Comment on Three-Element Item Selection Procedures for Multiple Forms Assembly: An Item Matching Approach

Applied Psychological Measurement

2016, Vol. 40(8) 641–649

© The Author(s) 2016

Reprints and permissions:

sagepub.com/journalsPermissions.nav

DOI: 10.1177/0146621616664075

apm.sagepub.com

Wim J. van der Linden¹ and Jie Li²

Abstract

A recent article in this journal addressed the choice between specialized heuristics and mixed-integer programming (MIP) solvers for automated test assembly. This reaction is to comment on the mischaracterization of the general nature of MIP solvers in this article, highlight the quite inefficient modeling of the test-assembly problems used in its empirical examples, and counter these examples by presenting the MIP solutions for a set of 35 real-world multiple-form assembly problems.

Keywords

automated test assembly, item matching, mixed-integer programming, MIP solvers, multiple-form assembly, parallel test forms, specialized heuristics

Introduction

Test-assembly problems consist of the selection of a set of one or more test forms from an item pool that has to be optimal with respect to some objective and at the same time meet a set of constraints representing all their other specifications. Attempts to automate the test-assembly process for operational testing programs has taught us that the required constraint set can be much larger and more varied than one might expect naively. Generally, the set is supposed to deal not only with the content blueprint and psychometric requirements typically met in manual test assembly but also with each of the requirements imposed more implicitly during the process, such as formatting rules, desired degree of item overlap between forms, limits on item-exposure rates, the selection of anchor items, logical requirements due to the presence of sets of “enemy items” or items organized around common stimuli, uniform answer key distributions, the necessity to meet a given time limit, and so on (for a complete review, see van der Linden, 2005). Omitting constraints that may seem most mundane from a content or psychometric perspective immediately kills the result from any automated test assembly (ATA). For example, if

¹Pacific Metrics Corporation, Monterey, CA, USA

²McGraw-Hill Education, Monterey, CA, USA

Corresponding Author:

Wim J. van der Linden, Pacific Metrics Corporation, 1 Lower Ragsdale Road, Bldg. 1, Ste. 150, Monterey, CA 93940, USA.

Email: wvanderlinden@pacificmetrics.com

a testing program has to keep the number of pages in its paper forms constant across years and the constraint set does not bound the total word count or lines of text for the items from below and above, no matter how well all other specifications are met, the results are just useless.

It may be tempting to think that, because of the nature of their possible constraints, optimization problems in ATA are unique, but they are not. These problems share their formal structure with combinatorial optimization problems daily solved in almost any other industry, business, or commerce. In fact, the testing industry has been quite late to arrive at the scene; numerous others have preceded and adopted the use of the same mixed-integer programming (MIP) solvers for their problems much earlier. It should thus not surprise us that most of them have already gone through extensive discussions of the same question of whether specialized heuristics or general MIP solvers are to be preferred as addressed for ATA in a recent article by Chen (2016) in this journal.

It is clear from Chen's article that she favors her specialized heuristic. The current authors would not have much of a problem with that as long as it is used repetitively for the same small problem addressed in her article. But they disagree with her general conclusions about the use of MIP solvers in her article. The conclusions not only go against a major trend in the field of constrained combinatorial optimization where these solvers have been developed but seem to be based on a serious mischaracterization of what they actually are and are based on results for empirical examples that appear to be both fundamentally flawed and irrelevant. But before presenting the objections in more detail, the general nature of the distinction between MIP solvers and specialized heuristics is first highlighted.

