GAGA Users Guide manuscript No.
(will be inserted by the editor)

User Guide, GAGA Version 1.2.0

GPU Accelerated Greedy Algorithms for Compressed Sensing

Jeffrey D. Blanchard · Jared Tanner
with contributions from Ke Wei and Rodrigo Mendoza-Smith

September 2015

1 Introduction

GPU Accelerated Greedy Algorithms (GAGA) for compressed sensing is a software package for massive parallelization of greedy algorithms on CUDA capable graphics processing units (GPU). The software consists of a testing environment and CUDA-C implementations of numerous greedy algorithms to be compiled as a Matlab toolbox for ease of use[1]. The list of algorithms include: Normalized Iterative Hard Thresholding [8], Hard Thresholding Pursuit [10], a hybrid implementation of CoSaMP [14] and Subspace Pursuit [9] denoted CSMPSP [4], Conjugate Gradient Iterative Hard Thresholding (CGIHT) [6], CGIHT restarted [6], CGIHT projected [6], Fast Iterative Hard Thresholding (FIHT) [6], and 1-ALPS(2) [12]. The following additional algorithms specific for expander[2] sensing matrices are also included: Expander Recovery (ER) [11], Sparse Matching Pursuit (SMP) [2], Sequential Sparse Matching Pursuit (SSMP) [1], Parallel LDDSR [15,13], Serial ℓ_0 [13], and Parallel ℓ_0 [13]. In addition to the aforementioned algorithms are three of historic interest: Thresholding (with steepest descent projection), Thresholding (with conjugate gradient projection), and Iterative Hard Thresholding [7]. Each algorithm returns a vector, $\hat{x} \in \mathbb{R}^n$, with at most k nonzero entries, which is an approximate minimizer of $\|y - Ax\|$ for $A \in \mathbb{R}^{m\times n}$. A full comparison of the performance of NIHT, HTP, and CSMPSP appears in [4]. An extension of that work including CGIHT, CGIHT restarted, CGIHT projected, and FIHT appears in [6,5]. Comparisons of the algorithms for sparse matrices are presented in [13].

This majority of this user guide for GAGA 1.2.0 includes a duplication of prior GAGA user guides with some minor modifications and the addition of Sec. 6 detailing the changes to the new version.

1.1 Requirements

This software requires an NVIDIA CUDA capable GPU with compute capability 2.0 or higher, a valid installation of CUDA-C 3.0 or higher, the appropriate nvcc compiler, and a valid installation of Matlab. There are no other requirements. This software **does not** require the parallel processing toolbox.

1.2 Installation

Download the software as a tar file and unpack the software; this will build the GAGA directory. In order to compile the software you must identify the proper path definitions through the makefile system. The Makefile in the folder

JDB was a National Science Foundation International Research Fellow at the University of Edinburgh under award NSF OISE 0854991 and is supported by NSF DMS 1112612. JT was supported by the Leverhulme Trust and an Nvidia Professor Partnership.

J.D. Blanchard
Department of Mathematics and Statistics, Grinnell College, Grinnell, IA 50112
E-mail: jeff@math.grinnell.edu

J. Tanner
Mathematics Institute, University of Oxford, 24-29 St. Giles', Oxford, OX1 3LB, UK
E-mail: tanner@maths.ox.ac.uk

[1] Direct access through C++ compiled executables can be accomplished by extracting the main functions.

[2] Here expander matrices are sparse matrices with a fixed number of nonzeros per column structured so that any set of k columns has nearly the maximum possible number of neighbours.

GAGA/gaga is generic and can be left unchanged if you are operating in a Linux environment on a 64 bit system. Otherwise, you will have to change this file for your specific architecture. These changes are minor and can be templated from the GAGA/CudaMATLAB/Makefile. Prior to compiling the software make the following changes to the Makefile system.

a. Alter GAGA/gaga/Makefile to the appropriate architecture. Skip this step if you are using a 64 bit Linux environment.
b. The file GAGA/config directs to the main Makefile in GAGA/CudaMATLAB and defines the paths to the installations of GAGA, CUDA, and Matlab. Make the following modifications to GAGA/config:
   (i) Define GAGA as the location to the GAGA installation.
   (ii) Define CUDAHOME as the location of the CUDA installation.
   (iii) Define MATLAB as the location of the Matlab installation and correct version.
   (iv) Define GPUARCH as the architecture type for your Nvidia GPU (e.g. sm_20 for C2050 or sm_30 for K10).

