Introduction to Shadow-Test Approach to Adaptive Testing

Wim J. van der Linden

Introduction

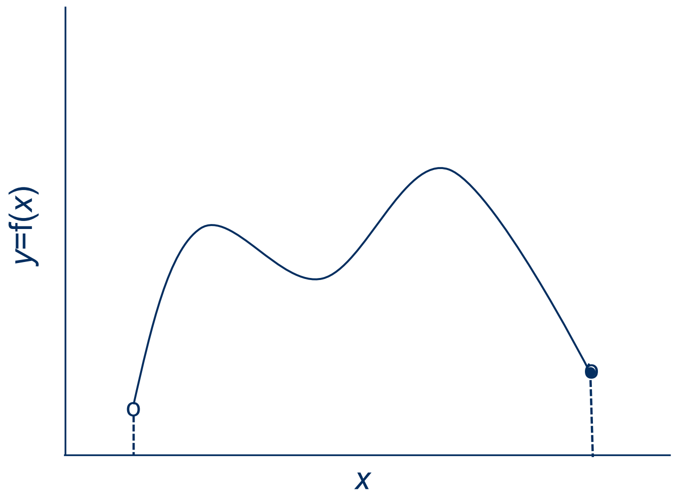
The distinctive feature of adaptive testing is sequential optimization of item selection.

After each new response, the test taker’s current ability estimate is updated and the next item is selected to be optimal at the new estimate.

Mathematically, the optimization process is quite different from what we have learned in our calculus classes.

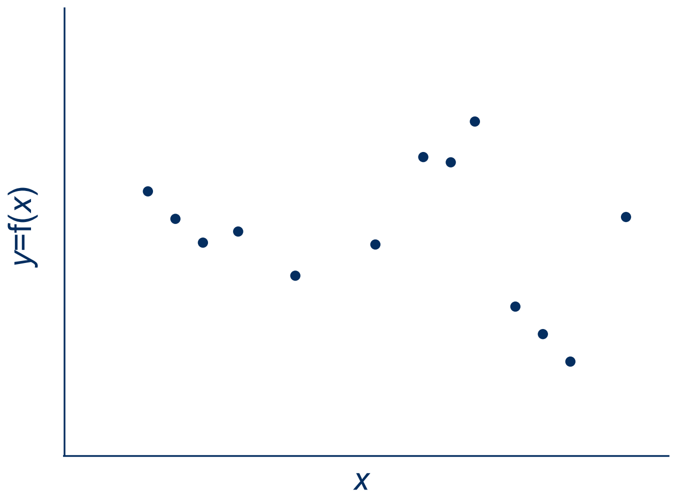
Mathematical Optimization

In calculus, the typical optimization problem is to find maxima and/or minima of a function *y*=*f*(*x*) over a real interval of *x*.



Mathematical Optimization

Optimization over a set of discrete values of *x* does not require calculus but an enumerative algorithm that checks each discrete point in the domain to find the point with the best function value.



Mathematical Optimization

The problem gets more and more complicated if the number of variables that define the domain increases.

In fact, the number of points that need to be checked growths exponentially with the number of variables.

For 80 discrete variables, each with 10 values, the number of points is already greater than the number of atoms in the universe!

