Listening. Learning. Leading.®

Research Report
ETS RR–15-05

Comparison of Integer Programming (IP) Solvers for Automated Test Assembly (ATA)

John R. Donoghue

June 2015

ETS Research Report Series

Since its 1947 founding, ETS has conducted and disseminated scientific research to support its products and services, and to advance the measurement and education fields. In keeping with these goals, ETS is committed to making its research freely available to the professional community and to the general public. Published accounts of ETS research, including papers in the ETS Research Report series, undergo a formal peer-review process by ETS staff to ensure that they meet established scientific and professional standards. All such ETS-conducted peer reviews are in addition to any reviews that outside organizations may provide as part of their own publication processes. Peer review notwithstanding, the positions expressed in the ETS Research Report series and other published accounts of ETS research are those of the authors and not necessarily those of the Officers and Trustees of Educational Testing Service.

The Daniel Eignor Editorship is named in honor of Dr. Daniel R. Eignor, who from 2001 until 2011 served the Research and Development division as Editor for the ETS Research Report series. The Eignor Editorship has been created to recognize the pivotal leadership role that Dr. Eignor played in the research publication process at ETS.

ETS Research Report Series ISSN 2330-8516

RESEARCH REPORT

Comparison of Integer Programming (IP) Solvers for Automated Test Assembly (ATA)

John R. Donoghue

Educational Testing Service, Princeton, NJ

At the heart of van der Linden's approach to automated test assembly (ATA) is a linear programming/integer programming (LP/IP) problem. A variety of IP solvers are available, ranging in cost from free to hundreds of thousands of dollars. In this paper, I compare several approaches to solving the underlying IP problem. These approaches range from traditional computer programming, through LP/IP-specific modeling languages, to plug-ins for common software such as Excel. The features of several of the major IP solvers are briefly reviewed, describing which of the features are more or less useful in the context of ATA. The appendices include a list of resources that I have found particularly useful.

Keywords ATA; integer programming; linear programming; LP/IP, test assembly

doi:10.1002/ets2.12051

When you have decided to use automated test assembly (ATA) following the procedures described by van der Linden (2005), you need two things: (a) a test assembly problem and (b) software to solve that problem. Finding a suitable problem is usually not a challenge. Figuring how to solve that problem can be a challenge. This paper reviews some of the alternatives.

No Silver Bullet

In his foundational paper, *No Silver Bullet*, Fred Brooks, (1995) described the two kinds of complexity encountered in software development: essential complexity and accidental complexity. Essential complexity is inherent in the problem one is trying to solve, whereas accidental complexity arises from the mismatch of the tools and the problem. Both of these ideas apply to ATA. Selecting three parallel test versions of five vertically linked forms, each with an embedded anchor, is an example of essential complexity of ATA. On the other hand, choosing a solver, converting the necessary data into a form that the solver will process, and figuring out what the output means are examples of accidental complexity in ATA.

Choosing software to solve an ATA problem can easily become overwhelming because of the variety of choices. In this paper, I provide a tour of the options, explain some of the terms and features, and suggest some factors to consider. However, if you are hoping to come away with a single, best answer after reading this paper, I'm afraid you will be disappointed. Brooks was correct: There *still* is no silver bullet.

One of the defining features of the linear programming/integer programming (LP/IP) world is that the field evolved around the problems attendant to running businesses (e.g., how best to cut bolts of cloth for clothing to minimize waste or how to schedule flights to optimize airplane use). Even the name of the field, operations research (OR), reflects the background of operating a large business.

This background has two key implications. First, because these problems affect the operations of large corporations, the money involved is substantial. For example, reducing the cost of operating a major airline through more efficient scheduling has large benefits; even an improvement of one tenth of 1% is still a substantial sum. On the up side, this money draws substantial resources to working on LP/IP problems, and the field has benefitted substantially as a result. On the down side, the prices of the commercial solvers can be shocking to those of us with more academic backgrounds. A single server license for one of the major commercial solvers (to remain nameless) costs one-third to one-half million dollars, plus an additional 20% in annual maintenance fees.

Corresponding author: J. R. Donoghue, E-mail: jdonoghue@ets.org

Another feature is that the industrial aspect sets the scale for the definition of "big." Problems with tens of millions of constraints can and do occur frequently. Thus, some of the features needed for big commercial problems may be overkill in the context of a fairly simple test assembly problem. On the other hand, ATA problems can and do become big and complicated. Finding the right match of solver power/cost to the problems at hand requires some experience. My advice is to start small then progress to more sophisticated/expensive alternatives once you have a good feel for the kinds of ATA problems you encounter.

One of the most surprising discoveries during my study was the relative lack of useful benchmark data to compare the performance of solvers. The best source I found is a decision tree created by Hans Mittelmann (http://plato.asu.edu/bench.html). A few data sets have been developed, but the behavior of the solvers can be highly dependent on the nature of the problem. When I first heard this assertion, I was somewhat dubious, but even the high-pressure sales staff of the commercial solvers told me the same thing: Behavior depends on the nature of your problem. There are some obvious, overall trends (the high-end commercial solvers tend to perform well regardless of the problem). Still, your mileage may vary.