The software package is now ready to be compiled. From the directory GAGA/gaga, simply enter

```
make clean_all
make all
```

on the command line to build three Matlab executable GPU functions, `gaga_dct`, `gaga_smv`, and `gaga_gen`. The user may call these functions directly, but a parent function, `gaga_cs` is typically used to call these GPU functions based on a matrix class input. The installation can be tested by starting Matlab in GAGA/gaga and calling the script `test_gaga.m`, `test_cgiht.m`, and `test_gaga_expander.m`. The CPU direct Matlab variant need not be compiled, requiring only the following modification to GAGA/gaga_matlab/greedyheader.m:

(i) Define Location as the location to the gaga_matlab installation.

Intermediate values can be displayed by invoking the `VERBOSE` option, and the location of any CUDA errors can be readily determined by invoking the `SAFE` option. Both options can be activated by removing the appropriate comments in `greedyheader.cu`.

2 The GPU Function: `gaga_cs`

For a detailed discussion of the functions, see [3]. The software has one main function `gaga_cs` which is overloaded to have two main functionalities: testing and application. The algorithm testing variant generates a random problem to test a specified algorithm and matrix class and returns algorithm performance characteristics. The application variant returns an estimated sparse solution to a specific problem passed from the Matlab workspace with the sparse solution obtained by applying a user specified algorithm along with the measurement matrix, measurements, and algorithm options. The overloading is determined by the number of output arguments when the function is called. It is important to note that you must always provide a full set of output variables when calling any of the functions in GAGA.

When used to test algorithm performance on randomly generated problems[3], the function automatically generates the problem using options *matrixEnsemble* and *vecDistribution* to specify the distribution of the random $m \times n$ measurement matrix and random k nonzeros in the sparse vector measured. The algorithm then solves the generated problem and returns to the Matlab workspace the following algorithm performance characteristics: errors, times, iterations, support identification, convergence rate, and the output of the algorithm. The function also records more detailed information to a date stamped text file. The input and output arguments for executing these functions are given in Tab. 1.

	Random Problem	
matrix class	output	input
gen	$[errors, times, iterations, checkSupport, convRate, \hat{x}]$	`gaga_cs`('alg','gen',$k,m,n,options$)
smv	$[errors, times, iterations, checkSupport, convRate, \hat{x}]$	`gaga_cs`('alg','smv',$k,m,n,p,options$)
dct	$[errors, times, iterations, checkSupport, convRate, \hat{x}]$	`gaga_cs`('alg','dct',$k,m,n,options$)

Table 1 Input arguments and output for generating and solving a random problem with `gaga_cs`. (For *smv*, p denotes the number of nonzero entries per column.) The *options* argument may be omitted in which case all variables take on their default values.

The function `gaga_cs` is also capable of taking a problem directly from Matlab, passing it to the GPU, and returning the solution to the Matlab workspace. In this case, there is no data written to a text file and the output to Matlab is

[3] The random sparse matrix generator is designed to be fast for $p \ll m$, and outside this regime can fail to construct the sparse matrix appropriately, resulting in a warning and the function terminating.

the approximate solution vector, iterations, and convergence rate. The function always requires inputs specifying the algorithm to be used to recover the sparse vector, the type of the matrix that is being passed, the number of nonzeros k in the output vector, and the vector of measurements y. When passing a general matrix, class *gen*, the matrix A is passed directly. Sparse matrices, class *smv*, are passed in coordinate list (COO) format of *rows*, *cols*, and *values* and for expander algorithms requires the number of nonzeros per column to be fixed. For discrete cosine transform matrices, class *dct*, the measurement matrix is passed by specifying its size $m \times n$ and the subset *rows* of the full discrete cosine transform matrix.

	Direct Problem	
matrix class	output	input
gen	$[\hat{x}, iterations, convRate]$	`gaga_cs`('alg','gen', $y, A, k, options$)
smv	$[\hat{x}, iterations, convRate]$	`gaga_cs`('alg','smv', $y, rows, cols, values, k, options$)
dct	$[\hat{x}, iterations, convRate]$	`gaga_cs`('alg','dct', $y, rows, k, m, n, options$)

Table 2 Input arguments and output for solving a direct problem with `gaga_cs`.

