The Common Optimization INterface for Operations Research: Promoting open-source software in the operations research community

R. Lougee-Heimer

The Common Optimization INterface for Operations Research (COIN-OR, http://www.coin-or.org/*) is an initiative to promote open-source software for the operations research (OR) community. In OR practice and research, software is fundamental. The dependence of OR on software implies that the ways in which software is developed, managed, and distributed can have a significant impact on the field. Open source is a relatively new software development and distribution model which offers advantages over current practices. Its viability depends on the precise definition of open source, on the culture of a distributed developer community, and on a version-control system which makes distributed development possible. In this paper, we review open-source philosophy and culture, and present the goals and status of COIN-OR.*

Introduction: The disadvantages of current research-software practices

Operations research (OR) is the "discipline of applying quantitative techniques to make decisions" [1]. Among the variety of techniques employed in OR are optimization, stochastic methods, decision analysis, simulation, and econometric methods. The standard approach in OR is to create a mathematical model—an abstract representation of the situation—which attempts to capture the pertinent aspects of the complex system in which the real-world problem arises. The model is used to predict what will happen to the system under different circumstances. The model is analyzed, and the equations of which it consists are solved repeatedly using computer programs.

OR researchers strive to invent new solution methods which can solve bigger, more complex models in as short a time as possible. A new solution approach is codified as a theoretical, or abstract, algorithm. To validate the approach and compare its performance against existing methods, the algorithm is implemented in software, and computational studies are conducted. The implementations are usually prototypes written in a high-level language and intended for the author's use only. The results of computational studies are often provided as a complement to theorems, proofs, and algorithms in peer-reviewed publications. For instance, in *Operations Research*, the flagship journal of the Institute for Operations Research and the Management Sciences, roughly 75% of the articles published in 2001 contained computational results. Several journals are devoted to the intersection of OR and computer science—for instance, the *INFORMS Journal on Computing* (*http://joc.pubs.informs.org/*), *Computers & Operations Research* (*http://www.elsevier.nl/homepage/sae/orms/cor/*), and *Computational Optimization and Applications* (*http://www.wkap.nl/prod/j/0926-6003/*). But while the algorithmic theory is peer-reviewed and openly disseminated, the software is not.

The following are consequences of current research-software development and distribution practices:

- *Results are irreproducible.* In the physical sciences, experiments are described in sufficient detail to

0018-8646/03/$5.00 © 2003 IBM

enable other scientists to reproduce the reported results. Results from computational experiments are products of the complexity of the algorithm and the efficiency of the implementation. When implementation decisions are not disclosed, results are not reproducible. For example, had it been possible to make the implementation used by Karmarkar publicly available, his computational studies could have been quickly repeated, and the confusion surrounding his contributions would have been lessened [2, 3].

- *Comparisons are unfair.* Without access to existing implementations, researchers needing to compare computational results from their new approach with those from a published approach must reimplement the original algorithm. The end result can be an unfair comparison between "my implementation of your idea" and "my implementation of my idea."
- *Models and implementations are lost.* Writing a theoretical paper for external review forces authors to express their ideas to a higher standard, and serves to document those ideas in such a form that any person with sufficient background can comprehend the results years after they are established. Work that is not archived is lost (e.g., Fermat's proof of his last theorem). Because implementations and models are not written for public consumption and archived, they are prone to loss. Many computational studies in the literature are based on code expediently written for the author's use only. If code is not used and maintained as the author's interests invariably evolve, the utility of the program quickly diminishes, and it becomes unusable. (This observation is so universal within software circles that it has a name, "software rot," and a jocular theory as to how it arises, namely "bit decay.")
- *Evolution is stunted.* Without access to the source code, OR researchers are not able to learn from and build on the best software ideas in the field. The software genius of our time is not being captured. Instead, OR professionals are limited to their own ideas and personal resources. Evolution of the field is stifled when ideas are not peer-reviewed and made public.
- *Wheels are reinvented.* Without access to source code, researchers are forced to reinvent rather than reuse existing code. For researchers, the time and effort spent reinventing is especially wasteful, considering the often incremental nature of scientific progress. Not atypically, researchers read a published paper with computational results and invent a way to extend the state of the art using an idea which at its heart is a novel twist on an existing algorithm or a new extension to a related problem. Ideally, the researcher would like to make a modification to the existing code from a published study, but that code is usually not available. Instead, they must start from scratch. Because published papers present the distilled essence of abstract ideas while implementations must operate within the confines of a computer's finite number and instruction sets, the cost of reinventing is far from negligible. The process of redesigning, reimplementing, and retesting takes time and skill.
- *Knowledge transfer is limited.* Theoretical advances in one problem class are often transferred to other problem classes. Because implementation advances are obscured, the opportunity to transfer such improvements is lost.
- *Collaboration is inhibited by lack of standards.* The lack of established venues for disseminating software contributes to the absence of software standards. Even the current "standard" Mathematical Programming System (MPS) file format is interpreted differently among commercial optimization-problem solvers. Today, many research algorithms are implemented on top of a commercial solver. The commercial solver is used for its data input/output features and data structures, and as an embedded engine to solve specialized subproblems called for by the new algorithm. The commercial engine is invoked using the application programming interface (API) of the product. This practice binds the implementation to that specific product and impedes collaboration between colleagues who do not have access to the same product.

