Top Ten Secrets to Success with Optimization

Gerald G. Brown
Distinguished Professor of Operations Research
Naval Postgraduate School
Monterey, California 93943
19 August 2004

Mission Statement:
"We military Operations Researchers plan to fight and win our nation's wars."

The Department of Defense coordinates the activities of millions of people using assets worth trillions of dollars. This is important, complex work. Computer-based decision support is universal, and this has enabled optimization to continue to enhance legacy manual planning methods. The success of real-world applications of optimization turns out to depend on a handful of trade secrets that are essential, but do not appear in textbooks. This short summary reveals ten of these secrets and motivates them by example.

#1: Document (Think) before you compute.

We have had an astonishing number of opportunities to unwind problems with optimization models that are implemented, and behaving badly, but that are not yet documented at all.

Documentation must include these three critical components:

- A non-mathematical executive summary,
- A mathematical formulation, and
- A verbal description of the formulation.

1.1) A ***non-mathematical executive summary*** must answer the following five questions, preferably in this order.

1) What is the problem?
2) Why is this problem important?
3) How will the problem be solved without your help?
4) What are you doing to solve this problem?
5) How will we know when you have succeeded?

Express your executive summary in your executive sponsor's language, rather than in technical jargon. If you have trouble writing such a summary in less than five pages, this is a sure sign that you are not ready to proceed.

- Have a non-analyst read your executive summary to you, out loud,

- Ask this reader to explain your executive summary to you,
- Listen well, and
- Revise and repeat.

1.2) A ***mathematical formulation*** should include, in this order.

- Index use (define problem dimensions),
- Given data (and units),
- Decision Variables (and units),
- Objectives (and constraints), and (perhaps)
- A dual formulation.

Remember to define terms before using them. To distinguish inputs from outputs, adopt a convention such as lowercase for indexes and data, and UPPERCASE for DECISION VARIABLES.

1.3 A **v*erbal description of the formulation*** explains in plain English, and in your executive sponsor's language, what each decision variable, objective, and constraint adds to the mathematical model. Here you have a chance to define what the mathematics means as well as why each feature appears in your model. Avoid literally translating mathematics into English. For instance, avoid saying "the sum of production X over item subscripts i must be no more than 50 for each time subscript t," and instead say "the total production of all items must not exceed 50 in any year." Do state and justify any simplifying assumptions (some examples include "Our planning time fidelity is monthly, with a ten-year planning horizon." "We allow fractional issue quantities of these large volumes." "We have assumed demand is constant, and not random.")

#2: Bound all Decisions

Bounds restrict the domain of every decision. There is no such thing as an unbounded variable in our real, OR analyst's world. Establishing bounds for each decision variable is a trivial concept, too often ignored. True, any reasonable optimization solver will do this for you automatically, but the solver can't tell you that its analysis is based on bogus or missing data in your model. If you manually apply simple ratio tests (e.g., "If I had all the steel GOOGLE says the world produced this year, how many tanks could I build?") and get ridiculous answers (e.g., "2.1 M1A2 tanks," or "10 billion tanks"), you have discovered an error in your data or description of the manner in which tank production consumes steel.

Remember all the formal "neighborhood" assumptions underlying your optimization method? Taylor's Theorem makes any continuous function appear linear, if you bound things tightly enough. All your costs and technology probably exhibit nonlinear effects across widely-varying magnitudes, but maybe not over a small enough neighborhood --- the domain you are planning for.

Enumeration of models with integer variables can be made much easier if the bounds on the integer variables are as tight as possible. This is worth working on before you start trying to solve large models. If the tightest bounds you can state permit a "large" integer domain, relax the integrality requirement and round the continuous result to the nearest integer. The inaccuracy rounding inflicts will be no worse that one over the final value of the variable.

Bounding all your decision variables pays an unexpected bonus. Pull out your favorite optimization textbook and look at the basic theorems you may have sweated over in class. Notice how much mathematical lawyering becomes superfluous when you rule out the unbounded case. Voila.

#3 Expect any Constraint to Become an Objective, and Vice Versa

Military planning models are notorious for multiple, conflicting objectives.

Get familiar with a "weighted average objective," and what it really means. E.g., if you maximize "2-times-apples plus 13-times-hand grenades" you have introduced a basic assumption (often hidden or overlooked) about the relative worth of apples and hand grenades.