Values of *alg* are admissible for all matrix classes are the following strings: 'ThresholdSD', 'ThresholdCG', 'IHT', 'NIHT', 'HTP', 'CSMPSP', 'CGIHT', 'CGIHTprojected', 'FIHT', and 'ALPS'. Expander algorithms designed only for the matrix class 'smv' can be called with the *alg* strings: 'er', 'smp', 'ssmp', 'parallel_lddsr', 'serial_l0', and 'parallel_l0'. Options are set using the `gagaOptions` function by passing pairs indicating a variable name followed by its specified value (which may be an integer, float, or string). For example,

```
>> options = gagaOptions('tol',0.0001,'maxiter',400,'vecDistribution','gaussian');
>> [err,tim,itr,supp,cnv,xout] = gaga_cs('NIHT','dct', k, m, n, options);
```

will generate a random problem with a $m \times n$ subsampled DCT matrix and a vector with k nonzeros from $\mathcal{N}(0,1)$, and then solve the problem using NIHT, with stopping criteria depending on a tolerance of 0.0001 and a maximum of 400 iterations. Options should be set using `gagaOptions` for typecasting and necessary ordering. A complete list of admissible options is presented in Tab. 3.

3 The CPU Function: `gaga_matlab_cs`

3.1 GAGA_matlab

The software package includes a partial mirror copy of the Matlab GPU software implementation written exclusively in Matlab using only the CPU. The primary purpose of this duplication of the code is to compare the performance of the GPU enabled functions to CPU only equivalent, and as a secondary feature to serve as a template to the CUDA-C code for those unfamiliar with this language. This software is included in the directory GAGA/gaga_matlab with the main function `gaga_matlab_cs` having the same input and output as given in Tab. 1 for random problem generation and evaluation. The CPU only function also records the pertinent information in date stamped text files whose names include `matlab`. The Matlab only variant is capable of having a measurement matrix and vector passed, but in that case the algorithms should be directly called as the Matlab function from the directory GAGA/gaga_matlab/algorithms. Many, but not all algorithms available in gaga are also available in gaga_matlab; for a complete list see GAGA/gaga_matlab/algorithms.

4 Matlab Scripts for Testing

The software includes script files for easily testing that the installation and compilation has succeeded.

- In GAGA/gaga the script `test_gaga.m` tests `gaga_cs` for each of the included algorithms with random problem generation for each matrix class both with and without noise, as well as with the timing option on, and with problems passed directly from the Matlab workspace.
- In GAGA/gaga_matlab the script `test_gaga_matlab.m` tests `gaga_matlab_cs` for each of the included algorithms with random problem generation for each matrix class both with and without noise, as well as with the timing option on.