In comparing the features of available options, consider a few important dimensions:

1. **How many problems do you have to solve?** A solver that takes a full 24 hours to yield a solution may be perfectly satisfactory if you have a few operational test assembly problems to solve each year. On the other hand, the same solver would clearly not be acceptable if you were doing a simulation study or doing CAT (computerized adaptive testing) based on (van der Linden, 2000) big-shadow-test method.
2. **How big is the problem?** For small problems (fewer than 300 – 500 items, fewer than 300 – 500 constraints), a variety of options exist that are not available for larger problems.
3. **How comfortable are you with programming? How proficient are you?** Some solutions are only available as software libraries. This availability gives a good programmer a substantial degree of control and flexibility. It also renders these options useless to the nonprogrammer.

The Layers of Options

I find it useful to picture the structure of solvers like geological strata (see Figure 1). My data and I are in the weeds on the surface. Below that we have the test assembly problem and then its algebraic formulation. The literature on ATA deals with these issues. In this paper, I focus on the layers below the algebraic formulation.

Down on the bottom layer is the solver engine. The next layer above is the application programming interface (API). Above the API are higher level language bindings. Above that are LP/IP modeling languages. The next layer consists of OR systems. At the top of this technology stack are interfaces: command files and graphical user interfaces (GUIs), and plug-ins. Note that these categories are not mutually exclusive. Many of the solvers (commercial and free) have capabilities at several of these levels. Nonetheless, the levels are useful for describing the features of the different ways to solve the ATA problem.

The Solver Engine

At the lowest level we have the solver engine. The solver engine is written in a low-level language such as C/C++ or Fortran. The use of these low-level languages is necessary for the high performance required by the solver engine. The chief challenge of working at the lowest level is the need to connect to the solver and to specify the ATA problem in a way that the solver engine can process it. The main term to be aware of at the level of the solver engine is application programming interface (API). The API is the collection of functions/subroutines that are used to control the solver. There might be one function to indicate the number of contrasts, another to indicate the number of items, another to send the data to the solver, and a fourth to retrieve the solution to the problem. When you see discussions of API, you know they are describing low-level interaction directly with the solver engine.[1]

Free examples of solver engine-based alternatives are COIN-OR and GPLK (discussed below), while commercial offerings include the Numerical Algorithms Group (NAG, http://www.nag.com) library in the C language and Visual Numerics's IMSL (http://www.vni.com/products/imsl). IMSL is mainly known as a Fortran library, but it has recently added support for Java, .NET, and Python. Thus, IMSL fits in this category as well as the next (higher level languages). One important distinguishing feature of NAG and IMSL is that they are general purpose libraries, and the solver is only

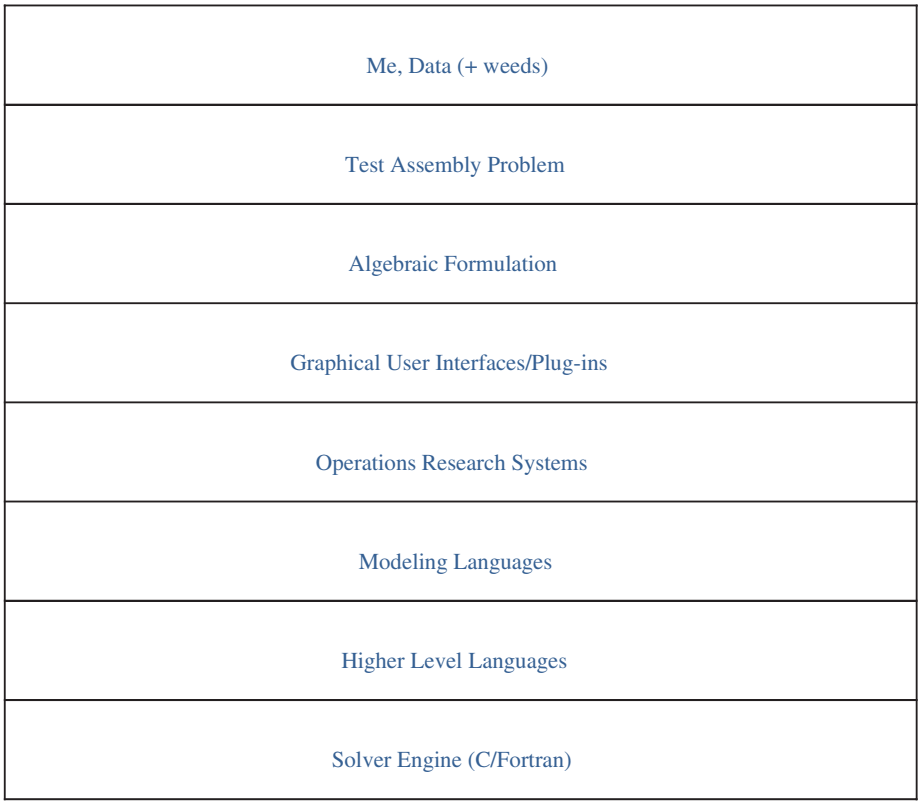

Figure 1 The strata of automated test assembly (ATA) from the user and the data at the top to the solver engine at the bottom.

one of many routines. Because IP/LP is only one of several types of algorithms that are supported, these packages can lag behind focused IP/LP products. Most of the alternatives we will consider focus solely on LP/IP.