MIP Solvers Versus Specialized Heuristics

The development in the field of combinatorial optimization from specialized heuristics to general MIP solvers has mainly been the result of the dramatic improvements in the performances of the latter in the last two decades or so. Before then, practitioners had to develop their own heuristics to resolve most of their real-world problems. A recent article by Bixby (2012), one of the chief engineers of the solvers in the *CPLEX* (International Business Machines Corporation, 2015) and *Gurobi* (Gurobi Optimization, Inc., 2015) software, gives a nice personal account of the improvements since the first implementations of branch-and-bound in the 1950 to 1960s on 36-bit machines with 32K words of storage through the current mix of problem preprocessing, capitalization on special problem structures, and efficient implementation of branch-and-cut used in the modern solvers (for an introduction to these methods, see Chen, Batson, & Dang, 2010). Using a set of 1,982 benchmark problems, Bixby was able to calculate the improvement in performance between the launch of the first versions of *CPLEX* in 1991 and *Gurobi* in 2009 to be greater than the factor of 5,280,000. The endpoint of this period is already 6 years behind us, and the improvements continue while this comment is written. Notice that this factor is for machine-independent improvement only. Multiply it with the improvements due to the dramatic increase in CPU performance, and the arrival of multiple-core processing in the same period and the result is just mind boggling. In fact, the performance of the MIP solver in *CPLEX* has increased so much that now its memory management is in need of further optimization; for large problems, it occasionally stops because of memory overflow. However, the issue was already fixed in the first version of *Gurobi* and is unlikely to happen for the solver in *XPress* (Fair Isaac Corporation, 2015), which in the experience of the authors currently tends to be faster than the previous two. Still, the current authors' observations should not be taken to guarantee solution to full optimality of *any* problem in reasonable time; more on this later in this comment.

Dramatic increase of performance is not the only reason why users have moved to MIP solvers. They are also truly general in the sense that it is possible to move from one type of

problem to a completely different type without having to recode any software. In addition, they are embedded in software packages that increasingly offer additional services, such as a supporting modeling language, choice of alternative settings with known effects, and infeasibility analysis of overconstrained problems.

Heuristics are on the opposite end of the continuum. They are built specifically to solve one type of problem, require extensive simulations with different data sets to tune them, and are not guaranteed to produce optimal solutions. More importantly, they tend to violate the constraint set, especially when it is large and varied. For instance, the current authors are not aware of any ATA heuristic that satisfactorily deals with constraints on quantitative parameters (e.g., word counts, response times, psychometric parameters) or logical constraints required to deal with conditional item selection. To illustrate the latter, Chen's heuristic runs into problems when the assumption of no overlap between test forms in her examples has to be replaced by a maximum number of common items in different forms but none of the items figuring in more than two forms. Add the possibility that some of the items in the pool are organized as sets around common stimuli and the number selected per set has to be between bounds and the heuristic completely breaks down.

It is tempting to think of these practical issues as a challenge and try generalizing heuristic ideas by providing them with more general computer code and options to deal with special applications. Some authors have actually gone this direction. But modern MIP solvers are precisely the current endpoint of the development toward general computer code! Why would we ever want to repeat their 60 years of history (and still miss the fundamental point that they allow us to replace computer programming by much more efficient mathematical programming)?

Specific Comments

Our first comment is on Chen's misinterpretation of the fact that test-form construction problems are known to be NP-hard. She has taken this rather abstract concept from computational complexity theory to predict that "computation time . . . increases exponentially with problem size" (p. 115). Her next sentences claim the same concept to imply that even "if the problem size is small, some problem types, such as the minimax model, are known to be difficult to solve." First of all, there exist no such thing as "the minimax model." Minimax is just the principle of minimizing the maximum distance between a sequence of function values or parameters and a target for them. It can be applied to any type of problem that has such sequences. More importantly, however, concepts as polynomial time (P), nondeterministic polynomial-time (NP), and NP-hard apply to upper bounds on computational time for worst cases in large abstract classes of decision problems and any algorithm that is nondeterministic (i.e., has choice points that need to be addressed by the user when implementing it; Nemhauser & Wolsey, 1999, chap. I.5). Specifically, NP refers to an upper bound on the time needed to verify a solution found by any algorithm for any problems in this class, not the time to find it. The subset of hard problems in a class of NP problems has the feature that if any of its problems can be solved in polynomial time, the same holds for any other problem in the same class. Whether this is actually possible (i.e., if $P = NP$) is still one of the open problems in computer science. But even if the equivalence would be proven for a large class of problems, the result would not enable us to predict the runtime of any real-world algorithm for each of its instances: The fact that problems can be solved in polynomial time does not tell us anything about the order of the polynomial (except that it is finite), let alone imply that any algorithm finds their solutions fast. On the contrary, as just highlighted for the history of MIP solvers, it is perfectly possible to rigorously optimize the performance of nondeterministic algorithms by making the right implementation choices.