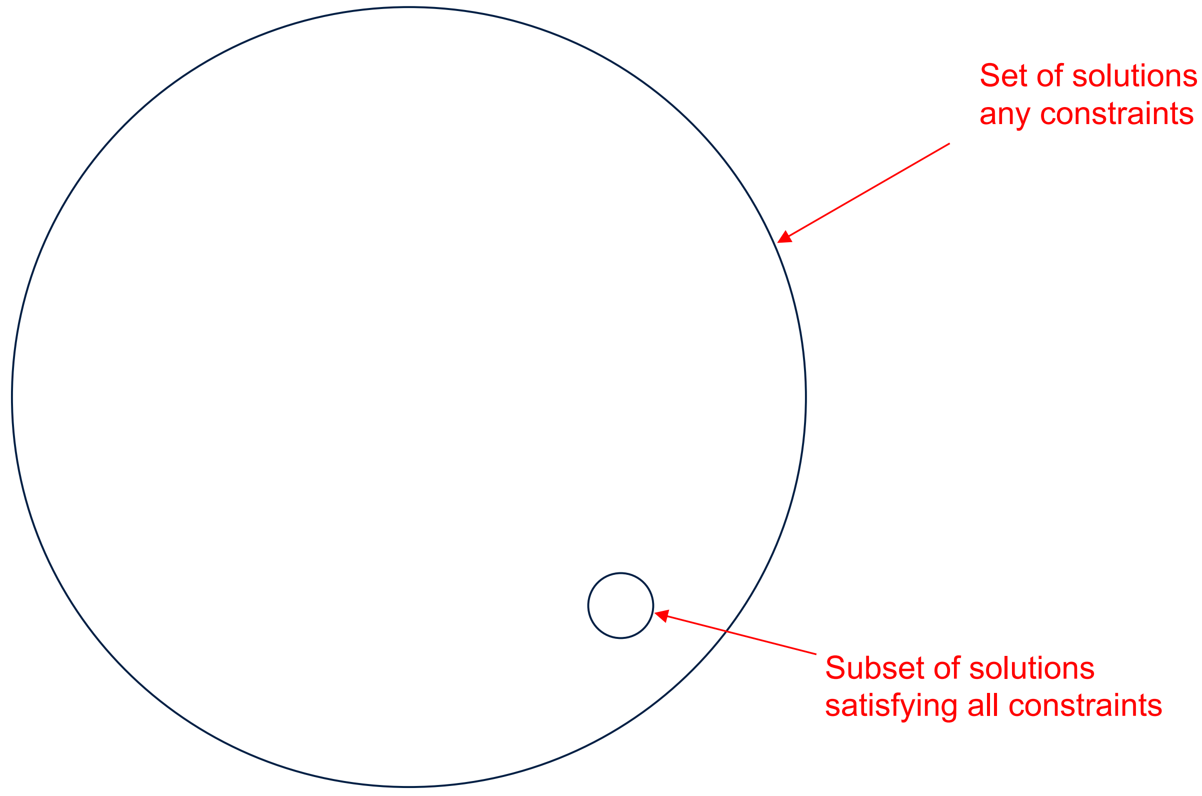
Mathematical Optimization

The problem gets even worse when the solution has to satisfy a number constraints, as is the case in automated test assembly where we typically want the result to satisfy large numbers of content and statistical specifications.

The subset of feasible solutions is then dramatically smaller than the space of all possible solutions but, for a realistic size of item pool, still astronomically large.

In fact, a naive search for just one feasible solution may then already take for ever.

Space of Solutions with Feasible Subset


Set of solutions without any constraints
Subset of solutions satisfying all constraints

Mathematical Optimization

Thanks to a computational revolution, we now have solvers that find solutions to large test-assembly problems to a required level of accuracy in a few seconds.

These solvers, which have been developed in the field of *mixed integer programming* (MIP), are available both in open-source and as commercial software (watch our next presentation!).

Mathematical Optimization

MIP solvers systematically search a tree with all possible solutions using *backtracking.* They go back to a previous node in the tree when they hit an infeasible solution or the value for the objective function gets lower than the best solution so far.

The dramatic improvement in the running times for the solvers has been made possible through powerful preprocessing of the optimization problem and optimal implementation of the algorithm (initialization; branch pruning; cutting planes; etc.)

Test-Assembly Problem

The steps necessary to solve a test-assembly problem are:

- choice of decision variables (mostly 0-1 variables);
- use of the variables to model the objective function and each of the constraint;
- feeding the model into a MIP solver to obtain the solution.

Decision variables

Item	1	2	…	i	…	99	100
Variable	x_1	x_2	…	x_i	…	x_{99}	x_{100}
Form 1	0	0	…	..	…	0	0
Form 2	0	0	…	…	…	0	1
Form 3	0	0	…	…	…	1	1
…	…	…	…	…	…	…	…
Form 2^{100}	1	1	..	…	…	1	1

Example of Optimization Model

test length:	$\sum_{i=1}^{100} x_i$
# of items on knowledge:	$\sum_{i=1}^{50} x_i$
# of items on applications:	$\sum_{i=51}^{100} x_i$
expected total time on test:	$\sum_{i=1}^{100} t_i x_i$

Example of Optimization Model

maximize $$\sum_{i=1}^{100} x_i$$ (maximal test length)

subject to

$$\sum_{i=1}^{50} x_i \geq 10$$ (# items on knowledge)

$$\sum_{i=51}^{100} x_i \geq 10$$ (# items on applications)

$$\sum_{i=1}^{100} t_i x_i \leq 60$$ (time slot)

$$x_i \in \{0,1\}, \qquad i = 1, \dots, 100$$ (range of decision variables)

Example of Optimization Model

The solution produced by the solver is a string of 0's and 1's identifying the test that satisfies each of the constraints and has an optimal value for the objective function.

Automated test-assembly software packages help you to specify the objective function and constraints without the use of any mathematics.

They also make a call to the solver and help you to analyze the solution.

Automated Test Assembly

This was only an unrealistically small example of a test-assembly problem.

MIP has been used to solve an extremely large variation of real-world test-assembly problems, including:

- every realistic type of constraint and objective function;
- item sets with a common stimulus;
- sets of multiple parallel tests;
- enemy items;

Automated Test Assembly

- sets of tests with optimal anchor items;
- formatted, printable test forms
- item pools assembled from an inventory;
- pre-equated test forms;
- test forms with automatically generated items:
- test forms with predetermined level of speededness;
- et cetera.

