On Decision Support for Deliberating with Constraints in Constrained Optimization Models

Steven O. Kimbrough
University of Pennsylvania
Philadelphia, PA, USA
kimbrough@wharton.upenn.edu

Ann Kuo
University of Pennsylvania
Philadelphia, PA, USA
aja@wharton.upenn.edu

Hoong Chuin LAU
Singapore Management University
Singapore
hclau@smu.edu.sg

David Harlan Wood
University of Delaware
Newark, DE, USA
wood@cis.udel.edu

ABSTRACT
This paper introduces the Deliberation Decision Support System (DDSS). The DDSS obtains heuristically (using a genetic algorithm) solutions of interest for constrained optimization models. This is illustrated, without loss of generality, by generalized assignment problems. The DDSS also provides users with graphical tools that support post-solution deliberation for constrained optimization models. The DDSS and this paper, as befits practical concerns, are focused on deliberation with respect to the constraints of the models being used.

Categories and Subject Descriptors
H.4.2 [Information Systems Applications]: Types of Systems—Decision support (e.g., MIS)

General Terms
Algorithms

Keywords
sensitivity analysis, deliberation support, constrained optimization, post-solution analysis, candle-lighting analysis

1. INTRODUCTION
Constraints in a COModel (constrained optimization model) serve to partition solutions for the model into two categories: feasible (satisfying the constraints) and infeasible (violating one or more constraints). Assuming that both categories are non-empty, this poses a number of interesting challenges for heuristic solvers of COModels. The first, most obvious challenge is the solution problem: How can we find a good, or even optimal, solution to the model? It is natural and appropriate that very much attention has been given to the solution problem for COModels, in both the traditional operations research (OR) and the heuristics communities, especially given the problem of how to handle constraints, or rather constraint violations, as a heuristic solver samples solution space in search of heuristically good results. Genetic operators, and in general heuristic procedures, do not respect feasibility. What to do? This is obviously a most important problem, to which a now very large literature is addressed (see [3] for an up-to-date bibliography).

There is a second problem, which is of great practical import and which has received much less attention, especially in the heuristics community. We call it the deliberation problem. It arises once a good solution is to hand, call it \(x^+\) with value \(z^+\), for a COModel: Should the best available solution be implemented exactly or should we reconsider the model? Are there profitable opportunities to acquire additional resources and thereby relax one or more constraints? On the other hand are there solutions available inferior to \(x^+\) in terms of \(z\), but which would consume substantially less in terms of valuable resources? And so on for other deliberations. (Related terms of art include: sensitivity analysis, post-solution analysis, post-evaluation analysis, post-optimality analysis, and candle-lighting analysis.)

Our thesis, or at least hypothesis, is that metaheuristics, particularly population-based varieties, may be used effectively to support deliberation (post-solution analysis, etc.) for difficult COModels [17]. (We are not, at present, including linear programming models, since these are reasonably well-supported for deliberation, courtesy of the simplex algorithm.) In the process of solving a COModel, population-based metaheuristics sample the solution space in an intelligent and biased fashion. Their bias seeks regions of better performance. These regions are typically on or near the boundary of the feasible region (otherwise the constraints are moot). But these regions are of primary interest for deliberation; deliberation in the context of COModels is primarily concerned with solutions near the feasible–infeasible boundary. Sampled solutions near the boundary, as generated by metaheuristics, can provide valuable input to post-
solution deliberation.

Given this hypothesis, two issues immediately arise. The first is Are there, typically, interesting solutions of this sort to be found? Call this the existence question. The second, assuming a positive answer to the first question, is What are the most effective means to discover these interesting solutions? Call this the effectiveness question. These questions have begun to be addressed [12, 13, 14].

We focus in this short paper on a third question, How should a Deliberation Decision Support System (DDSS) be designed to support post-solution deliberation? The next section sketches the concept of and motivation for post-solution deliberation. Following that, we describe and discuss a decision support system that affords post-solution deliberation with COModels. We then briefly describe how the data—what we call the solutions of interest—were generated for the decision support system.