Our second comment is on the problem of item matching used by Chen to show the superiority of her heuristic. The limited nature of this specific problem shows a stark contrast with the set of 1,982 different problems used in Bixby's evaluation. Neither is it representative of the everyday practice of test assembly. Actually, her problem goes back to a historic method known as Gulliksen's matched random subtests method, introduced by him in the very first edition of Gulliksen (1950) to deal with the problem of estimating the classical test reliability coefficient. As it is only possible to estimate a lower bound to the coefficient from a single administration, Gulliksen came up with the idea of maximizing the bound by splitting the test into two halves with items matched on the Euclidean distance between their pairs of p values and r_{pbis} discrimination indices and adjusting their correlation for test length. van der Linden and Boekkooi-Timminga (1988) showed that Gulliksen's method, which basically was a graphical procedure, could be modeled using MIP. However, at the time, due to statistical improvements on estimates of lower bounds of reliability coefficients, Gulliksen's method was already beginning to lose its relevance. The idea of item matching was used only incidentally at later times, specifically by Armstrong, Jones, and Wu (1992) as first step in their heuristic for item response theory (IRT)-based test assembly and Ariel, Veldkamp, and van der Linden (2004) to split an item pool into rotating parts for an adaptive testing program.

Gulliksen's method does not make much sense outside its historic context. Particularly its application to IRT-based selection of parallel test forms from a larger pool of calibrated items is an example of inefficient modeling. In IRT, test forms are considered as statistically parallel when they have identical test information functions (TIFs) (and contentwise parallel when they meet all other constraints). Requiring forms to be item-by-item parallel with respect to IRT item parameters does not add anything to this definition of parallelness; on the contrary, it leads to seriously overconstrained problems with an unnecessary, extremely large set of variables. To be more specific, efficient MIP modeling of the problem of assembling a set of test forms meeting the same target for their TIF is possible with the number of variables equal to the product of the size of the item pool and the number of forms. If item matching is added, the product has to be extended with the length of the forms. For the examples in Chen's first study, with 500 items in the pool and five forms of 40 items each, the extra factor leads to an increase from an efficient model with 2,500 variables to an extremely inefficient one with 100,000 variables. Similar inefficiencies hold for the number of constraints.

The models in Chen's article suffer from other aberrancies, too. For instance, her objective function in Equation 6 has a serious scaling problem in that it minimizes the sum of two entirely different bounds—one on the sum of inter-item distances between each pair of forms and another on the sum across all forms. The latter is much bigger and can thus be expected to have been quite dominant in her examples. A simple alternative would have been to minimize a single common bound on the sum of the differences between each of the forms and the target test. The alternative would have reduced the number of constraints in Chen's Equations 7 and 8 by a factor of two and 35 in the versions of the examples used in her first and second studies, respectively.

The model in Chen's Equations 9 to 13 was presented as an example of a minimax model based on TIFs. However, again, rather than minimaxing with respect to the differences between the TIF for each of the test forms and their common target, the author chose to minimize the sum of two quite unequal bounds. Surprisingly, one of them is a bound on the difference between the sum of the TIFs across all forms and their common target. It is completely unclear what could be the purpose of minimizing this bound. Another problematic aspect of the model is its control of the TIFs at no fewer than 13 points along the ability scale. Numerous published studies have shown that, as information functions are smooth, well-behaved functions, control at 3 to 5 points is sufficient to get perfect results.

