2010

Effective Heuristic Methods for Finding Non-Optimal Solutions of Interest in Constrained Optimization Models

Steven. O. Kimbrough
University of Pennsylvania

Ann Kuo
University of Pennsylvania

Hoong. C. Lau

Recommended Citation

Kimbrough, S. O., Kuo, A., & Lau, H. C. (2010). Effective Heuristic Methods for Finding Non-Optimal Solutions of Interest in Constrained Optimization Models. *GECCO '10 Proceedings of the 12th annual conference on Genetic and evolutionary computation,* 295-296. http://dx.doi.org/10.1145/1830483.1830538

Effective Heuristic Methods for Finding Non-Optimal Solutions of Interest in Constrained Optimization Models

Abstract

This paper introduces the SoI problem, that of finding *non-optimal solutions of interest* for constrained optimization models. SoI problems subsume finding FoIs (feasible solutions of interest), and IoIs (infeasible solutions of interest). In all cases, the interest addressed is post-solution analysis in one form or another. Post-solution analysis of a constrained optimization model occurs after the model has been solved and a good or optimal solution for it has been found. At this point, sensitivity analysis and other questions of import for decision making (discussed in the paper) come into play and for this purpose the SoIs can be of considerable value. The paper presents examples that demonstrate this and reports on a systematic approach, using evolutionary computation, for obtaining both FoIs and IoIs.

Disciplines

Computational Engineering | Other Computer Sciences

Effective Heuristic Methods for Finding Non-Optimal Solutions of Interest in Constrained Optimization Models

Steven O. Kimbrough Ann Kuo Hoong Chuin LAU

27 January 2010

Abstract

This paper introduces the SoI problem, that of finding *non-optimal solutions of interest* for constrained optimization models. SoI problems subsume finding FoIs (feasible solutions of interest), and IoIs (infeasible solutions of interest). In all cases, the interest addressed is post-solution analysis in one form or another. Post-solution analysis of a constrained optimization model occurs after the model has been solved and a good or optimal solution for it has been found. At this point, sensitivity analysis and other questions of import for decision making (discussed in the paper) come into play and for this purpose the SoIs can be of considerable value. The paper presents examples that demonstrate this and reports on a systematic approach, using evolutionary computation, for obtaining both FoIs and IoIs.

1 Introduction

A field of investigation may make progress in a number of ways. First, and most fundamentally, progress is realized by advances in solving or gaining better knowledge of the field's outstanding problems. Second, a field also advances when it acquires important new problems and begins to address them usefully. In this way, the field becomes broader in scope and increases its relevance to the wider world. Evolutionary computation heuristics have achieved much success in solving constrained optimization problems. They are widely used in practice and a vibrant research community continues to make substantial progress in techniques for using evolutionary computation to solve constrained optimization problems. This is progress of the first kind.

The contribution of the present paper is of the second kind. In the present paper we characterize a new, or at least under-investigated, problem pertaining to constrained optimization. We call it the *non-optimal solutions of interest*, or SoI, problem. This we divide into two related subproblems, the *non-optimal feasible solutions of interest*, or FoI, problem, and the *infeasible solutions of*

interest, or IoI, problem.[1] Further, we demonstrate how evolutionary computation may be used to address these problems, we identify a number of research issues, and we present initial, baseline results on these issues.

The subject may be framed as follows. First some terminology. Given a constrained optimization model (COModel), the *optimization problem* is the problem of finding an optimal solution to the model. An *exactly* optimal solution is one such that it satisfies the constraints of the model and there is no other solution that has a superior objective function value and that also satisfies the constraints. Oftentimes exact solution methods are not available or effective in finding (exactly) optimal solutions. Heuristics are then used to solve the problem. We say that a solution obtained by a heuristic method, e.g., by evolutionary computation, is *heuristically* optimal if it satisfies the constraints and there is no known feasible solution that is superior to it in objective function value. If a solution x is heuristically optimal we say it is the best known solution; we denote it by x^+ and its objective function value by z^+. Similarly, if a solution is exactly optimal we denote it by x^* and its objective value by z^*, as is standard in the literature.

The optimization problem is central to the subfield of constrained optimization with evolutionary computation (or more broadly, with metaheuristics). Moreover, a great deal of progress of the first kind has been made and is continuing to be made. This is a vibrant and quite progressive area, as evidenced by the hundreds of papers published each year investigating and describing use of evolutionary computation (and metaheuristics generally) to solve COModels.

The optimization problem, however, is not the only interesting and important problem pertaining to constrained optimization models. It may be most important, but it is not unique. We wish to discuss two other problems of considerable interest and import. They complement, and in no way conflict with, the optimization problem or its methods of solution.