These consequences cause difficulty and delay for individuals developing or using OR-research software. While the community reaps the benefits of the open literature for theory, it passes up similar benefits by not openly disseminating the supporting software. Why?

A historical perspective of current software practices

It is instructive to note that in the early days of computing, source code was freely shared among groups using the same hardware. Software was not very portable; programs were usually closely tied to a particular vendor's hardware. The hardware market supported a profit margin large enough to make giving away software as an incentive to hardware sales a successful business model for computer manufacturers. This marketing practice extended to optimization software. For example, the source code of the IBM MPSX package (a leading mixed-integer-program solver of its day, written in assembly language) was made available to members of SHARE (an IBM user group [4]) through the IBM Program Library. The IBM Program Library contained programming systems, application programs from IBM development centers, software from the IBM Scientific Centers [e.g., 5, 6], and customer-contributed code.[1]

[1] William White, personal communication, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, November 12, 2001.

Precipitated by a massive antitrust complaint filed against IBM by the Justice Department in January 1969, the company reexamined its practices and decided to stop requiring customers to buy software, services, and hardware as one bundle in June of the same year [7]. This pricing change opened up software markets to independent companies. Software started generating a greater revenue stream in its own right, and the culture correspondingly became increasingly proprietary. Today, free-for-academic-use-only licenses for the IBM Optimization Solutions and Library (the successor to IBM MPSX) are given away, but the source code is not.

Simple inconvenience was a factor contributing to the lack of research-software dissemination. The only sensible way to distribute software is in an electronic format, but until the advent of the Web, there was no practical, universally accessible means for distributing it on demand. But now that the Web is available, the inconvenience of making software available for peer review is arguably no greater than that of making theory available for peer review. If the benefits justify disseminating research theory, why do the analogous benefits not justify disseminating the related research software (even though the review processes may differ)?

OR research and OR practice (mathematical methods and their practical applications) are two extremes on a continuum, not a dichotomy. Along this continuum, research software can be developed into a service or product offering for sale. For researchers in academia, subsequent profits can support graduate students or provide summer funding, and in some cases are sufficient to start up a business or to be bought out by one. The majority of the research code written today does not travel down the path to market, but the realization that this is an available path encourages a proprietary attitude toward research code.

Clearly, in the traditional software business models, freely disseminating software with a significant potential for profit from licensing fees is ill-advised, and one may wonder whether it would not be prudent to protect the underlying theory as well. For software not in this class, namely that which has great technical or scientific value and is not otherwise going to be marketed, the decision to keep it proprietary is not always rational; the benefits of disseminating code are underestimated, and the potential for profit is overestimated [8].

One way researchers seemingly profit from not disseminating code is by raising the barrier to entrance for would-be competitors mining the same research vein. "Giving away" code, by publishing it along with the theory, may seem like giving away a lead in the race to extend the state of the art. Rather than foregoing the benefits of publishing code, what is needed is a suitable definition of "giving it away" that makes disseminating software an optimal strategy for the author. One such definition is the open-source definition.

The open-source alternative

The definition of open source

The underlying philosophy of open source is to promote software reliability and quality by supporting independent peer review and rapid evolution of source code. This philosophy is pragmatically advanced by using copyright law in a nontraditional way. When one "buys software," one actually purchases a license from the copyright holder to use the software product. Today, the buyer usually receives a precompiled library or executable program, but not the source code. This allows the buyer to exercise the program, but little else. Without source code, the buyer cannot fix a bug (or even pay someone else to fix it), nor can he modify or improve the software. In contrast, open source gives users access to source code and legal rights to modify and redistribute the modifications so that the code evolves. In this section, we give a brief introduction to open source; for a thorough discussion of the open-source philosophy and an analysis of the business aspects of open source, see [9] and [8], respectively.

Technically speaking, the term *open source* refers to a category of software licenses defined and promoted by a nonprofit corporation called the Open Source Initiative (OSI). To protect the term from misuse, the OSI runs a certification program. Software with the "OSI Certified" label is distributed under a license satisfying the *Open Source Definition* (*http://www.opensource.org/*). Much broader than simply requiring access to source code, the Open Source Definition version 1.9 states the following criteria on nine fundamental issues.