The current authors' final comments are on Chen's choice of examples in her three studies. If the purpose of a study is to evaluate the performances of different algorithms with respect to each other, we expect them to be compared for exactly the same set of problems. Bixby's comparison of different versions of MIP solvers for the same large set of benchmark problems is a good example of this type of study but Chen's first study immediately violates this obvious methodological principle. The six different cases compared in it consisted both of different models and heuristics/MIP instances. The fact that they share a common subset of constraints forcing the numbers of items in three content categories to be between identical bounds does not eliminate the huge differences between their numbers of variables and the rest of their constraints. It does not surprise us that the "minimax model" realized smaller differences between the TIFs of the five test forms than any of the four heuristics and the other MIP formulation used in this study (Chen, 2016, Figure 1). After all, it was the only model that directly attacked these differences. (It would have been nice to see the target for the TIFs of these five forms plotted in the same figure, though.) However, the subsequent comparison between the test characteristic functions (TCFs) produced for each of the six cases (Chen, 2016, Figure 2) did surprise us. Why TCFs all of a sudden? Because Chen's own heuristic now performed best? If the objective was to have equal TCFs for each of the assembled test forms, the problem should have been modeled to minimize their differences.

In the second study, the item pool size was systematically increased to include sizes of 500, 1,000, 3,000 and 6,000 items, the reference tests had 30 or 60 items, and the number of test forms to be assembled was five or 10. In addition, three cognitive categories were added to the three content categories. This time, no plots of the TIFs for the solutions are provided. The only thing that can be learned is that two of the heuristics had massive failure rates (27.1%-93.79%) for the case of three content categories only and failed completely when the three cognitive categories were added. As for the runtimes, the reader is only told that the MIP model "generally took much more time than did the other heuristic methods," with the exception of the case of the reference test of 60 items and 10 parallel forms when *CPLEX* stopped after 80 min (content categories only) and 40 min (cognitive categories added) because it ran out of memory.

The third study was mainly to repeat the same methods under "more practical conditions." The item pools were now simulated to have one set of two or three enemies per 100 items and a fixed response time for each item. It is unclear how the heuristics dealt with the enemies or the bounds on the total time for the test. The typical heuristic approach is to suppress all other enemies in a set for a form as soon as one of them is selected—an approach that does avoid the presence of enemies in the same form but is suboptimal because it accepts the first item selected from each set, not the ones that are best with respect to the objective function and constraints. A similar heuristic trick must have been used to deal with the constraint on the total time. No results for the objective function values and constraint satisfaction are given. The only thing that is learned is that the runtimes were "very similar" to the previous study and that *CPLEX* stopped after 44 min for the case of the reference test of 60 items and 10 parallel forms because of memory overflow.

Counterexamples

Chen's use of MIP modeling in these examples was extremely inefficient. How inefficient exactly remains unknown as we are not informed about the exact numbers of variables and constraints in her studies. The number of variables in the item matching model in her first study has been already estimated to be equal to 100,000. For the largest cases in her second and third study, the number must have been 36 times as great (product of $12 \times 1.5 \times 2$ for the increase in

item pool size, length of the reference test, and number of forms). It is not shocking to learn that *CPLEX* incidentally ran out of memory for an example with no fewer than 3,600,000 variables.

To give the readers of Chen's article a more balanced impression of the use of MIP in ATA, the current authors refer to Table 1. The table reports results for the last 35 test-assembly problems with sets of at least three parallel forms from various summative and interim assessments programs at the current authors' organizations (with the exception of the few cases where the forms were permitted to have item overlap, a case not addressed in Chen's article). Each of the forms in a set had to meet the same content blueprint for a subject area as mathematics, English language arts, reading, writing, science, or history/social science. All other constraints were for such item attributes as IRT parameters, information-function values, p values, point biserials, fit statistics, differential item functioning (DIF) statistics, depth-of-knowledge levels, and/or answer keys. Formally, the types of constraint varied from constraints that set lower and upper bounds on subsets of items, numbers of items per stimulus, and averages or sums of item and test attributes to constraints necessary to deal with all-or-none selection of specific subsets of items or subsets of stimuli and/or items that mutually excluded each other. The objective functions were of the minimax type with the targets either for the TIFs or TCFs of the test forms in use for the assessment programs. The pool size ranged from 165 to 1,647 items, the number of stimuli per pool from 2 to 278, and the number of forms assembled simultaneously from 3 to 6 forms. The numbers of MIP variables and constraints varied between 546 to 14,779 and 1,215 to 19,357, respectively.