Admissible options	
variable name	value type and default
'tol'	specifying convergence rate stopping conditions, positive float, default 0.001.
'maxiter'	maximum number of iterations, positive integer, default 300 for HTP and CSMPSP, 5000 for other algorithms.
'matrixEnsemble'	distribution of the nonzeros in the measurement matrix (not used for *dct*) for the test problem, string options: 'binary' for $\pm 1/\sqrt{p}$ with equal probability (default for *smv*), 'gaussian' for Normal $\mathscr{N}(0,1)$ (only valid for *gen*, default for *gen*), 'ones' for all nonzeros all equal to $1/\sqrt{p}$ (only valid for *smv*), and 'expander' for all nonzeros all equal to one (only valid for *smv*).
'vecDistribution'	distribution of the nonzeros in the sparse vector for the test problem instance, string options: 'binary' for ± 1 with equal probability (default), 'gaussian' for Normal $\mathscr{N}(0,1)$, and 'uniform' for uniform from the interval zero to one, $\mathscr{U}(0,1)$.
'band_percentage '	a float between $(0,1)$ which sets 'band_percentage ' fraction of the nonzero entries in the sparse vector for the test problem to be equal, drawn from the same 'vecDistribution'. Currently only works for 'matrixEnsemble' set to *smv*.
'seed'	seed for random number generator, unsigned int, default `clock()`.
'numBins'	number of bins to use for order statistics, positive integer, default to $\max(n/20, 1000)$
'kFixed'	flag to force the k used in the problem generate to be that specified, string options: 'off' (default) and 'on'.
'noise'	level of additive normally distributed noise as a fraction of the $\|Ax\|_2$, non-negative float, default to 0.
'convRateNum'	number of the last iterations to use when calculating average convergence rate, positive integer, default 16.
'gpuNumber'	identify GPU to be used, non-negative integer, default to 0.
'threadsPerBlockn'	number of threads per block for kernels acting on vectors of length n, positive integer, default to $\min(n, max\_threads\_per\_block)$.
'threadsPerBlockm'	number of threads per block for kernels acting on vectors of length m, positive integer, default to $\min(m, max\_threads\_per\_block)$.
'threadsPerBlocknp'	number of threads per block for kernels acting on vectors of length $n \cdot p$ (for *smv*), positive integer, default to $\min(np, max\_threads\_per\_block)$.
'threadsPerBlockBin'	number of threads per block for kernels acting on vectors of length *numBin*, positive integer, default to $\min(numBins, max\_threads\_per\_block)$.
'timing'	indicates that times per iteration should be recorded, string options: 'off' (default) and 'on'. (only valid for 'alg' equal to 'NIHT' and 'HTP'.)
'alpha'	specifying fraction $(1-\alpha)$ of bins counted in early support set identification steps, float between $(0,1)$, default to 0.25 which counts 75% of the bins initially. (only valid with 'timing' set to 'on'.).
'supportFlag'	method by which the support set is identified, integer options (only valid with 'timing' set to 'on'): 0 (default) for dynamic binning where binning is conducted only when the support set could have changed, 1 for binning at every iteration, 2 for using `thrust::sort` to find the largest entries when the support set could have changed, and 3 for using `thrust::sort` at every iteration.
'restartFlag'	flag for use with CGIHT to choose to restart the conjugate gradient projection when the support set changes 'on' or to always use previous search direction history regardless of support changes 'off' (default).
'projFracTol'	value for use with CGIHTprojected to indicate the restarting tolerance for the relative fraction of the residual projected onto the current support, positive float, default 3.
'l0_thresh'	Only admissible for 'matrixEnsemble' set to *smv* with p nonzeros per column and the *alg* string set to 'serial_l0' or 'parallel_l0'. An integer taking values from 1 to p specifying a vertex should be updated when it would results in a reduction of the residual's number of nonzeros by at least 'l0_thresh'.

Table 3 Optional arguments for `gaga_cs` set using gagaOptions.

- In GAGA/gaga_for_CS_timings the script `do_all.m` will generate representative data from all functions and all algorithms, analyze the data and create figures and plots similar to those in [3]. The `do_all.m` script is currently set to test smaller values of n due to time constraints. Running `do_all.m` as included in the software package will take approximately 8 hours on a C2050. Running the full set of n in [3] using `do_all.m` will take approximately 10 days.
- The following directories include scripts `do_all.m` which will generate and process the data for the plots and tables presented in the corresponding article: GAGA/PCGACScode [4], GAGA/CGIHTcode [6] and GAGA/EXPl0Ecode [13].

5 Licensing and Copyright

GAGA is copyright © Jeffrey D. Blanchard and Jared Tanner, 2010-2016. The software is available for research, educational, and non-commercial use according to the GAGA License which is included in the software distribution, available at the web page `www.gaga4cs.org/License`, and provided at the end of this document in the Appendix. The GAGA License applies to all versions of GAGA including prior and future versions.

6 Changes and Additions

6.1 GAGA 1.1.0 changes and additions

The second formal release of GAGA includes changes to algorithms, matrix-vector multiplications, and source code formatting. For most users, the changes result in the availability of more algorithms and faster dense matrix - sparse vector product. For those building off the code for their research, the changes may require minor reformatting of the code if the new version is adopted.

Additional Algorithms To coincide with the authors' announcement of a new greedy algorithm, Conjugate Gradient Iterative Hard Thresholding (CGIHT) [6], GAGA 1.1.0 includes the addition of CGIHT. Associated with the empirical testing in [6], two additional algorithms are included Fast Iterative Hard Thresholding (FIHT) and 1-ALPS(2) [12].

- CGIHT: this algorithm combines the flexible support changes of NIHT with the improved convergence rates of the projection methods by performing subspace restricted conjugate gradient updates when the support remains the same. When the support set changes, the user decides if the algorithm should restart the conjugate gradient search with a steepest descent step or continue to use the previous search direction information. To do so, this algorithm takes an additional option 'restartFlag '.
- CGIHTprojected: this is CGIHT using a subspace projection restarting condition rather than a support set evolution restarting condition. This algorithm takes an additional option 'projFracTol '.
- FIHT: FIHT is an adaptation of 1-ALPS(2) which tailors Nesterov's line search optimization to the sparsity setting.
- ALPS: ALPS is the version of 1-ALPS(2) announced in [12].