2. POST-SOLUTION DELIBERATION

Post-solution analysis is what is done after a constrained optimization model has been formulated, a solution or evaluation procedure applied, and results therefrom obtained. At this stage of the modeling life-cycle a number of questions arise naturally, and for applications, most crucially. Post-solution analysis, according to Greenberg [5], “is [the] probing into the meaning of an optimal solution. This includes conventional questions of sensitivity, and it includes some additional analyses that are unconventional in the sense that they go beyond textbook definitions.” Post-solution analysis has long been recognized in the operations research (OR) and management science community as an important, valuable aspect of applied modeling.¹ (See [5, 6, 7, 8] for a comprehensive discussion from the classical exact solution, OR perspective.)

One of the important motivations for undertaking post-solution analysis is, as we have noted above, to support the deliberation problem, which considers, at least in principle, actions that might be taken to revise the model’s assumptions. These considerations are based on weighing solution results along with knowledge not directly reflected in the model. Classical OR (exactly optimal solution methods) for post-solution analysis is most developed for linear programming models. Although there is important work for integer programming models and for scheduling (see [4], [9], and [10] for reviews) the results tend to be very model type specific and of limited scope. Moreover, these methods do not generally apply when the primary solution method is a metaheuristic, as it often is and must be in practice.

Briefly, because of space limitations, we can frame post-solution analysis of COModels as being organized around three types of questions. With what-if? questions we ask about the consequences of changing the values of one or more parameters. Sensitivity analysis falls under this heading. Examples: What if constraint 7 is tightened by 5%? What will be the new optimal solution and objective value? Why? and why-not? questions are aimed at understanding, e.g., why job a, instead of job b, was assigned to a certain processor in the optimal solution. At least part of the an-

¹Post-solution analysis is also called post-optimality analysis, postoptimal analysis and candle-lighting analysis [16, 2, 17, 14]. A related concern is “model busting” which is addressed in [20] and used evolutionary computation.
swer lies in finding solutions in which job b is so assigned and then examining the costs and consequences of this, such as a particular constraint being violated because of b's heavier use of that resource. (See [6] for a nuanced discussion of why-questions in a classic OR setting.) Finally, what-does-it-take? questions set a goal, such as a higher value of z or freeing up a certain amount of constrained resources, and ask for good solutions that satisfy the goal. To anticipate an example: At optimality $z = 644$, but what does it take—to get a value of z of more than 650?

These are all questions of great practical import in the use of COModels and none of them can be addressed having only the optimal solution to hand. We explore in this paper how the necessary information may be obtained and effectively delivered to a decision maker. We begin, in the next section, with this latter issue.

3. THE DELIBERATION DSS

The Deliberation Decision Support System (DDSS) affords users setting up and conducting experimental runs of constrained optimization models (COModels). Presently, the DDSS supports only generalized assignment problems (GAPs); we plan extensions to other kinds of models in the near future. Also, the DDSS uses a form of genetic algorithm (GA) for heuristically solving the COModel to hand. (We will describe our GA below.) Other solution heuristics could, and should, be investigated; that is an important item for future research.

In addition, and more importantly, the DDSS collects solutions of interest (Sols) that appear during the experimental runs of a chosen COModel. The key point here is to observe that in the course of heuristically solving a CO-Model any population-based metaheuristic, and any GA in particular, will sample the solution space with what we can expect to be a sensible bias.

For the sake of post-solution deliberation, the solutions of interest may be feasible or infeasible. On the feasible side, we are especially interested in solutions that have high objective values and comparatively large slack values (unused resources) on one or more constraints. We call these solutions FoIs, or feasible solutions of interest. On the infeasible side we are interested in solutions that are close to being feasible and that have high objective values. We call these solutions IoIs, or infeasible solutions of interest. (We make these concepts more precise below.) The main goal of the present section is to describe and illustrate how, with the Deliberation DSS, these solutions of interest may be used to support post-solution deliberation with a COModel. We now focus, thus, on the DDSS.
3.1 DDSS User Interface

The Deliberation DSS (DDSS) provides a graphical user interface (GUI), which is divided into 3 components: (i) the experiment configuration and execution area, (ii) the constraint resource allocation area, and (iii) the area for viewing of the sampled solutions. One can save results from different experiment runs into CSV files and import the data as needed.