Advantages

- **Control**. Working directly with the solver engine gives maximum control in tuning the settings to get the maximum performance. It also provides maximum flexibility in formulating the problem. If we have a complicated test design, it may be useful/necessary to be able to work directly with the solver in its native language rather than being forced to go through intermediate layers.
- **Availability**. Most solvers expose an API for working directly with the solver engine. Some solvers are only available this way. For example, COIN-OR is an initiative to connect academic findings in OR with commercial applications of the technologies. The solvers (including IBM's original solver) are available for free download and can be used for either academic or commercial purposes. However, the vast majority of COIN-OR routines are only available as solver engines, either as source code or with a compiled version of the solver engine alone. In either case, the user will have to write code to communicate the ATA problem to the solver.

Disadvantages

- **Need to program**. The chief disadvantage of working directly with the solver engine is that you need to be extremely comfortable programming, generally in the solver's native language. Many practitioners of ATA would not have the skills (or interest) to work at this level.
- **Emersion in detail**. Working at the API level often means that you need to invest substantial time[2] in understanding the detail of the solver. This added time commitment is a classic example of (Brooks, 1995) accidental complexity, as it is irrelevant to solving our real ATA problem.

Working directly with the solver engine may be worthwhile if you have the programming skills. There are also situations (e.g., doing a large-scale simulation) in which the need for speed and control make working directly with the solver engine attractive/necessary. However, for most applications of ATA, and most practitioners, the solver engine is too low a level in the strata to effectively solve the ATA problem.

Higher Level Languages

The next layer up from the solver engine is using a higher level language, such as Java, Python, or Ruby. A variation is using a statistical modeling system, such as R or MATLAB. The actual solver engine code still exists, but you can access it from a higher level language. Usually, you can still access all of the functionality of the solver, but the access functions as if the solver engine was written in the higher level language. The key term to look for in the descriptions of higher level languages is *bindings*. In this context, bindings mean a wrapper for the solver engine. The description of the solver typically will note that there are Java or Python (or whatever your language of choice) bindings available.

A concern about using higher level language is the effect it will have on performance; for example, will using the higher level language cause your problem to take forever to run. While this is a legitimate concern, in practice it will rarely be an issue. The higher level language will translate the problem into the native language of the solver engine. The actual solving process will take place in the lower level language, just as if you were using the solver engine directly. Because the vast majority of the solving time is spent running the solver engine, the overhead of translating the problem for the engine, and then translating the result back, is usually negligible. However, in the context of a large simulation, concerns over performance may be more important.

Advantages

- **Accessibility**. The chief advantage of using a higher level language is that these languages are often easier to learn. Most of the scripting languages (e.g., Python, Ruby, and Perl) have more advanced data structures than do the lower level languages (such as C and Fortran). Higher level languages also have better facilities for reading data from external files, and most have fairly simple GUI-creation facilities.
- **Reduction in complexity**. Often, the higher level languages reduce the complexity of interacting with the solver engine. The user is often able to operate at a level closer to the ATA conceptual entities, rather than in terms of the solver's native language.

Disadvantages

- **Still need to program**. Programming in a higher level computer language is still programming. While higher level languages are easier to learn and use, this kind of programming is still tangential to solving the ATA problem.
- **Limited to available bindings**. If you want to use a higher level language, someone has to have written the binding code. If there is no binding for your language of choice, you probably will have to find another solver. Writing the binding code is usually a bit complicated; it is done in a mixture of both the higher level language and the solver engine's language. If a user has the ability to write this code, he or she certainly has the ability to program a problem directly. Rather than pursue this route, most ATA practitioners will do better to find another solver.

Linear Programming/Integer Programming (LP/IP) Modeling Languages

A variety of specialized modeling languages are structured specifically to simplify the specification of LP/IP problems. Like the higher level programming languages, these modeling languages then translate the problem into a form that the solver engine can understand. The modeling language will then get the results from the engine and present it in a form that is much easier to understand. Essentially, these languages provide a specialized computer language for writing LP/IP problems. The key term here is *link*; the language will be described as providing links to specific solvers. This description means that ATA models created in that modeling language can then use the solver in question.