Dense matrix - sparse vector multiplication The *gen* matrix-vector multiplication was updated to take advantage of the sparsity in the vector being multiplied. The matrix-vector multiplications in the greedy algorithms are all with vectors which have been restricted to a given support set and are therefore sparse. This change is seamless to users, but users should be aware of the change. When using the *gen* matrix-vector product, if the vector is not sufficiently sparse ($\delta\rho \approx 0.7$), the original version from GAGA 1.0.0 may be faster.

Matlab Scripts for Testing The software includes script files for reproducing the GAGA generated data in [4,6,5].

- In GAGA/PCGACScode the script `do_all.m` will generate representative data from all algorithms and all matrix ensembles, analyze the data and create figures and plots similar to those in [4]. The `do_all.m` script is currently set to test the smaller value $n = 2^{10}$ and a smaller set of values of $\delta = m/n$ due to time constraints. Running `do_all.m` as included in the software package will take approximately 8 hours on a single C2070. Running the full set of n in [4] using `do_all.m` will take approximately 15 days.
- In GAGA/CGIHTcode the script `do_all.m` will generate representative data from all algorithms and all matrix ensembles, analyze the data and create figures and plots similar to those in [6,5]. The `do_all.m` script is currently set to test the smaller value $n = 2^{10}$ and a smaller set of values of $\delta = m/n$ due to time constraints. Running `do_all.m` as included in the software package will take approximately 8 hours on a single C2070. Running the full set of n in [6,5] using `do_all.m` will take approximately 30 days.

Stopping Criteria When solving problems from the model $y = Ax + e$ for some misfit/noise vector $e \in \mathbb{R}^m$, the algorithms frequently cycle between multiple limit points often with distinct support sets. This can cause the algorithms to continue to iterate until the convergence rate stopping condition is in effect. The algorithms now check if the current residual norm is identical (to single precision) to one of the previous 15 residual norms. If so, a cycle is declared and the algorithm terminates. This stopping condition is observed to be empirically robust, have no impact on noiseless tests from the model $y = Ax$, and significantly improve computational time for the additive noise model $y = Ax + e$.

6.2 GAGA 1.2.0 changes and additions

The third formal release of GAGA includes changes to algorithms and sparse matrix construction. For most users, the changes result in the availability of more algorithms and the removal of a bug when dealing with sparse matrices. For those building off the code for their research, the changes may require minor reformatting of the code if the new version is adopted.

Additional Algorithms To coincide with the authors' announcement of a new greedy algorithms for expander sensing matrices, Serial-ℓ_0 and Parallel-ℓ_0, GAGA 1.2.0 includes the addition of these algorithms. Associated with the empirical testing in [13], four additional expander algorithms are included: Expander Recovery (ER) [11], Sparse Matching Pursuit (SMP) [2], Sequential Sparse Matching Pursuit (SSMP) [1], Parallel LDDSR [15, 13].

Expander specific sparse matrix updates The construction of sparse matrices now checks to ensure there are no rows without any nonzeros, and also checks for and resolves potential out of memory access in the probability zero event of uniform $[0, 1]$ numbers being equal to 1. The aforementioned instances occur very infrequently, but have been observed. When a sparse matrix problem instance is passed through matlab the number of nonzeros in the first column will be computed and for expander algorithms (which require a fixed number of nonzeros per-column) it will be assumed all columns have this number of nonzeros.