Figure 1 shows a full view of the system, with all three components of the GUI visible (as well as parameter values for the runs on display). The details of the solutions in the Sols (FoIs, feasibles of interest; IoIs, infeasibles of interest) are listed in the Feasible Heap and Infeasible Heap tables, respectively (southwest quadrant of the display). The bar charts (in the southeast quadrant) show the number of retained solutions (SoIs) which have slack on the corresponding resource constraint. For the problem on display in figure 1 (GAP12.1 from [1], a standard test problem), the FeasH (FoIs) bar chart indicates, for example, that relatively only a few of the FoIs discovered (< 300) have any slack on constraint 8, while all or nearly all of these solutions do have some slack on constraint 2. On the infeasible side, the InfeasH (IoIs) bar chart tells us that among the retained infeasible solutions of interest (IoIs) a large number (∼200) had slack on constraint 7, but only about 65 had slack on constraint 5.

The box plots present the distribution of slack on each resource for the offered sample solutions. The system offers multiple ways to view the sampled solutions. As displayed in figure 1, there are multiple feasible solutions which provide high objective values and the highest objective value one can achieve with the solutions found by the heuristic solver is 1449 based on the provided solutions, of which there are 5 distinct solutions on display. Notice that these 5 solutions have different slack characteristics. Some use more resources than others, or different patterns of resources. Further, solutions nearly as good, with objective values close to 1449, are also on display and these also have varying resource usage patterns.

This will normally be information of considerable value to a decision maker, who will be concerned with possible redeployment of resources for other purposes or with events that may destroy or in other ways make unavailable particular resources. In the case of the GAP, the resource constraints are on the processors, e.g., machines or people, and may easily become less available or may see opportunities for more profitable deployment, factors not represented in the original model. Besides the pool of feasible solutions (FoIs), the system also provides a collection of infeasible solutions (IoIs) and a similar interpretation applies to their box plots.

3.2 Discovering a Better Objective Value

It is easy to identify a potential solution which can improve the objective value from the current 1449 to 1451 (by simply providing additional 3 units of resource 7 and resource 10). The north-east quadrant of the GUI has widgets that allow decision makers to answer the what-if questions by modifying the amount of available resources and observing the impact such modification has on the sampled solution set.

With objective value 1449, five feasible solutions with slack sum ranging from 32 to 15 are readily available, see figure 2. In some cases, it may still be profitable even though one has to pay for the additional resources in order to achieve higher objective value. As mentioned previously, a sample solution can be easily identified in figure 3.

3.3 Maintaining High Slack for a Certain Resource without Additional Budget

In terms of the amount of slack on each resource, the system provides the information shown in table 1 with regard to the combination of achievable objective value and slack amount – given that no additional resources are attainable. The table shows the Pareto or efficient frontier among the FoIs, comparing objective value versus slack by (single) constraint. Thus, for example, in table 1 we see that with respect to constraint 10 (R10, resource 10) the efficient frontier (among all the FoIs encountered) consists of two solutions, one with objective value 1,444 and slack of 36 on constraint 10, and one with objective value 1,446 and slack of 27 on constraint 10.

3.4 Paying to Maintain High Slack and/or High Objective Value

Suppose it is desirable to conserve as much of resource 7 as possible (since 4 out of the 5 top feasible solutions are actually tight on resource 7). One could have at least 14 unit slacks on resource 7 at the cost of one unit objective value (1448). If maintaining the objective value no less than 1449 is required, the system provides us two other alternatives: one can either purchase additional resource 1 and resource 10, holding 16 units of slack on resource 7, or one can obtain additional resource 3 and resource 6, and hold 15 units of slack on resource 7 (figure 4).