The test developers used an in-house ATA system to input all their test specifications, which the system then translated into a MIP model that was solved using the *CPLEX 12.3* solver on a Linux server machine. For each of the problems, the default *CPLEX* settings were used, except that (a) the relative MIP gap tolerance was set to 2%, (b) the optimizer time limit was sometimes set to a user value (default is 1 min), (c) the parallel search mode was set to optimistic (for faster performance in multi-threads environment), (d) the node file on disk and compressed was switched on (for saving memory in solving large problems), and (e) the tree limit parameter was set to 10,000 (10GB of space; to prevent oversized problems from overconsuming system resources). The choice to override the default of 1 min for the time limit by a user value was made by the test developers based on their previous experience with similar runs. The choice of 2% for the relative MIP gap tolerance was the standard choice for all assessment programs based on inspection of the results for numerous forms assembled in earlier studies. The output of the ATA system consisted of a report with the items selected, various plots based on the TIFs or TCFs, and distributions of the item attributes for the test forms. Of course, each of the constraints was always met. The test developers reviewed the report and accepted the forms for use in their program in each of these 35 examples.

Table 1 reports the times that *CPLEX* ran either to 2% gap optimality or the best feasible solution after the test developer's time limit was exceeded. For all examples with the time limit set by the test developers, the TIFs or TCFs were deemed to be close enough to their targets by the test developers not to rerun the solver with tighter settings.

Just to make sure, the results in Table 1 were for fixed-form test assembly only. The use of MIP solvers in the shadow-test approach to adaptive testing, with its re-assembly of a full-size shadow test prior to the selection of each item, is an entirely different type of application. As each next shadow test has to meet the same constraint set and differs only in the ability estimate at which the information is maximized, the solver uses the previous shadow test as the first feasible solution when assembling the next shadow test. This choice of a "hot start" for the solver leads to a phenomenal reduction in its runtime. In a recent study, Choi, Moellering, Li, and van der Linden (2016) reported times in the range of 40 to 50 ms per item selected.

Table 1. Results for 35 Different Test Assembly Problems Each Involving the Simultaneous Assembly of a Set of Three Parallel Forms.

Example	Number of items in pool	Number of stimuli in pool	Number of forms selected	Number of items selected	Number of MIP variables	Number of MIP constraints	User limit	Runtime
1	480	12	3	105	2,922	3,742	30'	30' 0"
2	1,296	7	3	127	7,827	8,613	12'	12' 30"
3	1,328	16	3	127	8,073	8,977	12'	12' 27"
4	1,296	14	3	127	7,869	8,749	10'	10' 1"
5	1,462	38	4	160	14,779	19,115	1'	1' 1"
6	275	5	6	42	1,689	5,355	1'	1' 0"
7	473	8	3	105	2,868	3,608	1'	1' 0"
8	399	16	3	109	2,448	3,334	1'	1' 0"
9	378	5	3	111	2,295	3,706	1'	1' 0"
10	409	10	3	107	2,490	3,405	40'	15' 12"
11	1,647	278	3	141	11,657	19,357	60'	14' 31"
12	533	30	3	32	3,387	5,000	1'	0' 45"
13	543	32	3	32	3,459	5,124	1'	0' 31"
14	423	19	3	32	2,661	3,859	1'	0' 18"
15	237	27	4	160	1,340	3,556	1'	0' 16"
16	423	22	3	32	2,679	3,904	1'	0' 11"
17	423	21	3	37	2,607	3,822	1'	0' 10"
18	423	21	3	32	2,673	3,894	1'	0' 6"
19	423	22	3	37	2,610	3,844	1'	0' 5"
20	423	19	3	37	2,601	3,794	1'	0' 3"
21	533	30	3	37	3,298	4,911	1'	0' 2"
22	543	32	3	37	3,360	5,019	1'	0' 2"
23	165	15	3	123	546	1,215	1'	0' 1"
24	1,015	119	3	132	3,810	6,945	1'	0' 1"
25	1,032	29	3	93	6,285	7,383	1'	0' 1"
26	1,027	27	3	93	6,333	7,383	1'	0' 1"
27	1,162	23	3	93	7,119	8,070	1'	0' 1"
28	175	29	3	120	618	1,533	1'	0' 0"
29	759	2	3	97	4,566	5,325	1'	0' 0"
30	531	36	3	32	3,411	5,082	1'	0' 0"
31	531	36	3	37	3,304	4,969	1'	0' 0"
32	948	110	3	141	3,558	6,366	1'	0' 0"
33	1,106	19	3	93	6,699	7,512	1'	0' 0"
34	1,107	29	3	90	6,735	7,839	1'	0' 0"
35	1,198	27	3	93	7,275	8,313	1'	0' 0"

Note. MIP = mixed-integer programming.

