Mathematical Software (TOMS)

Algorithms Policy

Matters of Content

A contribution should be in the form of an algorithm, comparison, certification, remark, or translation.

Algorithms

ACM Algorithms are published to make the fruits of software research accessible to as wide an audience as possible. An algorithm must either provide a capability not readily available or perform a task better in some way than has been done before. "Better" can mean anything from improved reliability or efficiency to more effective packaging. In all cases, an algorithm must be of lasting value and must represent a substantial contribution in terms of the amount of work or the originality required for its creation. In the case where the new algorithm is claimed to be an improvement over currently available codes supporting evidence must be included in the submission. Efficiency claims need to be supported on a wide range of practical problems.

An algorithm submission may consist of either one or two papers. There is always an *algorithm paper* which describes the code. Optionally, a second (*companion paper*) which describes the underlying algorithms in more detail, may also be submitted for review either to ACM TOMS or, in exceptional circumstances, to another of the ACM Transactions (see the Submission Formalities section for more details). In this case, both papers will be published simultaneously. The algorithm paper should give a brief description of what the software does and pertinent information on usage and maintenance. It should not duplicate information in another paper or in the algorithm sources; neither should it contain information that would more usually be found in a users' manual.

An ACM algorithm must be complete, portable, well documented, and well structured. The meaning of these terms is clarified below.

Completeness

A submission consists of all the code and test data necessary for the effective use and testing of the algorithm implementation by a large section of its intended audience.

The software part of the submission should consist of

The Algorithm Implementation:
In the vast majority of cases, this consists of one or more files that constitute a stand-alone implementation of the algorithm along with any pre-processors that are used. Typically, it should be possible for a user to incorporate this software into a larger program. The only exceptions to this would be in the case of explicitly standalone software, for example, a source code transformer. In particular, any graphical user interface provided to allow the easy input of data and the presentation of results should be considered as belonging to the testing material part of the submission.

Third Party Supporting Code:
This consists of any supporting software that is either called by the algorithm code or by the testing code and has not been written by the author. It is expected that all such supporting code will be in the public domain; code subject to more restricted use may be used if it is available from an established source for a nominal fee *and* there is no nearly equivalent code available in the public domain.

Documentation:
(see also the documentation section below) Authors are required to submit machine readable documentation that provides detailed information about the package and its use. In particular, it should be noted that detailed descriptions of user provided parameters, example programs, associated data and results, etc. should generally **not** appear in the accompanying paper.

Example Driver:
This should consist of a short test driver and should illustrate the use of the algorithm to solve a simple test problem. This part of the submission should include any test data that is required by the driver along with a model set of results.

Note: *the test driver must not require any data to be input interactively.*

Testing Material:
This should consist of one or more driver programs along with any relevant files containing required data and expected results. These programs should exercise all the main features of the implementation.

Command File:
This file should detail how the various package components are assembled to execute both the example driver and all the test material. This should take the form of either a Unix style *Makefile* or a *.bat* file with all references to externally required data (for example, compiler names, compiler flags, etc.) parameterized. In exceptional circumstances a detailed set of textual instructions may be used.

All the material described above is subject to the refereeing process and will form the material to be published in the *Collected Algorithms of the ACM (http://calgo.acm.org/)* (CALGO).

Portability

It must be possible to move the code in machine-readable form from one platform/compiler/interpreter to another with, at worst, only very minor, well-documented changes.

As initial evidence of portability the author may

1. include evidence that the package executes successfully on three different platform/compiler/interpreter combinations where at least two of these runs should be on different hardware; and/or
2. include evidence that the software has been checked for portability and/or conformance to an appropriate standard. This could take the form of either a special verifier tool or the use of a compiler with appropriate standard checking flags set (for example, -ansi with the Gnu C compiler, gcc).

In any case all submitted code should adhere to the latest appropriate language standard where one exists. In particular:

Fortran: ISO/IEC 1539-1:2004 [1]

C : ISO/IEC 9899:1999 [2]

C++ : ISO/IEC 14882:2003 [3]

Ada : ISO/IEC 8652:1995 [4]

Machine Dependencies

Machine-Dependent Constants

Where possible, standard conforming enquiry functions should be used to obtain values for any machine dependent constants (for example, the machine epsilon, the largest positive real number, etc.) required by the software. Where constants are unavailable in a particular language, standard names and definitions should normally be used [5].

Constants not available as built-in enquiry functions should be collected together and clearly identified by comments. Authors are encouraged to collect such constants into a separate function in order to simplify identification and changes.

Machine-Dependent Code

Machine-dependent routines may be used provided

1. They are clearly specified, limited in function, and either necessary or a substantial improvement over equivalent portable procedures.
2. Where possible, portable versions are also submitted to facilitate installation and testing, and to confirm the reader's understanding of the specifications. If portable versions are not provided, then tested versions for at least three different computers must be provided.
3. A portable test program, which exhaustively tests each machine-dependent module of the package, is provided.