Another interesting infeasible solution shows that while maintaining the same objective value, 1,449, the decision maker can actually achieve 33 units of total slack (including 7 units of slack for resource 3, which is tight for most of the solutions with equal objective value) by providing one extra unit of resource 8. See figure 5. From there, another extra unit of resource 1 can actually further push the total slack further, as demonstrated in figure 6.
slack up to 39 units without impacting the objective value; see figure 6. Figure 7 shows the resource changes which have been input via the Resource Management user interface component (northeast quadrant). One can see that resource 1 and resource 8 now both have the additional unit available. Providing the extra resources changes one or more infeasible solutions into one or more feasible solutions. The details about the additional 48 feasible solutions are now listed in the sampled feasible solution pool (figure 8).

3.5 Generalizing on the Example

We have briefly described features of the DDSS that support post-solution deliberation and analysis, and have done so in the context of a single (representative) example. Stepping back, abstracting a bit, the DDSS functionality supports deliberation with COModels by finding solutions (feasible and infeasible) of interest. The feasible solutions of interest, Fols, have high objective function values and, generally, larger amounts of slack. They are interesting because they are comparatively further from the feasible-infeasible boundary, set by the problem constraints. The infeasible solutions of interest, IoIs, have generally high objective function values and comparatively lower amounts of constraint violation. They are interesting because they are comparatively closer to the feasible-infeasible boundary.

A main aspect of post-solution deliberation in practice is to reconsider the constraint right-hand side values of the COModel.

4. PRIORITIZED SOLUTIONS

Now to the details of our approach to the technical question of how to obtain SoLS. It will help to have before us a representative COModel. We will use the generalized assignment problem (GAP), since it is what is currently supported in the DDSS.

An integer programming formulation for GAP is given in expressions (1)–(3) of Figure 9, where \( p_{ij} \) is the profit from assigning job \( j \) to processor \( i \), \( a_{ij} \) the resource required for processing job \( j \) by processor \( i \), and \( b_j \) is the capacity of processor \( i \). The decision variables \( x_{ij} \) are set to 1 if job \( j \) is assigned to processor \( i \), 0 otherwise. The constraints, including the integrality condition on the variables, state that each job is assigned to exactly one processor, and that the bounded capacities of the processors are not exceeded [11, 18]. The parameters of the model are the matrices \( P \) and \( A \), with elements \( p_{ij} \) and \( a_{ij} \), and the vector \( b \) with elements \( b_j \). Each inequality in expression (3) is said to represent a constraint (on the corresponding processor) and the \( b_j \)s are the right-hand-side (RHS) values.

In solving a GAP we find an (exactly or heuristically) optimal setting of the decision variables, \( x^+ \), with corresponding objective value \( z^+ = z(P, A, b)^+ \). Deliberation and post-solution analysis are about changes in solutions and objective values of the problem under modification of the parameters, \((P, A, b)\). It is not practicable to alter the parameters and resolve the model multiple times, given the scale—the number of reruns—necessary to do this. Our thought is to use population-based metaheuristics, and evolutionary computation particularly, to populate the Fols and IoIs as a by-product of solving the model. We shall now explain how we have done this. The next section illustrates with examples.

Evolutionary computation is a natural choice for the problem of populating the Fols. In a successful run, or series of runs, of a genetic algorithm (for example) we would expect (and do find repeatedly in practice) that the GA (genetic algorithm) will produce many feasible solutions with fitness values (objective function values, \( z \)) close to the best found, \( z^+ \). As a meliorizing population-based metaheuristic, a GA will tend to produce many solutions with similarly high fitness values (providing of course that they exist and can be found). It is just these good but non-optimal solutions that, we observe, constitute the Fols.
1. Determine: HashAttribute, ConditionAttribute.
2. Initialize: MaxHeapSize, CandidateSolutions.
3. Initialize Heap to MaxHeapSize elements with poor scores on ConditionAttribute.
4. Heap ←− UpdateHeap(Heap, CandidateSolutions, HashAttribute, ConditionAttribute).