Most modeling languages are similar in complexity to statistical packages such as SAS. It is fairly simple to do straightforward analysis. Also like statistical packages, modeling languages offer the ability to do fairly complicated analyses; however, this ability comes with the price of engaging with the modeling language more deeply.

Modeling languages come in two main types. Multisolver languages allow the user to write in the modeling language and then choose which solver engine will be used. The modeling language statement of the problem does not have to alter if the user decides to change solvers. The best example of a multisolver language is AMPL (http://www.ampl.com). The AMPL logo (a black leopard on a blue background) is distinctive, and the book (Fourer, Gay, & Kernighan, 2002) associated with the software is a very readable introduction to the software and some of the general issues in the LP/IP field.

Solver-specific modeling languages are typically developed by the manufacturer of the solver. These languages may be bundled with the solver purchase or come with an added cost (however, this is usually a fraction of the cost of the solver itself). The CPlex solver has the OPL language, and the FICO solver has the Mosel language. Solver-specific languages can often take special advantage of the features of that solver. One drawback of these languages is the problem of vendor lock-in. If the user decides to switch solvers, he or she may have to completely rewrite the models for the new solver's language. This additional work often serves as a powerful disincentive to changing solver vendors.

Advantages

- **Ease of use**. It is much easier to go from the algebraic formulations in the literature (e.g., van der Linden's, 2005 book) to the problem the solver can deal with. The level of translation from the algebraic ATA problem to something that can be processed by the solver engine is usually much less effort than that needed to program in a computer language. For example, modeling languages provide good support for a variety of constraints in a fairly natural way.
- **Better documentation**. Because they are targeted to practitioners, modeling languages usually have better documentation than programming language alternatives. In addition to being easier to understand and use, the examples in modeling language documentation are presented in the language of LP/IP. Thus, users are more likely to be able to find an example that can be adapted to their problems. When dealing with the documentation for solver engine programming language APIs, this possibility is almost nonexistent.
- **Increase in functionality**. The AMPL language provides the ability to access data stored in relational databases and limited connectivity to Microsoft Excel. This kind of database access also exists in the commercial modeling languages.

Disadvantages

- **Cost**. Most modeling languages are not free. Because they take time and effort to develop, modeling languages are usually geared more toward business users. One notable exception is GNU's MathProg (described below under GLPK), which is a free implementation of AMPL. However, the full AMPL language is not implemented, making purchase of AMPL desirable and even necessary in many cases.
- **Highly specialized language**. The strength of modeling languages is also their weakness. Time spent learning SAS can be translated into other kinds of statistical analyses. In contrast, the user's investment in learning the modeling language is likely to be of benefit only for solving LP/IP problems.
- **Needs to fit the solver**. Much like higher level languages, not all modeling languages support all solvers. Before investing in a modeling language (even a free one), make sure it supports the solver that you want to use.
- **Cost of solver may not be included**. Even if a modeling language supports a specific solver, the cost of the language may not include a license for the actual solver. Commercial solvers are separate programs sold by different companies. In many cases, it is possible to buy a bundled version (modeling language plus solver) at a discount compared to purchasing the two products separately. But in any case, it is important to understand what is and is not included in the modeling language. Does the purchase include the solver, or will you have to pay an additional fee?

Operations Research (OR) Systems

The OR systems present a variety of solvers and capabilities under a single umbrella. Like multisolver modeling languages, OR systems support a variety of solvers; indeed, the division between the two is somewhat arbitrary. However, the OR systems are usually more extensive and can provide additional analysis capabilities, such as product-specific modeling language, tools to examine the solution, and other features. Like modeling languages, the ability to support a specific solver will often be termed a *link* to that solver. However, OR systems may simply say that they support a particular solver.

Frequently, OR systems advertise ease of developing GUI-based front ends for the solver. In the context of a business with separate software development and test-development departments, this capability can provide a balance between the cost of developing a user-friendly system from scratch and picking an off-the-shelf solution.

All three major OR systems — GAMS (http://www.gams.com), AIMMS (http://www.aimms.com), and Maximal (http://www.maximal-usa.com) — provide links to a variety of solvers plus additional decision support tools. Of the three, GAMS is the least polished, having a more academic feel. AIMMS and Maximal are much more polished, both in their advertising and in their interfaces. AIMMS and Maximal are commercial products aimed at commercial audiences. The licensing of GAMS is somewhat more complicated but also a bit more academic-friendly. Consult the respective Web sites to investigate the details of licensing and to get a feel for the differences.

Advantages

- **Documentation**. OR systems generally provide better documentation than solver-specific solutions. The documentation typically demonstrates a variety of example problems, and in some of the documentation, you can find surprisingly good descriptions of LP/IP modeling issues.
- **Unified interface to a variety of solvers**. Most OR systems provide access to a large number of solver engines. Indeed, the number of solvers supported is a feature that these systems compete on. However, like modeling languages, it is crucial to find out whether an additional license is required to use the solver of your choice. In many cases, using a commercial solver engine will require additional expense.