Adaptive Test Assembly

Adaptive test assembly is even more complicated than fixed-form assembly.

In addition to discrete variables and large numbers of constraints, the solution now must be found *sequentially*, instead of *simultaneously* as for fixed forms.

Not being able to backtrack and replace earlier items, we are immediately faced with the dilemma between the selection of suboptimal items and violation of the set of constraints.

Shadow-Test Approach

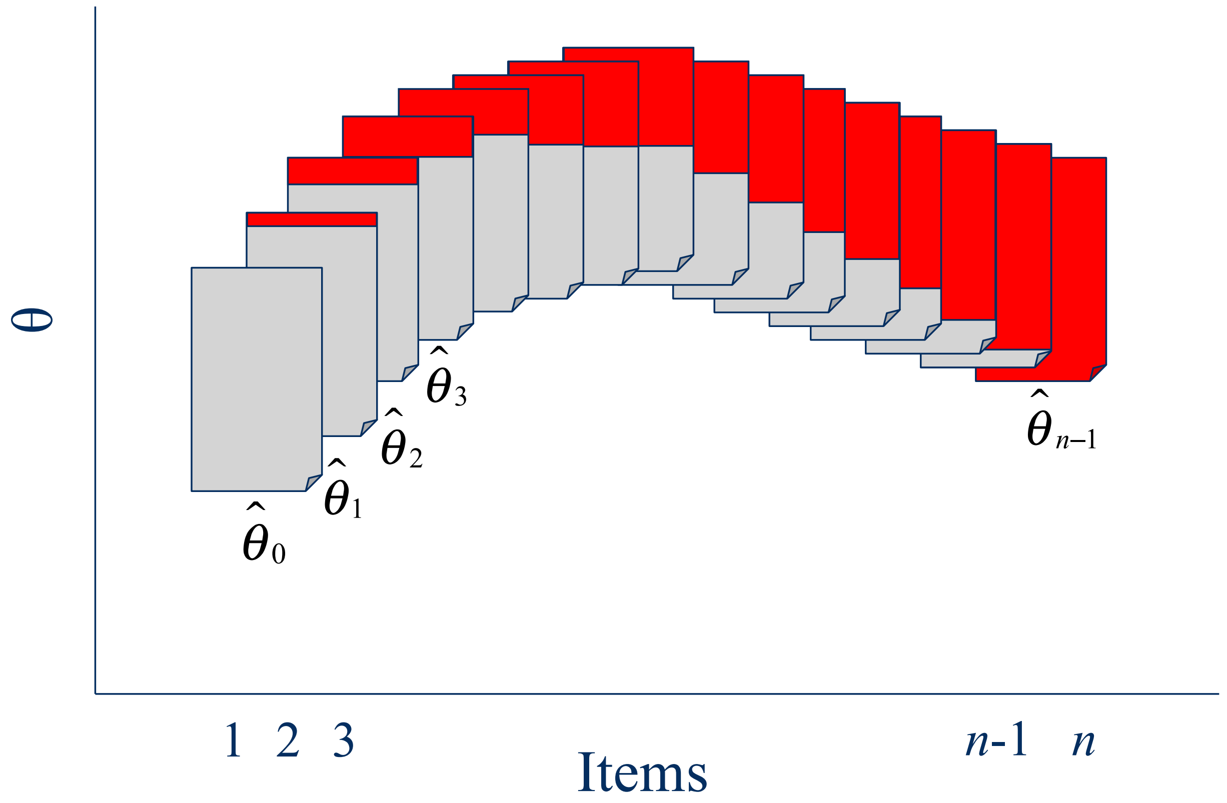
The (somewhat counterintuitive) *shadow-test approach* resolves the dilemma by looking ahead!

It conceives of the test-assembly process not as a sequence of individual items but of *complete test forms*.

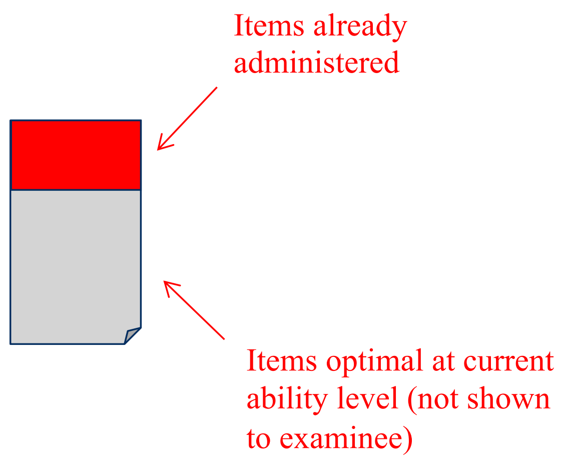
Its alternate steps are:

- selection of an optimal full test at the ability estimate;
- administration of the best free item in the test:
- update of the ability estimate;
- and so on.