Learn about "hierarchical (i.e., lexicographic) objectives," and how to coerce off-the-shelf optimization software into following your hierarchy. For instance, you might maximize the highest-priority objective, and then add a constraint on this objective to maintain this performance in all subsequent solutions. Repeat with each lower-priority objective until these successive restrictions have addressed your entire hierarchy, or your model is so over-constrained that further restriction would be pointless. With an algebraic modeling language, you can automate all of this as a single model excursion.

You can see that there is a continuum (sic) between weighted average objectives and hierarchical ones. Make sure to use your solver, rather than your model, to make this asymptotic transition from finite weights to the infinite weights required to make a weighted average objective function render lexicographic results. Floating point numerical errors increase in direct proportion to the relative magnitude of the terms in your additive weighted-average objective. You may be able to express such an objective, but the computer won't see what you intend.

Expert guidance from senior officers and executives frequently filters down to modelers as constraints (i.e., orders). In our experience, constraints deriving from literal interpretation of such guidance inevitably lead to an infeasible planning model. Nobody likes that.

Discovering what can be done changes your concept of what should be done. This leads you to "aspiration constraints," where you determine, for instance, just how much of something you can maximize in isolation, and then write a constraint saying ("I'll settle

for 90 percent of this isolated maximum.") Work with your senior sponsors using these simple methods, and you can guide them to give you better advice. (Remember, as OR analysts we may think our job is to give advice, but it's really to help the sponsor make the right decision.)

A huge literature advises how to deal with multiple objectives, but the simple ideas here usually work best.

#4 Classical Program Sensitivity Analysis is Bunk

Blind application of dual values, right-hand side ranging, and other neat textbook tricks actually offer very little useful advice on how the solution responds as the inputs *all* change. Even for those few models that are continuously linear, classical textbook sensitivity analysis is rarely useful. In fact, some of the best off-the-shelf mathematical modeling languages and solvers do not support such analysis. We professors love to teach this stuff, and we'll keep teaching it because it conveys lessons on the foundations of our optimization methods, and how to interpret the quantitative (how much to do) and qualitative (what to do) influence of restrictions and relaxations.

Just plan on solving a lot of model excursions, and don't be afraid that these will take weeks to complete. In the past 15 years, improvements in linear program (LP) solvers and in particular in integer-linear program (ILP) solvers and their controls have improved performance by a factor of at least 10,000 *in addition to the much faster speeds of newer computers*. Many in our profession still recall overnight batch processing of MPS tapes, and this is not a fond memory. Get over it. All you need today is a reasonably endowed desktop or laptop computer. For almost any modeling engagement, we can expect to rig up an optimization that lets us express a question and get an answer while our sponsor still remembers that question.

In our world, it's more important to seek "scenario- (i.e., warplan-) robust" solutions than to worry about individual parameter changes. Always seek the current defense planning guidance as filtered and tended by your sponsor. These war plans are the currency of our military planning realm. Try to find some way to express your solution(s) to respond directly to this guidance and the requirements it has justified.

#5 Bound your Dual Variables

A conventional linear program constraint has a dual variable that is unbounded and can be interpreted "this is how much it would be worth to relax this constraint by one unit." Please refer to #2, above.

An *elastic* linear program constraint uses a linear penalty per unit of violation. Allowing this constraint to be violated at some finite penalty cost per unit violation bounds its dual variable (i.e., 'this is the most it is worth to me to satisfy this constraint, otherwise violate

it, pay this penalty, and I'll deal with the consequences." There is no such thing as an infinitely-valuable constraint. Decision makers get paid to deal with infeasibilities, and cannot rule them out in the real world.

When you convince your sponsor to work with you to state each constraint with a well thought out penalty for its violation, you have enormously enhanced your control and understanding of your decision support model.

Another surprise bonus appears with elastic constraints: integer linear programming is a lot easier to deal with when you know a priori that every candidate integer solution in an enumeration is, by definition, admissible (i.e., satisfies the constraints, albeit perhaps with some penalties.) Further, if you set your elastic penalties carefully, you will be rewarded with remarkable improvement in linear-integer solution quality and solver responsiveness.

#6 Pay Attention to the Dual

If you have a linear program, or can relax to one, state its dual. If you can't write an abstract of what this dual means, if you can't interpret your dual at all, or if your dual is nonsense (e.g., unbounded or infeasible), your primal problem is ridiculous. OK, that's strong language: Amend this to read "your primal problem needs more attention before you're ready to use it."