Disadvantages

- **More expensive**. OR systems tend to be more expensive than modeling languages. The facilities are usually more extensive, but it is important to decide if those additional facilities are actually useful. For many applications of ATA, an OR system may be overkill.
- **Complication**. The variety of features can make OR systems hard to master. Again, the ATA practitioner should think about balancing his or her investment of time in learning the system against the expected payoff. This balance is likely to be very different for someone who needs to assemble a few test forms for operational use compared to someone doing active, ongoing research in the field of ATA.

Modules and Plug-Ins

Many general use pieces of software have LP/IP solvers built into them. In the world of mathematical/statistical software, most products contain some capability to solve these problems, and so could be used for ATA. The R language was mentioned earlier. SAS, MATLAB, Mathematica, and other products also have LP/IP solver capabilities.

The most common, widely supported example here is Microsoft Excel. Recent versions of Excel come with a solver built in. For small to medium ATA problems, this capability may well be sufficient. A lesser known alternative is Microsoft's Solver Foundation (http://www.solverfoundation.com). The basic concept is that products like Microsoft Excel can accept more powerful solvers as plug-ins. The default is a throttled (limited in capacity) version of the new, high-efficiency Gurobi solver. The vision of the Solver Foundation initiative is to provide access to a variety of solvers in a uniform way through Microsoft products like Excel or languages such as Visual Basic and C#.

Another option focused largely on Excel is provided by a commercial vendor, Frontline Systems (http://www.solver.com). The company primarily targets LP practitioners and advertises themselves as the developers of Excel's solver.

It provides links to a wide variety of solvers (its own and other commercial solvers), all of which can be accessed from within Excel. For many operational uses of ATA, the Excel-based solution may provide an excellent vehicle for ease of use. For other applications, such as ATA research and simulations, the Excel-based solutions are generally a poor fit.

Advantages

- **Familiarity/ease of use**. Modules and plug-ins offer access to solvers via methods that are in widespread use. Users who routinely analyze data with SAS will find its OR module easy to use. Similarly, Excel is widely used. The ability to continue to use the worksheet format to formulate ATA problems can be a real advantage.
- **Leverage existing knowledge**. Time spent learning to access a solver via MATLAB is likely to generalize to facilitate using other features MATLAB. Similarly, some of the time spent using Excel-based ATA solutions generalizes well to more sophisticated use of Excel in other contexts. This cannot be said of modeling languages and OR systems, which are much more likely to yield knowledge that is specific to solving LP/IP problems.

Disadvantages

- **Limited control**. When using a solver in the context of a statistical analysis package or a plug-in, some of the various tuning parameters are likely to be unavailable to the user. When the solver proceeds normally to an acceptable solution, this progression is unlikely to be an issue. However, if difficulties are encountered in solving the IP problem, the user may have limited options for modifying the behavior of the solver to get an acceptable solution.
- **Cost**. The use of a plug-in requires that the user has the base software. For example, to make use of the plug-ins for Excel from Solver.com or Solver Foundation, the user must first have a copy of Excel. This requirement can be an issue for Unix/Linux users or for users of a different office suite. As with the other multisolver engine alternatives, it is important to know whether additional licenses must be purchased.
- **Performance**. Using another program to launch the solver can require quite a bit of overhead, both in terms of RAM consumed and start-up time for the base program. These concerns are likely to be less important for ATA practitioners than for ATA researchers.

Miscellaneous

Two kinds of products that do not fit into the categories above deserve description: ad hoc GUIs and enhanced computer languages.

Many of the open source and academic solvers have ad hoc GUIs. Most of these GUIs have been created by the author of the solver or another unpaid volunteer. As a result, these GUIs are fairly basic but usually functional. In many cases, the GUI introduces another layer of language that must be mastered. However, for users unaccustomed to using command line interfaces or writing control files, these GUIs can help make using the associated solver more familiar. In many of the programs with an ad hoc GUI, documentation can be sparse. It may take some searching of the installation, documentation, and/or Web site. If you find yourself in this situation, leave some time to explore — and have a tolerance of some frustration — in order to make use of these tools.

Enhanced computer languages sit between higher level modeling languages and modeling languages. The two main products in this niche are OptimJ (http://www.ateji.com/optimj.html) for the Java language and Tomnet (http://tomopt.com/tomnet/) for Microsoft's .NET platform. Both products add support within the target language to allow the user to write in a language, say Java, that is close to a modeling language in its form. For example, OptimJ generates Java source code from the OptimJ modeling code. The generated code is then incorporated into the project and compiled as regular Java. Both OptimJ and Tomnet are commercial products targeted at software developers writing business-specific applications that use solver technology. These products also can be useful for research programming, but you must carefully weigh the cost against the benefit. Again, consider the product-specific learning curve.