1. Free redistribution.
   "The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license shall not require a royalty or other fee for such sale."
2. Access to source code.
   "The program must include source code, and must allow distribution in source code as well as compiled form. Where some form of a product is not distributed with source code, there must be a well-publicized means of obtaining the source code for no more than a reasonable reproduction cost—preferably, downloading via the Internet without charge. The source code must be the preferred form in which a programmer would modify the program. Deliberately obfuscated source code is

not allowed. Intermediate forms such as the output of a preprocessor or translator are not allowed."

3. Derived works.
   "The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software."
4. Integrity of the author's source code.
   "The license may restrict source-code from being distributed in modified form only if the license allows the distribution of "patch files" with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of software built from modified source code. The license may require derived works to carry a different name or version number from the original software."
5. No discrimination against persons or groups.
   "The license must not discriminate against any person or group of persons."
6. No discrimination against fields of endeavor.
   "The license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used for genetic research."
7. Distribution of license.
   "The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties."
8. The license must not be specific to a product.
   "The rights attached to the program must not depend on the program's being part of a particular software distribution. If the program is extracted from that distribution and used or distributed within the terms of the program's license, all parties to whom the program is redistributed should have the same rights as those that are granted in conjunction with the original software distribution."
9. The license must not restrict other software.
   "The license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other programs distributed on the same medium must be open-source software."

A common misconception is that open source is synonymous with public domain. Unlike software in the public domain, open-source software is clearly copyrighted. Open source differs from shareware or freeware, which are commonly only distributed as binaries. And unlike "free-for-academic-use-only" licenses, open-source licenses do not discriminate, so that diversity and participation are maximized.

As of this writing, there are more than 25 different OSI-certified licenses listed by the Open Source Initiative on their web page. Among the OSI-certified licenses are the GNU Public License (GPL), the Lesser GNU Public License (LGPL), the Berkeley System Distribution License (BSD), the Mozilla Public License (MPL), the IBM Public License (IPL), and the Common Public License (CPL, developed by IBM to extend benefits of the IPL to non-IBM software). All of the various OSI-certified licenses have certain attributes in common by virtue of satisfying the Open Source Definition, but open-source licenses can have significant differences as well. For example, all open-source licenses must *permit* unrestricted redistribution of source code, but some licenses *require* it. In the Apache and BSD licenses, distribution of source code is permitted, but not mandated for compiled or derivative works. (These are among the least restrictive of open-source licenses [10].) In contrast, the GPL requires it.

Perhaps the most significant difference, and certainly the most controversial, is whether a license is "viral"—that is, whether it requires that derived work, combination, or modification of the licensed software retain the same license. Licenses with this provision are called viral because of the infectious way in which they propagate: The only way to compile with software under such a license is to adopt the license. The GPL is viral; the MPL is not viral.

The development of open-source code

In a typical successful open-source software project (e.g., the Linux** operating system), a virtual community of volunteer developers spontaneously arises from among users. Users download the source code from the Web, and may use it as is or modify it for their own purposes. Users may find and fix bugs, extend functionality, and port to new platforms. They then typically (and, depending on the license, may be required to) contribute their modifications to be incorporated into the base code distribution. Because of the relatively large number of developers working simultaneously, the code evolves rapidly.

The virtual community shares a set of values and has a pyramid structure. The position of various people in the pyramid depends on the value of what each person contributes. At the bottom are the many casual users, in the middle are developers, and at the top is a small core team that controls the changes to the project's officially distributed code. The core team may make decisions by consensus or majority rule, and may be led by a "benevolent dictator" with ultimate authority. Open-source communities are sometimes described as "brutal meritocracies"; they run on ego and reputation. Developers are volunteers who take pride in their work. The software does not have owners, but rather maintainers. Bugs are made public, not hidden. Open exchange is valued, and information hoarding is abhorred.

Managing open-source projects

Open-source projects typically distribute at least two versions of the code: a production version and a development version. The production version is relatively stable, with only maintenance changes being incorporated. By contrast, the development version is volatile. In active projects, it changes daily if not hourly. The rapid code evolution is a consequence of the parallel development by the virtual community. Each volunteer developer is simultaneously editing the same source code. To manage this seemingly unwieldy process, a version-control system is used.

At the heart of a version-control system is a source-code repository which stores the base distribution of the source. Individual users make a copy of the repository code on their local machines for their own use. Changes to the base distribution code can be made by only those select people with write-access to the repository. Leading version-control systems cleverly record the history of source file changes by storing the differences. These systems enable concurrent revisioning, without locking out users, by requiring that commits are made sequentially to promote compatibility. To update the local copy with the changes that have occurred to the repository code since the last update, a user issues a command which synchronizes the two versions. The update attempts to apply "patches" to the user's local copy to bring it up to the current repository level. If the version-control system is unable to automatically resolve differences, a conflict message is issued. Conflicts must be manually resolved by the user. One widely used management system among open-source projects is CVS, the Concurrent Versions System [11]. Like many tools used in open-source development, it is itself an open-source project.