Consider this example of a simple maximum flow model. You have a source node, a destination node, and a capacitated directed network through which you wish to push the maximum flow volume from source to destination. Write this primal linear program and solve it. Now, recover the dual solution. Admire these dual values, and note that each arc on a minimum cut is distinguished by two incident dual values that differ. *If you want to attack this maximum flow network, and can cut these arcs, you have decapitated it*.

It's a short trip from here to resource-constrained network interdiction and the most cutting-edge, two-sided, attacker-defender models.

The dual is the keystone for all this.

#7 Model Robustly

Make sure your model considers alternate future scenarios, and renders a robust solution.

In the military, we plan for what is possible, not what is likely, so we seldom employ random variables to represent the likelihood of each alternate future. We do use simulation to make quantitative (perhaps random) changes to data elements, but we rarely randomly sample qualitative future changes.

There may be many theater war plans, but you normally only get one chance per year to ask for what you need to get ready for all of them.

Pay attention to the current defense planning guidance, and try your best to address the resulting requirements for your sponsor with your model.

Suppose our guidance is to fight and win one engagement while suppressing another, and then turning to fight the other and winning it. If there are, say, 20 fashionable war plans on the shelf for such potential engagements, and we don't get to choose our favorites, we may have to plan for 20*19 permutations of engagement pairs.

Worst-case plans are expensive. So what? It's better to convey truthful insights to your sponsor than to delude ourselves with baseless optimism.

From the worst case, we can devolve to, for instance, meeting a maximum subset of scenario requirements, or maximizing some gauge of scenario fulfillment, or some-such. Whatever turns out to be advisable, do your best to document with exquisite clarity what assumptions have been made, and what compromises have been necessary, in contrast to the naked, overarching defense planning guidance. This is key: we all have to try to sing from the same sheet of music. Despite apocryphal tales of the demise of analysts bearing bad news, an OR analyst using diplomatic, but unambiguous language and careful analysis to deliver bad news is a hero. Witness that in our OR community, we know who these outspoken heroes are, and honor and protect them.

Start with the robust (worst) case, because that's what we all are duty-bound to worry about and defend against.

#8 Model Persistence

Optimization has a well-earned reputation for amplifying small changes in inputs to breathtaking changes in advice.

Decision support engagements typically require many model excursions, followed by analysis, followed by revisions and more model plays. When we have heavily invested in analyzing a *legacy* scenario, and make some trivial adjustment to attend to some minor planning flaw, the last thing we want is a *revision* that changes everything.

If your model is ignorant of its own prior advice, it is ignorant, and you can expect annoying turbulence and disruption when solving any revision of a legacy model. Any prescriptive model that suggests a plan, and, if used again, is ignorant of its own prior advice, is free to advise something completely, needlessly different, and lose the faith of its sponsor. This is silly.

Persistence means "continuing steadily in some course of action," and that is exactly what we do with long-term optimization-based decision support engagements. We have

to successively meld expert judgment of our sponsors with optimal advice from our models.

It is easy to add model features that limit needless revisions. To do this, you need to *state a legacy solution as an input*, and then add model features to retain attractive features or limit needless revisions of this legacy. These persistent features might include:

- Don't change this legacy resource consumption by more than 2 percent;
- Between this legacy solution and any revision, add (or delete, or change) no more than 3 of the binary options in this set;
- Don't change this unless you also change that; and so forth.

We give our students a handout showing how to state integer linear constraints expressing the ubiquitous logical relations required in decision support (for instance, for binary options A and B, "A only if B," "A and B, or neither," "A or B, but not both," "A or B, or both," etc.). We also show them how to state persistent guidance for revisions (because --- you knew this was coming --- this stuff rarely appears in textbooks).

#9 Spreadsheets (and Even Algebraic Modeling Languages) are Easy, Addictive, and Limiting

OK, we need a quick answer, we need database support for model development and cataloging solutions, and we need a graphical user interface that supports ad-hoc analysis and graphical output. This means we spend a small fortune developing a purpose-built graphical user interface, or just use our off-the-shelf office software suite.

Spreadsheets with embedded optimization solvers are inviting. After all, even your executive sponsors likely know how to bring up, or at least admire a spreadsheet, so you gain immediate acceptance by adopting a familiar "look and feel" standard. And having a spreadsheet solution means you can immediately catalog it and display it with all the tools you use every day in your integrated office software suite.