The first of these new problems is the *feasibles of interest* or *FoI* problem. A COModel will (almost always) partition its solutions into two classes: the *feasibles*, which satisfy all constraints in the model, and the *infeasibles*, which each violate at least one constraint. Roughly speaking (we will be precise in the sequel) the *FOI*s are those feasible solutions that are high (assuming maximization) in their objective function values relative to z^+ and that consume fewer resources than x^+. The *FoI* problem, then, is the problem of finding the *FOI*s for a given COModel.

The second problem we introduce is the complement of the first. Roughly speaking (again, we will be precise in the sequel) the *infeasibles of interest* or *IOI*s are those infeasible solutions that are high (assuming maximization) in their objective function values relative to z^+ and that are close to being feasible. The *IoI* problem is the problem of finding the *IOI*s for a given COModel.

Both the FoI problem and the IoI problem, then, may be classed under the more general SoI problem, the problem of finding non-optimal solutions of

[1] Terminology: We shall also use SoI for *solution of interest*, plural, SoIs or SOI'S; similarly for FoI and IoI.

interest for a constrained optimization model. In what follows we will propose and investigate algorithms—which we call *prioritized solutions* algorithms—for solving the FOI and IoI problems. First, however, a word on why these are interesting and potentially important problems.

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 Greenberg (1993a), "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 and valuable aspect of applied modeling.[2] (See Greenberg (1993a,b,c, 1994) for a comprehensive discussion from the classical exact solution, OR perspective.)

One of the important motivations for undertaking post-solution analysis is to support what has been called 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. The deliberation problem

> ...arises once a good solution is to hand, call it $\mathbf{x}^+$ with value z^+, for a COModel [constrained optimization model]: *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* $\mathbf{x}^+$ *in terms of* z*, but which would consume substantially less in terms of valuable resources?* And so on for other deliberations. Kimbrough et al. (2009)

Classical OR (exactly optimal solutions) methods for post-solution analysis are most developed for linear programming models. Although there is important work for integer programming models and for scheduling (see Geoffrion and Nauss (1977), Greenberg (1998), and Hall and Posner (2004) 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.

[2] Post-solution analysis is also called *post-optimality analysis*, *postoptimal analysis* and *candle-lighting analysis* Kimbrough et al. (1993); Branley et al. (1997); Kimbrough and Wood (2008); Kimbrough et al. (2009). A related concern is "model busting" which is addressed in Miller (1998) and used evolutionary computation.

This brings us to the FoIs and IoIs. They are interesting and are defined as they are because if they can be effectively populated, the solutions in them can be used to support deliberation and post-solution analysis, as just described. They are principled heuristic responses to the need for post-solution analysis.

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 answer 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 Greenberg (1993b) 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—what do we need to do—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. What this paper is about is how the necessary information may be obtained.

These considerations put two fundamental questions into play. The first is Why are the FoIs and IoIs (as characterized above) interesting and useful to have? Call this the motivation question. The second is Given that we are interested in FoIs and IoIs, what are effective and comprehensive ways of finding them? Call this the technical question. We have already given a short answer to the motivation question: FoIs and IoIs can be used to support, and are what we need to support, post-solution analysis and deliberation with COModels (perhaps excluding linear programming models). We can use the FoIs and IoIs to answer valuable what-if?, why?, and what-does-it-take? questions. What follows in this paper begins to address both of the motivation and the technical questions.

3 Complexity of the Problems

The COModels that we are concerned with, e.g., integer programming models (linear or not), mix-integer programming models (linear or not), are all NP-hard as optimization problems, and in practice will often (but not always, this is not necessary, as we shall see shortly) be approached with heuristic solvers. One approach to obtaining FoIs and IoIs is to alter the COModel's parameters systematically in the neighborhood of the boundary and re-solve the model. To see the problem with this approach, consider only the right-hand side values of the constraints in a small model, one having just 5 constraints. Assume we are

Function: *RunFI2PopGA*(`MaxGenerations`, `FMutationRate`, `IMutationRate`, `FXoverRate`, `IXoverRate`, `FPopSize`, `IPopSize`, `FPop`, `IPop`).