When visions differ: Forking

Core-team members are developers with write access to the source-code repository. As maintainers of the projects, it is their job to determine which of the contributed changes should be accepted and which should be rejected. The overwhelming majority of the time, these decisions are based on a shared vision using sound reasoning that is accepted by the contributor. But sometimes visions diverge. If a contributed change is not accepted, contributors are free to start their own open-source projects by taking the current repository, incorporating the rejected contribution, and distributing the divergent code. When the repository tree of revisions is split off in this manner, it is referred to as a "fork" in the project. A fork is a serious step, and while forks do happen (e.g., the XEmacs editor split off from the GNU Emacs editor over a desire for a graphical user interface version for the package), they are surprisingly rare, especially considering the diverse, decentralized nature of the developer pool.

OR and open source

It may seem counterintuitive at first that such an asynchronous, virtual community of volunteers could produce working code, let alone high-performance, high-quality, reliable, secure code. And yet it has. Much of the Internet is run on open source. (Roughly 60% of Web sites run on the open-source Apache HTTP server, vs. 25% for Microsoft's IIS [12].) The benefits of the open-source paradigm have been proven by many successful projects, and these benefits address adverse consequences typical in the development and distribution of OR research software. By opening source code for peer review and rapid evolution under an open-source license, computational results can be reproduced, fair comparisons of algorithm performance can be made, the best implementations can be archived and built on, code reinvention can be minimized, implementation innovation knowledge can be transferred, and collaboration and software standards can be fostered. Moreover, source code can be "dual-licensed," so not all profit potential from software sales would be foregone by the contributing author should an unforeseen business opportunity arise. For example, an author might offer software under both free and paid licenses with differing conditions on reuse. A license that *requires* distributed modifications to be opened ensures that anyone who contributes to the source (as well as the rest of the world) has access to future improvements by others.

While open source is an attractive alternative for the OR community, it is by no means a panacea. Open source originated in the field of computer science, and while there is a significant computing component to OR, the cultures and characteristics of the two communities are different. For example, open source gains advantage from a large community of volunteer developers. Operations research is a comparatively specialized area, and the number of developers is correspondingly smaller. Open-source projects that have succeeded have been fairly low on the software stack [e.g., operating systems (Linux), Web servers (Apache), scripting (Perl), Internet naming services (bind), Internet mail (sendmail)]. OR software belongs in the mid- to high-level application area of the software stack. To explore the viability of open source and its ability to accelerate progress in the field, COIN-OR was conceived.

Open-source software for the operations research community

The COIN-OR initiative

The Common Optimization INterface for Operations Research (COIN-OR) is a broad initiative to advance open source for the operations research community

[13–15]. The main thrust of COIN-OR is to build an open-source repository of OR software analogous to the open literature for OR theory, with the expectation of reaping analogous community benefits. A repository cannot be created and sustained without a community. To that end, COIN-OR serves to educate, to promote awareness, to provoke discussions, to encourage developers and users, and to otherwise build an open-source community for OR.

The public initiative was spearheaded by the IBM Research Division and kicked off with a conference presentation [16], the first organizational meeting, and the launching of the project's Web site (*http://www.coin-or.org/*) at the 17th International Symposium for Mathematical Programming in August 2000. IBM Research made a three-year commitment to support the online infrastructure of the initiative until the next triennial meeting of the Mathematical Programming Society. The software repository on the COIN-OR Web site was seeded with two state-of-the-art projects opened by IBM Research under the OSI-certified IBM Public License (IPL), and with two newly initiated projects. COIN-OR is intended for all aspects of OR; however, the four initial contributions featured tools for large-scale mixed-integer linear programming and combinatorial optimization. These tools, which continue to evolve today, are the following.

- The *Open Solver Interface* is an API, written in C++, that enables implementations to be "solver agnostic." Algorithms can be implemented once and then run using any solver having an open solver interface instantiation with no additional effort. The Open Solver Interface promotes collaboration by enabling researchers without access to the same solver to use and build upon the one common software base. Currently, interfaces to the mixed-integer-linear components of three commercial solvers (the ILOG CPLEX, the IBM OSL, and XPRESS-MP from Dash Optimization), and to one subgradient solver (the Volume Algorithm), are available in the repository [17]. Efforts to interface with open solvers are underway.
- The *Volume Algorithm* is an implementation of an extended subgradient method due to Barahona and Anbil [18] that produces approximate primal solutions as well as the usual dual solutions. More generally, the Volume Algorithm can be used as a C++ framework for Lagrangian relaxation. Implementations of the Volume Algorithm have been deployed in practice by IBM Research to solve large-scale scheduling problems arising in the airline industry [19–22].
- The *Cut Generation Library* is a collection of cutting-plane implementations, currently including a variety of knapsack-cover cuts, simple rounding cuts, and odd-hole cuts [36], and most recently, lift-and-project cuts [23]. The Cut Generation Library is written in C++ and integrated with the Open Solver Interface [24]. More cut generators are under development.
- The *Branch-Cut-Price (BCP) Framework* is a framework written in C++ for solving mixed-integer programs in parallel on a distributed network of workstations using LP-relaxation-based branch, cut, and price techniques. In the tradition of [25], the BCP framework liberates users from having to code the backbone of functionality that is common to all branch, cut, and price techniques so that they can more quickly develop and deploy their own problem-specific applications [26]. BCP has been used by IBM Research in successful customer engagements, including projects in the airline and steel industries [27, 28]. BCP is now fully integrated with the Open Solver Interface [29], and work is underway to integrate it with the Cut Generation Library [30].