Conclusion

Due to continued optimization of their performances, MIP solvers are now generally used throughout the field of constrained combinatorial optimization. On the other hand, heuristics are typically built for one type of problem, often with subsequent efforts to generalize their use when the area of application widens. They are fast just because they do not enumerate the entire set of feasible solutions, but consequently err both on the sides of the objective function and the constraints. In addition, they need to be retuned each time the problem changes more substantively than just a different numerical value for the bounds in some of the constraints. Actually, reporting their runtime rather than the application time, as in Chen's article, is misleading. From the point of view of a practitioner, what counts is the total time required to prepare the heuristic and run the application for each new problem.

Instead of developing a heuristic for the incidental cases in which a solver does not yet produce a solution in realistic time, these authors recommend adjusting the settings of the solver. This alternative works for any test-assembly problem without violating any of its constraints. The only effect is a slightly less than optimal value for the objective function. The gap parameter is especially an effective parameter to try. For an objective function of the minimax type, its setting at 2% guarantees a greatest deviation of the TIF or TCF from their target for any of the forms in the set not larger than 2% of the greatest deviation in the optimal solution—a result clearly better than for any of the curves in Figures 1 and 2 in Chen's article. In fact, the parameter could easily be set at higher percentages without any noticeable visual impact on the shape of the TIF or TCF. But the current authors have found 2% to be a good compromise. They expect to be able to set even lower percentages for the next generation of MIP solvers that currently dominate the market.

Declaration of Conflicting Interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) received no financial support for the research, authorship, and/or publication of this article.

References

- Ariel, A., Veldkamp, B. P., & van der Linden, W. J. (2004). Constructing rotating item pools for constrained adaptive testing. *Journal of Educational Measurement*, 41, 345-359.
- Armstrong, R. D., Jones, D. H., & Wu, I. L. (1992). An automated test development of parallel tests from a seed test. *Psychometrika*, 57, 271-288.
- Bixby, R. E. (2012). A brief history of linear and mixed-integer programming. *Documenta Mathematica, Extra Volume ISMP*, 107-121.
- Chen, D. S., Batson, R. H., & Dang, Y. (2010). *Applied integer programming*. New York, NY: John Wiley.
- Chen, P. H. (2016). Three-element item selection procedures for multiple forms assembly: An item matching approach. *Applied Psychological Measurement*, 40, 114-127.
- Choi, S., Moellering, K., Li, J., & van der Linden, W. J. (2016). Optimal reassembly of shadow tests in computerized adaptive testing. *Applied Psychological Measurement*. Advance online publication. doi:10.1177/0146621616654597
- Fair Isaac Corporation. (2015). *Xpress-Optimizer reference manual*. Retrieved from <http://www.fico.com/en/products/fico-xpress-optimization-suite>

- Gulliksen, H. (1950). *Theory of mental tests*. New York, NY: John Wiley. (Reprinted in 1987 by Hillsdale, NJ: Lawrence Erlbaum).
- Gurobi Optimization, Inc. (2015). *Gurobi optimizer reference manual*. Available from <http://www.gurobi.com>
- International Business Machines Corporation. (2015). *IBM ILOG CPLEX optimization studio*. Retrieved from <http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/>
- Nemhauser, G. L., & Wolsey, L. A. (1999). *Integer and combinatorial optimization*. New York, NY: John Wiley.
- van der Linden, W. J. (2005). *Linear models for optimal test design*. New York, NY: Springer.
- van der Linden, W. J., & Boekkooi-Timminga, E. (1988). A zero-one programming approach to Guiliksen's matched random subtests method. *Applied Psychological Measurement*, 12, 201-209.