Comparison of Solvers

A Word on Linear Programming (LP) Versus Integer Programming (IP)

For solving ATA problems, the fundamental problem is to derive an integer solution; hence we want an IP solver engine. One of the confusing aspects of comparing solvers is that many of the IP solvers also require an LP solver because some IP methods (especially the cutting planes method) need to solve a series of LP problems. LP problems are substantially easier than IP problems, so the solution begins with a linear relaxation of the problem, and then the integer constraints are enforced. The best solvers incorporate a complex mixture of LP-based procedures with IP (such as the branch and bound method or the branch and cut method).

The use of an LP solver is important to bear in mind when considering the performance of IP solvers. The performance of many of these solvers is dependent upon speed and quality of the underlying LP solver. As an example of this phenomenon, the SCIP Web site (scip.zib.de) shows three different results for their solver, depending upon the LP solver that is used. Other solvers are less forthcoming about this aspect, and the dependence on an underlying LP solver is one of the reasons that make performance comparisons difficult.

Open Source

Open source software implies that the user can get access to the source code. Two slightly different versions of open source are available: Free Software (as produced or recognized by the Free Software Foundation [FSF]) and the Open Source Initiative (see Appendix B). The chief difference is in the terms of the licensing. For academic/research use, these differences are unlikely to cause issues. However, if you plan to use the solver to support commercial work, it is important to understand the difference in licensing.

COIN-OR (http://www.coin-or.org) is a loose collection of programs available for download. Some are available only as source code. Others have simple command line interfaces or rudimentary GUIs. The goal of COIN-OR is to foster open source collaborations of research and industry; much of the code was donated by IBM. In general, the site is not very user-friendly and is geared much more toward programmers/researchers than end users. Still, some nice things are available; see the CBC (http://www.coin-or.org/projects/Cbc.xml) and SYMPHONY (https://projects.coin-or.org/SYMPHONY) subprojects as a start.

lp_solve (http://lpsolve.sourceforge.net/5.5/) is an open project with solver to handle IP and LP problems. In addition to the basic solver (for LP and IP problems), lp_solve includes bindings for a number of languages, including Java, Python, R, MATLAB, and PHP. A simple but functional GUI is available for Windows users. There appears to be a dated plug-in to Excel (http://web.nps.navy.mil/buttrey/Software/Lpsolve/), but I have not tried it out. In research contexts, lp_solve is probably the most well-known solver engine and is widely used.

GLPK (http://www.gnu.org/software/glpk) is the GNU LP kit. The main software is a solver engine API. A command line interface is available. Probably most interesting is the GNU MathProg interface, which supports a subset of the AMPL modeling language. Some anecdotal evidence suggests that it has good performance relative to lp_solve. However, the GNU public license (GPL) places specific demands (that COIN-OR and lp_solve do not) on the user, which may or may not be acceptable. Consult the GLPK Web site for more details.

Academic Software

This class of software is available for download, but the license states that it may only be freely used for academic purposes.

QSOPT (http://www2.isye.gatech.edu/wcook/qsopt/index.html) is a solver for mixed IP. It has Java bindings available. Two GUIs are also available for download. The authors are essentially focused on a specific IP problem—the traveling salesman problem (http://en.wikipedia.org/wiki/Travelling_salesman_problem). QSOPT is free for academic and research use but not commercial use.

SCIP (http://scip.zib.de) stands for solving constraint integer problems. It is a solver for IP, but it makes explicit use of an LP solver as part of its solving process. SCIP makes use of a couple of additional approaches (constraint programming). An associated LP solver (SoPlex) is the default LP solver. The download page also includes versions that use QSOPT and the COIN-OR CLP solver. Practitioners will find much of the documentation opaque, as it is mostly directed toward programmers.

Minto (http://coral.ie.lehigh.edu/minto/index.html) was developed at Lehigh University. It is a mixed IP solver. The license allows use for education. The Web site can be a bit off-putting in that the licensing agreement is aggressively thrust in one's face before one is allowed to download the software. Little documentation about how to use the software is offered on the site, but some more is available in the download.

Commercial Software

Most of the commercial solvers indicate that they can handle non-LP as well as LP problems. This functionality is not necessary for ATA, and thus the user should be aware of paying for functionality that probably will not be needed.

LINDO (http://www.lindo.com) is one of the smaller companies. It has a modeling language, LINGO, and bindings for Java. LINDO supports plug-in functionality for Excel and features an additional product, What's Best, which is a spreadsheet-based decision aid. The student version (see Appendix A) of LINDO is much more constrained (less powerful) than several of the competitors.

KNITRO (http://www.ziena.com/knitro.htm) incorporates a solver that specializes in non-LP problems (as well as LP). This functionality is not necessary for ATA, and thus, the user should be aware of paying for functionality that probably will not be needed. KNITRO has links for AMPL modeling language and MATLAB.