Since the debut of COIN-OR, two more contributions have been made (and more are in the pipeline):

- *Derivative Free Optimization* is a solver for general nonlinear optimization problems in which the objective function is relatively expensive to compute, and the derivatives are not available and cannot be estimated efficiently. Problems with these characteristics arise, for instance, in engineering design, where the function evaluations are made via simulation. Although the solver can be used on larger problems, reasonable performance expectations make the approach attractive for problems with no more than 100 variables. The Derivative Free Optimization solver was developed by Conn, Scheinberg, and Toint, and is written in Fortran 77 [31–33].
- *Open Tabu Search* is a framework written in Java** for implementing tabu search metaheuristics. As a framework, Open Tabu Search provides users with a structured environment for implementing their own customized tabu-search algorithms and offers for reuse a backbone of capabilities commonly used in tabu search. The Open Tabu Search code was developed and contributed by Robert Harder under the OSI-certified Common Public License. Open Tabu Search was the first contribution to COIN-OR fully developed outside IBM. To the best of our knowledge, it is also the first open-source software project fully developed outside IBM to be distributed from any IBM-owned server. As a first-of-a-kind, Open Tabu Search was influential in provoking the creation of the Common Public License, a new license with the benefits of the IPL for non-IBM software [34, 35].

COIN-OR after year one

Progress was made in several directions during the first year of COIN-OR. The open-source code repository expanded as new projects were added, existing projects improved through community involvement, and research efforts were facilitated via reuse of the software components in COIN-OR.

The contribution of the original CPLEX Open Solver Interface is an example of how open source has successfully led to the evolution of existing projects during the first year of existence of COIN-OR. At the COIN-OR debut, an initial implementation of the Open Solver Interface was posted in an online repository at the project Web site. The COIN-OR team and project launch were based in the United States. Originally, the repository contained interfaces for the Dash Optimization XPRESS-MP and the IBM OSL solvers. A few weeks later, the core team was surprised and delighted by a communication from Tobias Pfender of Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB), Germany. Unbeknownst to anyone actively involved in the project up to that time, Pfender had implemented an open solver interface to the ILOG CPLEX solver and was offering the contribution to be incorporated into COIN-OR.

Contributions to COIN-OR, such as Pfender's, can be counted. Measuring the usage of the COIN-OR software is more difficult. Software in the COIN-OR repository is available on demand from the project Web site. Anyone with access to the Web can download the COIN-OR repository via the Web interface or via the daily *tarball* (UNIX** tape archive format). Consequently, there is no formal mechanism for knowing whether the software is being used, nor for tracking precisely where and how. Beyond gross Internet traffic statistics, code use can be inferred from exchanges on the discussion mailing lists and from personal contacts (which tend to arise primarily when users encounter bugs or other technical issues). Currently, there are approximately 140 total subscriptions to the coin-announcement and coin-discussion mailing lists.

One hard measure of software usage in independent research efforts is the number of citations in the refereed literature. COIN-OR has been cited in research papers submitted for publication by IBM and non-IBM authors. We briefly mention two such works to illustrate the type of research endeavors in which COIN-OR has successfully been used, and which exemplify the software reuse.

Traditionally, branch-and-cut methods [36] use the dual simplex algorithm to optimally solve the linear programming relaxation arising at each node of the search tree. IBM researchers Barahona and Ladanyi showed that certain classes of combinatorial optimization problems can be solved much faster if the approximate solutions produced by the Volume Algorithm are used instead of the optimal solutions produced by the simplex algorithm. They tested their ideas on two classic NP-hard combinatorial optimization problems (where NP means *nondeterministic polynomial*): the *Steiner tree problem* and the *max-cut problem*. Given a set of fixed points in the plane, and the ability to add new points (Steiner points), the Steiner tree problem is to determine a tree of minimal Euclidean length which spans the given set. The max-cut problem is to divide the vertices of a graph into two parts so that the number of edges between them is as large as possible. Both of these problems arise in numerous direct applications, such as very large-scale integration (VLSI) circuit design. Using an implementation based on the BCP framework and the Volume Algorithm, Barahona and Ladányi were able to calculate the optimal solution to several previously unsolved instances of both the Steiner tree and max-cut problems [29]. Their implementation for the max-cut problem is available in COIN-OR for others to reuse, and their implementation for the Steiner tree problem is forthcoming.

