function calls, et cetera.

File name	function
algtest.cpp	This is the main file, which contains the code to generate DNA strands, and also does the preprocessing steps.
algtestc.cpp	This contains modified versions of two implementations, which are similar to the originals except that they also carry along a list of counters. For each operation, the corresponding counters are incremented, such that the list contains the counts for various types of actions upon completion of the algorithm.
rng.h	This file contains an implementation of a Mersenne Twister-type random number generator, which is used to create random DNA strands.
globvar.h	Contains a few global variables.

3.2 Procedure and parameters

A large number of different strands is generated for each data point (note that we vary the strand length, so we do calculations for a range of lengths). The average counts are computed for each type. The parameters NUMSTR and MAXLEN available, are identical in function to those described above for the AlgTest program.

3.3 Results and output

The resulting counts are written to a text file. The output format is identical to that of the AlgTest program.