But spreadsheets support only 2-dimensional views (and pivot tables) of many-dimensional models, exhibiting what my colleague Art Geoffrion calls "dimensional arthritis:" You can support a many-dimensional model, but it isn't easy or natural.

We get many calls from spreadsheet users who wonder why their optimization results either take forever (literally), or turn out to be wrong. We ask "how much did you pay for the solver you used?" You get what you pay for. Spreadsheet solvers have been used many times with good success, but before you commit to one, check the credentials of the optimization software provider you will depend on. How will you get help if you have problems? Be careful, and consider spending a few thousand dollars per seat on a well-known, off-the-shelf, supported, documented, commercial-quality optimization package.

Modeling languages are crafted to accommodate multi-dimensional models, feature interface links to all contemporary database, spreadsheet, and presentation managers, and make great prototypes. But if a prototype works and gains acceptance, the prototypic implementation modeling language may not make a good decision support tool. Modeling languages isolate monolithic models from off-the-shelf commercial solvers. Large-scale, indirect solution methods are not supported well (for instance, column generation, or decomposition). If you're working on an important problem, why would you jettison 40 years of wisdom?

The transition from hasty prototype to production model generator and interface is not easy. But, in our experience the result always justifies the investment. Model generation and solution times can be reduced from hours to just seconds. This is the real deal. If you have ever wondered why so few published models end up being used in the real world, this essential missing step may be key.

#10 Heuristics Can Be Hazardous

A heuristic --- whether a simple thumb rule or some well-known local search method --- is so easy to explain and implement, we are tempted to use one in lieu of more formal methods. Heuristics may not require optimization software and may offer a first choice to assess a "common sense" solution, but heuristics should never be your only choice.

We can also develop bounds on the best solution possible, although this is not as much fun to do as building a solution-seeking method. Without some bound like this, our advice is of completely unknown quality. This quality certification is important: a bound on the best possible solution is just as important as the best solution you have.

A mathematical optimization model takes longer than a heuristic to develop, and perhaps to solve, but it can provide a bound. In fact, we develop models of relaxations of very hard problems merely to get the bounds they provide.

Lacking a trustworthy assessment of the quality of your advice, you are betting your reputation that nobody else is more scrupulous or just plain luckier than you are.

Publishing a bound with your solution is the right thing to do, but there is a risk. We have been told: "Hey, you're leaving money on the table." Well, maybe we are, and maybe not.

The *interval of uncertainty* is what we call the interval between the value of a solution and a bound on the value of the best possible solution. (There are various sources of these, such as integrality gap, decomposition gap, Lagrangean gap, duality gap, etc.) When you compare two alternate scenarios, you can be absolutely sure about the winner as long as the two intervals of uncertainty are disjoint, no matter how large each of these intervals is. Realizing this, you can work just hard enough to find a distinguishing difference, and no harder.

We have also been in a private-sector competition where our heuristic competitors wrote the sponsor and said "Brown et. al admit their solutions may not be right." Boy, they got us there, didn't they? But their heuristic, which they boast "gets better solutions the longer you run it," reminds us of the difference between "known unknowns" and "unknown unknowns." We can work with the former, and get nightmares from the latter. A solution with no bound? No thanks.

We have also been told (sigh, and have read in our literature) "This ILP is NP-hard, so we use a heuristic." Please. Even if (ahem) you prove your ILP is NP-hard, this only means it's as hard as a lot of other problems we are already solving. How much better is a heuristic with exponential run time than a bounded ILP enumeration controlled by several hundred documented tuning parameters representing many decades of research?

Acknowledgements

This derives from an original presentation at our MORS meeting in June, 2002, at Ft. Leavenworth, KS. Rob Dell has helped me hone these topics for some years now with our graduate students. Kirk Yost encouraged me to write this and made valuable suggestions for improvement. Most of all, I am grateful for so many students who have confronted real-world problems using the optimization tools we teach, and are direct enough to come back to my office and tell me "Sir, neither my textbooks nor my notes from our courses explain this." You were right. We fixed this with each of you, and learned a lot on the way. Thank you all.

This appears in PHALANX, Vol. 37, No. 4, pp. 12 ff, and is reprinted with permission of Military Operations Society, 2004.