Guglielmo Lulli and Suvrajeet Sen of the Department of Systems and Industrial Engineering at the University of Arizona developed a branch-and-price methodology for solving specially structured multi-stage stochastic integer programming problems. To test their approach computationally, they implemented their algorithm using the BCP framework and considered a stochastic version of the well-known batch-sizing problem.[2]

During the first year, a significant effort was directed toward publicizing and educating. At the November 2000 meeting of the Institute for Operations Research and the Management Sciences (INFORMS), an invited presentation on open source [37] was sponsored by the INFORMS Computing Society. Two invited technical talks were presented, and the first COIN-OR "installfests" [38, 39] and user meetings were held. Seminars on COIN-OR were given at the University of Arizona, Cornell University, two IBM Research Centers, the Optimization Symposium of the INFORMS Austin Texas Chapter, and the Third International Workshop on Integration of AI (Artificial Intelligence) and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'01) in Wye, England. At the 2001 INFORMS International Meeting in Hawaii, three technical talks on COIN-OR related projects were given, and the initiative was mentioned by name in the Omega Rho Distinguished Lecture by William Pulleyblank. In fact, COIN-OR has been mentioned in more than half a dozen university and conference presentations, including presentations on the state of the art at the first INFORMS conference for practitioners in La Jolla, California. Several articles on

[2] Guglielmo Lulli and Suvrajeet Sen, "A Branch-and-Price Algorithm for Multi-Stage Stochastic Integer Programming with Application to Stochastic Batch-Sizing Problem," submitted for publication.

COIN-OR have been submitted; a conference paper for CP-AI-OR'01 [13], an *ORMS Today* article [14], and an INFORMS Computing Society Newsletter article have appeared, and a book chapter is forthcoming [15].

As a result of the interest surrounding COIN-OR, conference organizers of the 2001 INFORMS meeting in Miami, Florida, invited the COIN-OR team to organize the first invited cluster on open source. Five technical presentations from nine authors at six different institutions, a workshop on open-source tools for branch, cut, and price, and a panel discussion to explore new directions for the MPS file format involving ILOG, Dash Optimization, IBM, Maximal Software, and AMPL were held. We expect that COIN-OR will play a significant role in creating and providing tools for this new standard. In addition, three talks involving open-source software were also contributed in the traditional invited cluster on discrete optimization [40–42].

Currently groups at the University of Arizona, Rutgers University in conjunction with Sandia National Laboratory, and Simon Fraser University are actively involved with the COIN-OR initiative, as well as core-team members at Clemson University, Lehigh University, the Pentagon, and Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB). We hope that their innovative computational work will be opened to the community, either on COIN-OR or elsewhere.

Conclusion

In this paper we have described the motivation, goals, and status of COIN-OR. It is our hope that the operations research community will find COIN-OR to be a key asset in promoting future development. COIN-OR invites participation and contributions at all levels, whether from users, developers, contributors, or thought-provokers. In addition to expanding existing projects, new projects (e.g., modeling languages, visualization, spreadsheet plug-ins, solvers) and people to lead them are welcome.

To maximize software reuse, it would be ideal if the projects under COIN-OR used licenses that allowed source code from one project to be used in another. Because of the lack of such licenses, we encourage all contributors to use the same license, and suggest the CPL. Contributed projects and licensing decisions are considered on a case-by-case basis. A list of OSI-certified licenses is available through the Open Source Initiative Web site at *http://www.opensource.org/*.

The code in the COIN-OR repository is available for practitioners, academics, and students for use in business, research, and teaching. We encourage anyone who is interested in open source for OR to join the mailing lists, to visit the COIN-OR Web site at *http://www.coin-or.org/*, and to help make the vision of open source in OR a reality.

Acknowledgments

The author thanks her fellow COIN-OR core-team members, J. P. Fasano, John Forrest, Robert Harder (U.S. Air Force), Laszlo Ladanyi, Tobias Pfender (ZIB, Germany), Ted Ralphs (Lehigh), Matthew Saltzman (Clemson), and Katya Scheinberg; colleagues in academia and industry, Ranga Anbil, Vernon Austel, Francisco Barahona, Andy Conn, Bob Daniel (Dash), Brenda Dietrich, Marta Eso, Jonathan Eckstein (Rutgers University), Lou Hafer (Simon Fraser University, Canada), Bjarni Kristjansson (Maximal Software), Bertrand Le Cun (University of Versailles, France), Guglielmo Lulli (University of Arizona), Irv Lustig (ILOG), Mikhail Nediak (Rutgers), Greta Pangborn (Cornell University), Suvrajeet Sen (University of Arizona), and Stephen Tse (Simon Fraser University); as well as the many other COIN-OR users and mailing-list participants for their contributions of code, documentation, licenses, enthusiasm, and ideas which collectively are COIN-OR. Special thanks are due Brenda Dietrich, Marta Eso, Laszlo Ladanyi, Ted Ralphs, Matthew Saltzman, and Eric Wolman (George Mason University) for their helpful suggestions on an earlier draft of this paper.