Shadow-Test Approach



Shadow-Test Approach


Items already administered
Items optimal at current ability level (not shown to examinee)

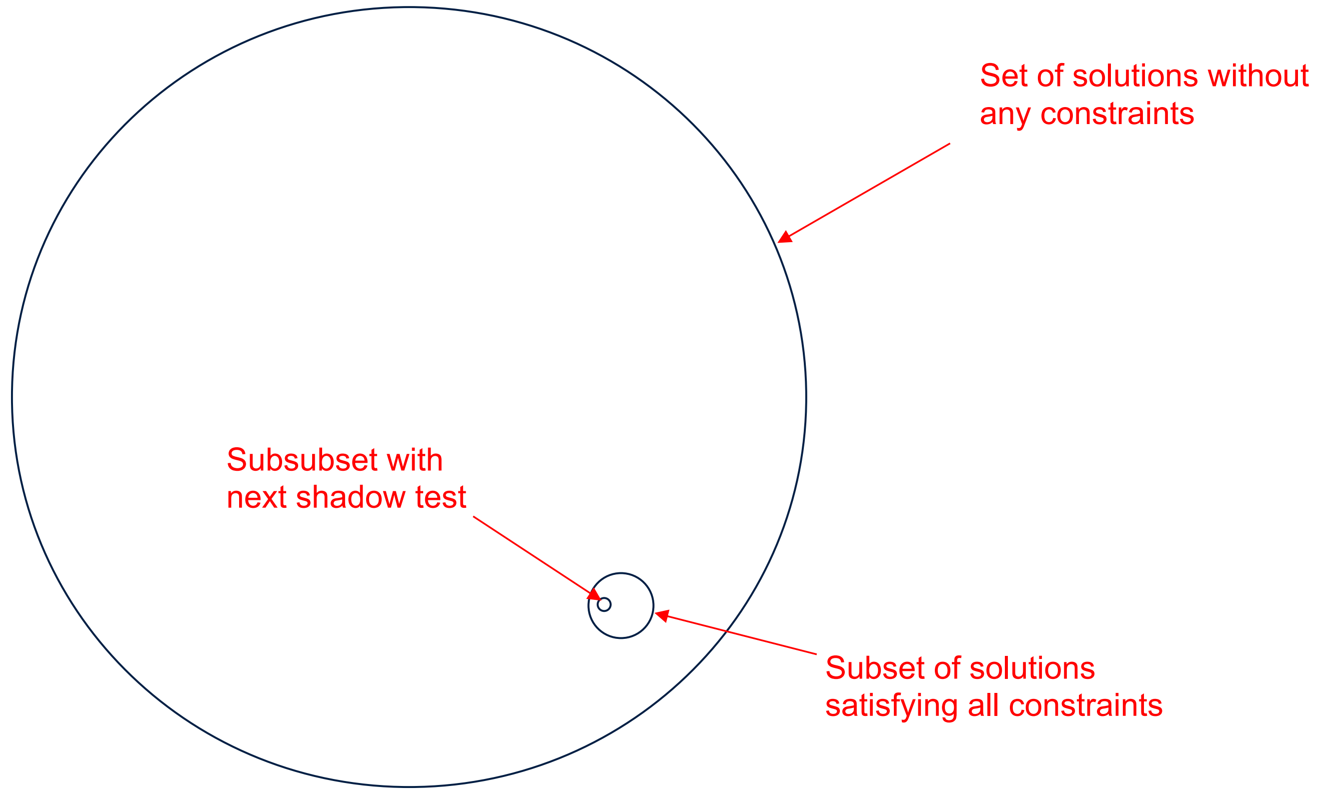
Shadow-Test Approach

It may look as if the approach is unrealistic due to all necessary calls to the MIP solver, one after each item for each examinee.

However, the job for the solver is much simpler than for fixed-form assembly. The previous shadow test can be fed back into the solver as initial solution, which then needs to search only for a solution close to it in a subset of the same feasible space.

Numerous simulation studies and applications have shown item-selection times in milliseconds.

Search for Next Shadow Test



Set of solutions without any constraints
Subsubset with next shadow test
Subset of solutions satisfying all constraints

Conclusions

The shadow-test approach allows us to do anything that is possible for automated assembly of fixed test forms but then in an adaptive format.

In fact, as shown in the next presentation, it can even produce the same test content in any possible format.

For example, just by temporarily “freezing” and “thawing” of the shadow test, it turns adaptive testing into linear-on-the fly testing, multistage testing with fixed subtests, multistage testing with an adaptive first subtest, et cetera.