1. `GenerationCount` $\longleftarrow$ 0.
2. While (`GenerationCount` $\leq$ `MaxGenerations`)
   (a) Increment `GenerationCount`.
   (b) `TempFeasible`, `TempInfeasible` $\longleftarrow$ []
   (c) If *Size*(`FPop`) > 0, then
      i. Create `FPopSize` progeny (new solutions) by applying mutation and crossover operators at rates `FMutationRate` and `FXoverRate`. Use `ObjVal` as the fitness value for each solution in `FPop`.
      ii. For each new solution call *EvaluateSolution*(`Solution`). If the solution is feasible, add it to `TempFeasible`, else add it to `TempInfeasible`.
   (d) If *Size*(`IPop`) > 0, then
      i. Create `IPopSize` progeny (new solutions) by applying mutation and crossover operators at rates `IMutationRate` and `IXoverRate`. Use `InfeasVal` as the fitness value for each solution in `IPop`.
      ii. For each new solution call *EvaluateSolution*(`Solution`). If the solution is feasible, add it to `TempFeasible`, else add it to `TempInfeasible`.
   (e) `FPop` $\longleftarrow$ `TempFPop`, `IPop` $\longleftarrow$ `TempIPop`.

End Function.

Figure 1: Pseudocode for the Basic FI2Pop GA.

interested in infeasibles that are within 5 units of violation on each constraint and feasibles that have slack of at most 5 on each constraint. This implies $(5 + 5 + 1)^5 = 161{,}051$ different models that would have to be solved. If we are interested in ± 10 units on each side of constraints, we get $21^5 = 4{,}084{,}101$ different models to solve. And things get worse if we start considering other parameters in the model. Even if the COModel can be solved very quickly by an exact optimization solver, these kinds of numbers are overwhelming.

Clearly, except for very small problems, it will not be computationally feasible to sweep out the FoIs and IoIs and re-solve systematically. We need to sample, unless we have a fast (polynomial) method of generating solutions in the FoIs and IoIs from a solution (or solutions) obtained by other means. In general, such methods are not available, or at least they are unknown. There are special cases (linear programming and changes of basis, and some work in mixed integer programming Greenberg (1998)) but they are model-specific and

are not responsive to a pre-defined feasible or infeasible region of interest (e.g., infeasible solutions within 5 units of the constraint boundary). Of course, no potentially useful method should be dismissed; these and similar methods merit investigation in the present context of deliberation support and the problem of populating the FoIs and IoIs. In consequence, our investigation has begun with a general-purpose technical approach and it is mainly on this that we report in the following sections.

$$\max_{x_{ij}} z(\mathbf{P}, \mathbf{A}, \mathbf{b}) = \sum_{i \in I} \sum_{j \in J} p_{ij} x_{ij} \quad (1)$$

subject to:

$$\sum_{i \in I} x_{ij} = 1 \quad \forall j \in J \quad (2)$$

$$\sum_{j \in J} a_{ij} x_{ij} \leq b_i \quad \forall_i \in I, x_{ij} \in \{0, 1\} \quad (3)$$

Figure 2: Generalized assignment problem (GAP) formulation

4 Prioritized Solutions

Now to the details of our approach to the technical question. It will help to have before us a representative COModel. We will use the generalized assignment problem (GAP), an NP-hard problem that is important in practice and that is prototypical of difficult optimization problems.

An integer programming formulation for GAP is given in expressions (1)–(3) of Figure 2, 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_i 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 Kellerer et al. (2004); Martello and Toth (1990). The parameters of the model are the matrices $\mathbf{P}$ and $\mathbf{A}$, with elements p_{ij} and a_{ij}, and the vector $\mathbf{b}$ with elements b_i. Each inequality in expression (3) is said to represent a constraint (on the corresponding processor) and the b_is are the *right-hand-side* (RHS) values.

In solving a GAP we find an (exactly or heuristically) optimal setting of the decision variables, $\mathbf{x}^+$, with corresponding objective value $z^+ = z(\mathbf{P}, \mathbf{A}, \mathbf{b})^+$. Deliberation and post-solution analysis are about solutions and objective values of the problem under modification of the parameters, $(\mathbf{P}, \mathbf{A}, \mathbf{b})$. As we have seen, it is not practicable to alter the parameters and resolve the model, given the scale necessary to do this. Our thought is to use population-based metaheuristics, and evolutionary computation particularly, to populate the FoIs and IoIs as a

1. Determine: `HashAttribute`, `ConditionAttribute`.
2. Initialize: `MaxHeapSize`, `CandidateSolutions`.
3. Initialize `Heap` to `MaxHeapSize` elements with poor scores on `ConditionAttribute`.
4. `Heap` $\longleftarrow$ *UpdateHeap*(`Heap`, `CandidateSolutions`, `HashAttribute`, `ConditionAttribute`).