Documentation
Each module of the code, including test drivers, must be adequately commented. Comments should include the purpose of the module, definitions of all arguments passed through the calling sequence, through global variable declarations, or obtained via input, and comments setting off and explaining major parts of the code. Comments defining machine dependencies should be gathered together and clearly flagged. A comment can simply point to the place where full comments are given in cases where large comment blocks would otherwise be repeated. An alphabetical list of internal variables, together with how they are used, is highly recommended.

Machine-readable documentation giving more detailed information about the package than can be published is encouraged. This material may be prepared in a number of ways

1. Tex/Latex: both a pdf version and the original Tex/Latex sources along with all dependency files (for example, .bib and image files) should be provided.
2. Word.
3. HTML: this should contain only relative paths for linking purposes.

Structure
Code should be organised so that it is easy to use and modify, and yet is flexible enough to be useful for most problems in the problem area covered. Indentation should be used to identify loops, compound statements and blocks, and continuation of a statement onto another line. If integers are used for statement labels, they must be strictly increasing and should be far enough apart to allow for future modifications. If the space required by an array is highly problem dependent, then that array should have a variable dimension.

All contributions will be made available in their entirety via the *Collected Algorithms from ACM* (http://calgo.acm.org/) (CALGO) website and the ACM Digital Library (https://portal.acm.org/dl.cfm).

Other Algorithm-type Submissions

Other, often short, submissions that refer to numbered algorithms that have been previously published in CALGO are also acceptable. The requirements of completeness, portability, quality of documentation and structure, described above in the context of a full algorithm paper, also apply, where relevant, to these submissions.

Remarks
This is a brief report on a previously published algorithm and is concerned with correcting or modifying the code. The paper should describe the reasons for the proposed changes and, in the case of simple modifications to the software, details of the alterations made. The revised software should be

submitted along with evidence that illustrates the original problem that has been corrected or improved (for example, particular test data and results).

Translations
A translation may report either the conversion of an existing algorithm into a different high-level language or the provision of machine-dependent modules for a platform not covered by the existing algorithm. A translation will be considered only if it is a translation of an algorithm that still represents the current state of the art.

Comparisons
A comparison is a report on the relative merits and features of highly similar software packages for a specific subject area. This practical study would include reporting and interpreting various cogent observed facts about the packages based on solving an extensive set of test problems. The drivers, test problems, and data used in the study would be considered as the software component of the submission.

Certifications
A certification is a report on a previously published algorithm but, unlike a Remark, does not involve code changes to the published software. It can be a careful study of performance characteristics, a verification of correctness, or a report on extensive testing.

This is the only form of an algorithm-type paper for which there might not be any accompanying software component.

Submission Formalities

In addition to the general submission guidelines () authors of Algorithm papers should also upload a separate file containing the software component of the submission. This file should be in the form of a, preferably compressed, archive file (for example, zip or tar.gz or tar). The individual files should be arranged in a suitable directory structure so as to allow the easy identification of the components of the submitted material (for example, algorithm source, supporting material, test code, etc.)

Please note that, except in very unusual circumstances, detailed descriptions of the calling sequences of individual routines will not be published in the journal and should be provided with the software submission as a machine readable user manual.

1. If a companion paper is to be submitted to ACM TOMS then both papers and the software should be uploaded as a single submission. If it is more appropriate for the companion paper to be submitted to another of the ACM Transactions, please contact the Algorithms Editor (mailto:t.r.hopkins@kent.ac.uk) prior to submission.
2. Processing of an algorithm will not proceed until it is verified that machine-readable copy of the algorithm is in a suitable form. If an algorithm requires the use of code subject to restricted use, the use of this code should be cleared first with the Algorithms Editor (mailto:t.r.hopkins@kent.ac.uk).
3. Evidence of portability must be included with an algorithm submission.

Dissemination Agreement

Copyright Agreement

References

[1] International Organization for Standardization. Information technology -- Programming languages -- Fortran - Part 1: Base language, ISO/IEC 1539-1:2004, International Organization for Standardization, Geneva, Switzerland, 2004.

[2] International Organization for Standardization. Programming languages -- C ISO/IEC 9899:1999 International Organization for Standardization, Geneva, Switzerland, 1999.

[3] International Organization for Standardization. Programming languages -- C++ ISO/IEC 14882:2003 International Organization for Standardization, Geneva, Switzerland, 2003.

[4] International Organization for Standardization. Information technology -- Programming languages -- Ada ISO/IEC 8652:1995 International Organization for Standardization, Geneva, Switzerland, 1995.

[5] Ford, B. Parameterization of the environment for transportable numerical software. *ACM Trans. Math. Softw.* 4, 2 (June 1978), 100-103.

Fred T. Krogh
Revised March 1979 by Webb Miller
Revised January 1982 by Richard Hanson
Revised October 1999 by Tim Hopkins
Revised August 2007 by Tim Hopkins

All ACM Journals | See Full Journal Index (https://dl.acm.org/pubs.cfm)