References

1. *Occupational Outlook Handbook 2000–2001 Edition*, U.S. Department of Labor, Washington, DC; see *http://stats.bls.gov/oco/ocos044.htm*.
2. Murray Gill, M. Saunders, J. A. Tomlin, and M. Wright, "On Projected Newton Barrier Methods for Linear Programming and an Equivalence to Karmarkar's Projective Method," *Math. Program.* **36,** 183–209 (1986).
3. J. J. H. Forrest and J. A. Tomlin, "Implementing Interior Point Linear Programming Methods in the Optimization Subroutine Library," *IBM Syst. J.* **31,** No. 1, 26–38 (1987).
4. SHARE; see *http://www.share.org/*.
5. L. Papayanopoulos, "Linear Programming System for the OS/360," IBM New York Scientific Center, Doc. No. 69NYSC1, New York, June 1969.
6. L. Bodin, M. Grigoriadis, K. Harrow, L. Papayanopoulos, K. Spielberg, S. Torok, and W. White, "The NYLPS/MIPIS System for Mixed Integer Programming," IBM Philadelphia Scientific Center, Doc. No. 71PSC1, February 1971.
7. Thomas J. Watson, Jr., *Father, Son & Co.: My Life at IBM and Beyond*, Bantam Books, New York, 1990.
8. Eric S. Raymond, *The Magic Cauldron*, June 1999; see *http://www.tuxedo.org/~esr/writings/magic-cauldron/*.
9. Eric S. Raymond, *The Cathedral and the Bazaar*, O'Reilly, 1999; see *http://www.tuxedo.org/~esr/writings/cathedral-bazaar/hacker-history/*.
10. Rex Brooks; see *http://www.vrml.org/TaskGroups/vrml-ipr/open_source_overview.html*.
11. Concurrent Versions System; see *http://www.cvshome.org/*.
12. Netcraft; see *http://www.netcraft.com/*, July 2001.