Mosek (http://www.mosek.com) is a vendor of a commercial solver that handles both LP and non-LP programming problems. Mosek is unique in the large number of language bindings that they advertise (C/C++, .NET, Java, Python, and MATLAB).

CPlex (http://www-01.ibm.com/software/integration/optimization/cplex) is the market leader in solver technology. Until 2009, the solver was owned by Ilog. The company was acquired by IBM, and the takeover (including pricing) was complete as of July 2009. As an Ilog product, CPlex was expensive; IBM approximately doubled that price. In addition to the basic solver, CPlex has a modeling language, OPL, and Java bindings and offers an integrated modeling environment. All of the CPlex products are solid and have very good performance, but the price will put the solver out of reach of all but the largest commercial enterprises.

FICO (http://www.fico.com/en/Products/DMTools/Pages/FICO-Xpress-Optimization-Suite.aspx) is largely a financial organization, formerly known as Fair-Isaac. The solver was known as Dash Xpress-MP, and so some of the discussion/links involving this solver (e.g., Frontline Systems) may still refer to the old name. FICO also targets high-end business use, with an example of scheduling airplanes for a major airline featured prominently on their home page.

Gurobi (http://www.gurobi.com) is a recent entry into the commercial solver market. The company was founded by former staff from Ilog in the beginning of 2009. The solver is focused on mixed IP programming and so is appropriate for ATA use. Gurobi has aggressively sought out partners and is available via AMPL, OptimJ, and all of the major OR Systems. It is also noteworthy that a throttled version of the Gurobi solver is the default for the free version of Microsoft's Solver Foundation. Gurobi offers a free academic version (subject to rather tight licensing) for students and faculty. The main product is clearly targeted at the same large business market that CPlex and FICO are after, but Gurobi offers several alternative configurations that make it more attractive for many ATA uses.

Conclusion

As can be seen, you have access to a number of ways to solve the IP problem that lies at the heart of ATA. This paper touches on several factors that affect the choice:

1. How comfortable/proficient are you as a programmer? Low-level options require some skill but provide more control; higher level options allow you to stay focused on the substantive issues of your problem but may cost more.
2. What is the size of the problem? If it is relatively small (fewer than 300 – 500 items and fewer than 300 – 500 constraints), student versions of IP software or the default implementation of Solver Foundation may be useful; larger problems require higher end IP products.
3. How many problems do you need to solve? For an ongoing testing program that needs to solve a few assembly problems per year, a good but longer running solution (such as NEOS described in Appendix A) can be perfectly fine. However, if you are doing a simulation or want to use ATA in support of a CAT, short running time is an important requirement.

4. What context are you working in? Several of the software options have license terms that are liberal in an academic or research context. One the other hand, if you are working for a business or a not-for-profit that is using the results for products that it sells, only a subset of the options are available; it may not be acceptable to your employer to be forced to release source code it has paid to develop. If you plan to sell a product that includes a solver, these license issues become very important and require study as part of your decision making process.

I cannot offer a single, best solution; however, I hope this paper has given you enough information to frame the issue and narrow the field to make the most effective choice for your circumstances.

Acknowledgments

This paper was presented in the symposium, Implementation Issues in Automated Test Assembly, at the 2010 meeting of the National Council on Measurement in Education, Denver, Colorado, May 2010. The statements in this paper represent the personal opinions of the author and should not be taken to represent positions of Educational Testing Service nor CTB/McGraw-Hill. Note: This work was completed while the author was employed at CTB/McGraw-Hill.

Notes

1 This statement is generally true. Some of the higher level languages (described in subsequent sections) also describe APIs. Overall, API is a software/programming term, and its use implies some type of programming.
2 I am reminded in Terrance Parr's (2007) motto: "Why program by hand in five days what you can spend five years of your life automating?" (p. xv).
3 Or you have already paid for it, depending upon your view of taxes.

References

Brooks, F. P. (1995). *The mythical man-month: Essays on software engineering* (2nd ed., pp. 179–203). New York, NY: Addison-Wesley.

Fourer, R., Gay, D. M., & Kernighan, B. W. (2002). *AMPL: A modeling language for mathematical programming* (2nd ed.). Pacific Grove, CA: Duxbury Press.

Parr, T. (2007). *The definitive ANTLR reference: Building domain-specific languages*. Raleigh, NC: Pragmatic Bookshelf.

Raymond, E. S. (2001). *The cathedral and the bazaar: Musings on Linux and open source by an accidental revolutionary*. Sebastopol, CA: O'Reilly.

van der Linden, W. J. (2000). Constrained adaptive testing with shadow tests. In W. J. van der Linden & C. A. W. Glas (Eds.), *Computerized adaptive testing: Theory and practice* (pp. 27–52). Boston, MA: Kluwer.

van der Linden, W. J. (2005). *Linear models for optimal test design*. New York, NY: Springer.

Williams, S. (2002). *Free as in freedom: Richard Stallman's crusade of free software*. Sebastopol, CA: O'Reilly.