Function: UpdateHeap(Heap, CandidateSolutions, HashAttribute, ConditionAttribute).

1. While (CandidateSolutions ≠ [])
   (a) Candidate ←− head(CandidateSolutions)
   (b) CandidateSolutions ←− tail(CandidateSolutions)
   (c) If (Candidate satisfies ConditionAttribute) and (Candidate /∈ Heap) and (HashAttribute of Candidate > HashAttribute of Extractmin(Heap)), then
      i. Deletemin(Heap)
      ii. Insert Candidate into Heap.
2. Return Heap.

Figure 10: Pseudocode: prioritized solutions.

What about infeasibles and the Iols? Here we have to worry that standard penalty function approaches to handling infeasible solutions will not very comprehensively explore the infeasible region(s) near the feasible-infeasible boundary(ies). In the extreme case, amounting to a ‘death penalty’ for infeasible solutions, there will be comparatively few solutions found and they will not be parents of subsequent exploration. This worry has received some empirical confirmation [14, 19]. For these kinds of reasons we chose to begin our explorations using a version of the feasible-infeasible 2-population (FI2Pop) GA [12, 15], which maintains two populations, one of feasible solutions and one of infeasible solutions. Feasibles are selected with respect to objective function values, infeasibles with respect to minimizing distance to feasibility, or degree of constraint violation. New solutions, however parented, are placed in the feasible or infeasible population according to their evaluations.

Given the choice of GA, in order to populate the Fols and Iols, we set up heaps, or priority queues, two for feasibles and two for infeasibles. See Figure 10 for the pseudocode of what we call our prioritized solutions algorithm. Each heap comes with a maximum size parameter, MaxHeapSize, which we set to 2000 solutions. We fix the problem to be solved, e.g., a particular GAP, and we conduct a number of replications, each beginning with a different randomized initialization. The heaps, however, are maintained throughout the run, and so at the conclusion they contain the best solutions found, by their criteria, over all the replications in the run. We emphasize that what goes into the heaps does not affect the search process of the GA. This method of collecting data (figure 10) is computationally efficient.

On the feasible side we have heaps FoI(Obj) and FoI(Slacks|MinObj). In FoI(Obj) we store feasible solutions, ranked by objective function value, limited to the best MaxHeapSize encountered. FoI(Slacks|MinObj) contains the best feasible solutions whose objective values equal or exceed MinObj (normally set at 97.5% of $z^*$), where the evaluation criterion is the sum of the slacks in the constraints. Recalling Figure 9 and constraints (3), the sum of the slacks...
for any given feasible solution is \( \sum_{i \in I} (b_i - \sum_{j \in J} a_{ij} x_{ij}) \).

On the infeasible side, we have heaps \( \text{IoI}(\text{SumV}) \) and \( \text{IoI}(\text{Obj}|\text{MaxDist}) \). \( \text{IoI}(\text{SumV}) \) contains the best infeasibles found as measured by the sum of constraint violations. Recalling Figure 9 and constraints (3), the sum of the constraint violations is \( \sum_{i \in I} \min \{ 0, (b_i - \sum_{j \in J} a_{ij} x_{ij}) \} \). (Only violated constraints count towards the sum of the violations.) These are the infeasibles that are closest to feasibility. \( \text{IoI}(\text{Obj}|\text{MaxDist}) \) contains the best infeasible solutions as measured by objective value, \( z \), provided their sum of constraint violations is less than or equal to \( \text{MaxDist} \), typically \( \leq 5 \). These are high objective value infeasible solutions that are near the feasible region.

5. REFERENCES

Figure 7: Extra Unit on Resource 1 & Resource 8 and Its Impact on

Figure 8: Expanded Feasible Solution Pool