13. Robin Lougee-Heimer, "The COIN-OR Initiative: Open Source Software for Optimization," *Proceedings of the Third International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems* (*CP-AI-OR'01*), Wye, England, 2001, pp. 307–319.
14. Robin Lougee-Heimer, Francisco Barahona, Brenda Dietrich, J. P. Fasano, John Forrest, Robert Harder, Laszlo Ladanyi, Tobias Pfender, Theodore Ralphs, Matthew Saltzman, and Katya Scheinberg, "The COIN-OR Initiative: Open Source Accelerates Operations Research Progress," *ORMS Today* **28,** No. 4, 20–22 (October 2001).
15. Matthew J. Saltzman, "COIN-OR: An Open-Source Library for Optimization," *Programming Languages and Systems in Computational Economics and Finance*, Soren S. Nielsen, Ed., Kluwer Academic Publishers, Boston, in press.
16. William Pulleyblank, Brenda Dietrich, John Forrest, and Robin Lougee-Heimer, "Open Source for Optimization Software," presented at the International Symposium for Mathematical Programming (ISMP), Atlanta, August 2000; abstract, p. 33.
17. Matthew J. Saltzman, "The COIN-OR Open Solver Interface: Toward Solver Independent MIP Algorithms," presented at the INFORMS Annual Meeting, San Antonio, Fall 2000; abstract, p. 106.
18. Francisco Barahona and Ranga Anbil, "The Volume Algorithm: Producing Primal Solutions with a Subgradient Method," *Research Report RC-21103(94395)*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1998.
19. Francisco Barahona and Ranga Anbil, "On Some Difficult Linear Programs Coming from Set Partitioning," *Research Report RC-21410*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1999.
20. Francisco Barahona and Ranga Anbil, "Solving Large Scale Uncapacitated Facility Location Problems," *Research Report RC-21515*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1999.
21. Francisco Barahona and Fabain Chudak, "Near-Optimal Solutions to Large Scale Facility Location Problems," *Research Report RC-21606*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1999.
22. Laura Bahiense, Francisco Barahona, and Oscar Porto, "Solving Steiner Tree Problems in Graphs with Lagrangian Relaxation," *Research Report RC-21847*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 2000.
23. Egon Balas, Sebastian Ceria, and Gerard Cornuejols, "A Lift-and-Project Cutting Plane Algorithm for Mixed 0–1 Programs," *Math. Program.* **58,** 295–324 (1993).
24. Robin Lougee-Heimer, "The COIN-OR Cut Generation Library: Toward Greater Code Reuse," presented at the INFORMS Annual Meeting, San Antonio, Fall 2000; abstract, p. 106.
25. M. Eso, L. Ladanyi, T. K. Ralphs, and L. E. Trotter, Jr., "Fully Parallel Generic Branch-and-Cut Framework," presented at the 8th SIAM Conference on Parallel Processing for Scientific Computing, March 1997.
26. M. Eso, S. Ghosh, L. Ladanyi, and J. Kalagnanam, "Bid Evaluation in Procurement Auctions with Piece-wise Linear Supply Curves," *Research Report RC-22219*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 2001.
27. R. Anbil, F. Barahona, L. Ladanyi, R. Rushmeier, and J. Snowdon, "Airline Optimization," *ORMS Today* **27,** No. 6, 26–29 (December 1999).
28. L. Ladanyi, J. J. Forrest, and J. Kalagnanam, "Column Generation Approach to the Multiple Knapsack Problem with Color Constraints," *Research Report RC-22013*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY 2001.
29. F. Barahona and L. Ladanyi, "Branch and Cut Based on the Volume Algorithm: Steiner Trees in Graphs and Max-Cut," *Research Report RC-22221*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 2001.
30. Theodore K. Ralphs and Laszlo Ladanyi, *COIN/BCP User's Manual*, 2001; see *http://www.coin-or.org/*.
31. Andrew R. Conn, Katya Scheinberg, and Ph. L. Toint, "On the Convergence of Derivative-Free Methods for Unconstrained Optimization," *Approximation Theory and Optimization: Tributes to M. H. D. Powell*, A. Iserles and M. Buhmann, Eds., Cambridge University Press, Cambridge, England, 1997, pp. 83–108.
32. Andrew R. Conn, Katya Scheinberg, and Ph. L. Toint, "Recent Progress in Unconstrained Nonlinear Optimization Without Derivative," *Math. Program.* **79,** 397–414 (1997).
33. Andrew R. Conn, Katya Scheinberg, and Ph. L. Toint, "A Derivative Free Optimization Algorithm," presented at the 7th AiAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, St. Louis, 1998.
34. Robert Harder, OpenTS—Java tabu search; see *http://www.coin-or.org/OpenTS/*, 2001.
35. Robert Harder, "OpenTS: An Open Source Java Tabu Search Framework," presented at the INFORMS Annual Meeting, Miami, 2001.
36. George L. Nemhauser and Lawrence A. Wolsey, *Integer and Combinatorial Optimization*, John Wiley & Sons, New York, 1988.
37. Donald K. Rosenberg, "A Primer on Open-Source Software," presented at the INFORMS Annual Meeting, San Antonio, Fall 2000; abstract, p. 62; see *http://www.stromian.com/*.
38. Laszlo Ladanyi, J. P. Fasano, Robin Lougee-Heimer, and Matthew J. Saltzman, "COIN Installfest for Linux: A Hands-On Workshop Using Open Source Software for OR," presented at the INFORMS Annual Meeting, San Antonio, Fall 2000; abstract, p. 22.
39. Laszlo Ladanyi, J. P. Fasano, Robin Lougee-Heimer, and Matthew J. Saltzman, "COIN Installfest for Window: A Hands-On Workshop Using Open Source Software for OR," presented at the INFORMS Annual Meeting, San Antonio, Fall 2000; abstract, p. 33.
40. Laszlo Ladanyi and Ted K. Ralphs, "Branch, Cut & Price: The Next Generation," presented at the INFORMS Annual Meeting, Miami, 2001; abstract, p. 52.
41. Suvrajeet Sen and Guglielmo Lulli, "A Branch & Price Algorithm for Stochastic Lot-Sizing," presented at the INFORMS Annual Meeting, Miami, 2001; abstract, p. 52.
42. Ted K. Ralphs and Joseph C. Hartman, "Branch & Cut for Capacitated Network Routing," presented at the INFORMS Annual Meeting, Miami, 2001; abstract, p. 52.

Received November 21, 2001; accepted for publication July 25, 2002

Robin Lougee-Heimer *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (robinlh@us.ibm.com).* Dr. Lougee-Heimer received her Ph.D. degree in mathematical sciences from Clemson University, joining IBM in 1994 as a Research Staff Member in the Mathematical Sciences Department. As an industrial researcher, Dr. Lougee-Heimer is charged with both conducting basic research and supporting IBM's businesses. Her main research interest is in developing efficient solutions to large-scale discrete optimization problems arising in industry. Dr. Lougee-Heimer is a leader in the initiative to promote open-source software for the operations research community. As a member of the Common Optimization Interface for Operations Research (COIN-OR) core team, she is a principal infrastructure manager and is leading the development of an open cutting plane library.