Appendix A

Additional Resources

Here are some additional sources of information to start your own explorations.

John Chinneck's Web Site

http://www.sce.carleton.ca/faculty/chinneck/StudentOR.html

Dr. Chinneck is a faculty member at Carleton University. I recommend his Web site as the first stop in learning more about LP/IP. It incorporates a draft version of a very readable book (*Practical Optimization: A Gentle Introduction*) and some interactive examples of various solution methods. The site also links to student versions (see below) of software as well as free/open source solvers. I very highly recommended this site as an entryway into the world of LP/IP and solvers.

Operations Research/Management Science Survey

http://www.lionhrtpub.com/orms/surveys/LP/LP-surveymain.html

The magazine *OR/MS* does an annual survey of solvers and solver systems. The survey is voluntary, but going through the tables gives a good overview of the (mostly commercial) alternatives. It also gives a feel for which features the vendors offer.

Student Versions

Virtually every product has some form of a student version. In general, these versions of the programs are limited in the number of constraints (typically 300 – 500) and the number of items (also 300 – 500). The modeling language AMPL and the OR systems (AIMMS and Maximal) also offer student versions. Also, as noted, Gurobi has a throttled version of their solver available as part of Microsoft's Solver Foundation.

Although the constraints (pardon the pun) on the student versions can put severe limitations on the nature of the problems that can be solved, student versions provide an excellent way to get a feel for ATA, how one builds the constraints specifies the objective function, and in general formulates an ATA problem.

Trial Licenses

Virtually all of the commercial products provide some type of trial license. Trial licenses typically run 15 – 30 days, so it is wise to make sure you have set aside the time to actually review the software before your trial license runs out.

NEOS

http://neos.mcs.anl.gov/

A unique resource in the LP world is NEOS. This site, run by Department of Energy at the Argonne National Laboratory, allows users to submit remote LP/IP problems. The problem is then submitted to the NEOS server, run, and the results returned. The batch processing is reminiscent of computing in the 1980s, but the price (free[3]) is right. A very wide variety of high quality solvers are available. The procedure for submitting jobs is too complicated to describe here, but tools to download on the site help ease the process.

Appendix B

Kinds of Software Licenses

This very brief summary identifies the main kinds of software licenses one is apt to encounter when searching for a solution to an ATA problem. These are only vague descriptions; nothing here is legal advice: Always read the actual license before you use the software.

Free Software

The free software movement was founded by Richard Stallman (Williams, 2002). Stallman believes that source code should be freely available and that proprietary software (i.e., commercial software) is morally evil. The free software movement is decidedly political. Why do you care? The Free Software Foundation (FSF) has produced two main software licenses: the GNU Public License (GPL) and the Lesser GNU Public License (LGPL); the lesser GNU license is somewhat more permissive. You *can* use GPL software to make money. But if you create a program that uses it, you have to release the source code of your program as well. In many academic settings, this is not an issue (indeed, we often try to sneak copies of our programs into people's briefcases when they are not looking in the hope that they will consider using them). But if you work for a company or other organization, you need to be conscious of what commitments you are implicitly making.

Open Source

The effects of open source software licenses are similar to the effect of free software licenses; you can download the program for free, and you can examine and modify the source code. However, open source is usually different in intent (Raymond,

2001). Open source is apolitical and generally business friendly. Most open source licenses allow you to take the code, make any changes and enhancements, incorporate it into a program of your own, and sell that program. Generally, the licenses require you to (a) give credit to the people and/or institution that developed the software and (b) agree not to sue them if something goes wrong while using the software. Some licenses also require that, if you find and fix any bugs in the code you obtained, you submit the code (for the bug only) back to the person/project from whom you obtained it.

Academic

Academic licenses essentially say that you can use the software for research, but you cannot use it to make a profit. Some are restricted to only be used at degree-granting institutions, while others allow any use for research, as long as it is not to make a profit.

Commercial

What we have come to think of as "regular software," sold. Although you think of buying Microsoft Word, you are actually buying a license to use Word. Commercial licenses allow you to use the program in your business but may put severe restrictions on the use, allowing it to only be run on a single, specific computer, for example. In addition to being expensive, the terms of commercial licensing can get very complex. Moreover, because large sums of money are usually involved, the vendors can be fairly aggressive in enforcing the license.

Why Do You Care?

If you are just going to take a program and run it, you may not need to care. However, if you are going to modify a program, using it as part of a larger program that you are creating, or working for a business, the license may limit which solvers are available to you. Again, always read the actual license before you use the software.

Suggested citation:

Donoghue, J. R. (2014). *Comparison of integer programming* (*IP*) *solvers for automated test assembly* (ATA) (ETS Research Report No. RR-15-05). Princeton, NJ: Educational Testing Service. doi:10.1002/ets2.12051

Action Editor: James Carlson

Reviewers: Peter van Rijn and Frank Rijmen