References

1. Berinde, R., Indyk, P.: Sequential sparse matching pursuit. In: Communication, Control, and Computing, 2009. Allerton 2009. 47th Annual Allerton Conference on, pp. 36–43. IEEE (2009)
2. Berinde, R., Indyk, P., Ruzic, M.: Practical near-optimal sparse recovery in the l1 norm. In: Communication, Control, and Computing, 2008 46th Annual Allerton Conference on, pp. 198–205. IEEE (2008)
3. Blanchard, J.D., Tanner, J.: GPU accelerated greedy algorithms for compressed sensing. Mathematical Programming Computation **5**(3), 267–304 (2013)
4. Blanchard, J.D., Tanner, J.: Performance comparisons of greedy algorithms in compressed sensing. Num. Lin. Alg. Appl. **22**(2), 254–282 (2015)
5. Blanchard, J.D., Tanner, J., Wei, K.: Conjugate gradient iterative hard thresholding: Observed noise stability for compressed sensing. Oxford Numerical Analysis Technical Report **14**(03), http://eprints.maths.ox.ac.uk/1806/ (2014)
6. Blanchard, J.D., Tanner, J., Wei, K.: Conjugate gradient iterative hard thresholding for compressed sensing and matrix completion. Information and Inference: A Journal of the IMA p. in press (2015)
7. Blumensath, T., Davies, M.E.: Iterative hard thresholding for compressed sensing. Appl. Comput. Harmon. Anal. **27**(3), 265–274 (2009)
8. Blumensath, T., Davies, M.E.: Normalised iterative hard thresholding; guaranteed stability and performance. IEEE Selected Topics in Signal Processing **4**(2), 298–309 (2010)
9. Dai, W., Milenkovic, O.: Subspace pursuit for compressive sensing signal reconstruction. IEEE Trans. Inform. Theory **55**(5), 2230–2249 (2009)
10. Foucart, S.: Hard thresholding pursuit: an algorithm for compressive sensing. SIAM Journal on Numerical Analysis **49**(6), 2543–2563 (2011)
11. Jafarpour, S., Xu, W., Hassibi, B., Calderbank, R.: Efficient and robust compressed sensing using optimized expander graphs. Information Theory, IEEE Transactions on **55**(9), 4299–4308 (2009)
12. Kyrillidis, A., Cevher, V.: Recipes on hard thresholding methods. In: 4th IEEE Inter. Workshop on CAMSAP, pp. 353–356 (2011)
13. Mendoza-Smith, R., Tanner, J.: Expander ℓ_0-decoding. arXiv preprint arXiv:1508.01256 (2015)
14. Needell, D., Tropp, J.: CoSaMP: Iterative signal recovery from incomplete and inaccurate samples. Appl. Comp. Harm. Anal. **26**(3), 301–321 (2009)
15. Xu, W., Hassibi, B.: Efficient compressive sensing with deterministic guarantees using expander graphs. In: Information Theory Workshop, 2007. ITW'07. IEEE, pp. 414–419. IEEE (2007)

Appendix

GAGA License

In order to use the GAGA library, or any of its constituent parts, a user must agree to abide by a set of conditions of use. The library is available at no cost for "Internal" use. "Internal" use of the library is defined to be use of the library by a person or institution for academic, educational, or research purposes under the following conditions. Any use of the library implies that these conditions have been understood, and that the user agrees to abide by all the listed conditions.

1. The copyright of the GAGA library, its constituent packages and documentation, and the function names gaga_cs, gaga_dct, gaga_smv, gaga_gen, gaga_matlab_cs, gaga_matlab_dct, gaga_matlab_smv, and gaga_matlab_gen are protected by this license and my not be reused.
2. The library, or any modification thereof, may not be sold or supplied, nor embedded within other software products which are subsequently sold or supplied, to a third party.
3. The library comes with no guarantee, expressed or implied, that it is suitable for any specific purpose nor that it is free of error. It should not be relied on as the basis to solve a problem whose incorrect solution could result in injury to person or property. If the library or any of its constituent parts is employed in such a manner, it is at the users' own risk and the authors disclaim all liability for such use.
4. The library is distributed as is. In particular, no maintenance, support, troubleshooting, or subsequent upgrade is implied.
5. The use of GAGA or its constituent packages must be acknowledged, in any publication which contains results obtained with the library or any of its parts. A citation to the paper
   J. D. Blanchard and J. Tanner, *GPU Accelerated Greedy Algorithms for Compressed Sensing,*
   Mathematical Programming Computation, 5(3): 267-304, 2013
   is suitable.
6. Users of the library are encouraged to submit examples of problems they have solved or algorithms they have added to info@gaga4cs.org. These examples or algorithms may subsequently be included in GAGA without compensation to the user.
7. It is the responsibility of the licensee to ensure that each user of the library is aware of, and agrees to abide by, all the conditions in this License.

A commercial license and conditions for commercial or non-academic use of the GAGA library or any of its constituent parts must be negotiated separately with

Jeffrey Blanchard
GAGA for CS
Grinnell College
Grinnell, IA 50112, USA
Phone: +1 (641) 269 - 3304
Fax: +1 (641) 269 - 4984
email: commercial@gaga4cs.org

to whom all commercial inquiries should be addressed.
A copy of this license is included in the distribution package.