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

1. While (`CandidateSolutions` $\neq$ [])
   (a) `Candidate` $\longleftarrow$ *head*(`CandidateSolutions`)
   (b) `CandidateSolutions` $\longleftarrow$ *tail*(`CandidateSolutions`)
   (c) If (`Candidate` satisfies `ConditionAttribute`) and (`Candidate` $\notin$ `Heap`) and (`HashAttribute` of `Candidate` $\succ$ `HashAttribute` of Extractmin(`Heap`))), then
      i. Deletemin(`Heap`)
      ii. Insert `Candidate` into `Heap`.
2. Return `Heap`.

End Function.

Figure 3: Pseudocode for basic prioritized solutions algorithm.

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 FoIs. 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 FoIs.

What about the infeasible side and the IoIs? 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 Kimbrough et al. (2009). For these kinds of reasons we chose to begin our explorations using a version of the feasible-infeasible 2-population (FI2Pop) GA Kimbrough et al. (2008, 2003), 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. Figure 1 presents pseudocode for our version of the FI2Pop GA.

Given the choice of GA, in order to populate the FoIs and IoIs, we set up heaps, or priority queues, two for feasibles and two for infeasibles. See Figure 3 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. In a single run, 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, and this method of collecting data (Figure 3) 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 2 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 `IoI(SumV)` and `IoI(Obj|MaxDist)`. `IoI(SumV)` contains the best infeasible solutions found as measured by the sum of constraint violations. Recalling Figure 2 and constraints (3), the sum of the constraint violations for any given infeasible solution 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. `IoI(Obj|MaxDist)` contains the best infeasible solutions as measured by objective value, z, *provided* their sum of constraint violations is less than or equal to `MaxDist`, typically = 5. These are high objective value infeasible solutions that are near the feasible region.

5 Using the SoI's

We have tested our approach by extensive running of our deliberation support system (the FI2Pop GA with prioritized solutions to collect FoIs and IoIs), under various settings, on the Beasley OR-Library GAP sets Beasley (2009), collecting FoIs and IoIs as described in the previous section. As a representative example of our results, we report here with regard to the FoIs and IoIs solutions found for the Beasley OR-Library GAP set 4 problem 2 (c530-2). The problem dimension of GAP4-2 is 5×30 (5 machines, 30 jobs) and its known optimal objective value is 644. A main purpose of this section is to illustrate (very briefly for lack of space) how having access to the SoIs may indeed be interesting and contribute to

improved decision making. (Data was generated with GA runs with a mutation rate of 0.09, crossover rate of 0.5, population size 250, 5000 generations, 20 replications or trials, and heap sizes of 1000.)

5.1 GAP4-2: Feasible solutions

On the feasible side, the deliberation system found not only one but two solutions with the known optimal objective value of 644. Tables 1 and 2 show the two optimal solutions found. The differences are shown in red. We note that (i) conventional solvers will typically find just one optimal solution and (ii) there is generally real value in knowing of more than one. For example, one optimal solution to the *model* may be preferable to another because the underlying problem has changed or has relevant aspects that are not captured in the model.

	0	1	2	3	4	5	6	7	8	9
0	-	3	3	5	1	2	1	4	1	4
1	2	3	2	1	4	4	5	2	2	5
2	3	4	5	3	5	3	1	4	1	5
3	2	-	-	-	-	-	-	-	-	-

Table 1: GAP4-2: Optimal solution #1

	0	1	2	3	4	5	6	7	8	9
0	-	3	3	5	1	2	1	3	1	4
1	2	3	2	1	4	5	5	2	4	5
2	3	4	5	3	4	2	1	4	1	5
3	2	-	-	-	-	-	-	-	-	-

Table 2: GAP4-2: Optimal solution #2

The GA also produced other feasible solutions which have near-optimal values. See Table 3. At objective value 643, the GA found eight solutions which spare more units of certain resources. To compare the slacks among the optimal / near optimal solutions, we display them in the resource usage mode. If a resource, say R4 (corresponding to the constraint whose RHS value is b_4), can be used in other more profitable ways, the decision maker may be particularly interested in the alternative solution number 3 (which spares 11 units of resource 4). Also, the sampling results show that we have fewer opportunities for redeployment on resources 3 and 5. Looking at Table 4, from the `FoL(Slacks|MinObj)` heap, we note that 629 is an objective value above 97.6% of optimality (644) and the heap contains three solutions whose sum of slacks is 36 or more. This is potentially a significant savings on resources from either of the optimal solutions and may well be a good bargain, depending of course on actual opportunities and prices.

	Obj.Val.	R1	R2	R3	R4	R5
0	644	2	1	1	2	0
0	644	2	5	0	1	1
1	643	5	4	1	3	0
2	643	0	2	1	0	0
3	643	0	4	1	11	0
4	643	2	1	1	3	0
5	643	0	1	0	7	0
6	643	5	4	0	1	5
7	643	2	1	0	4	0
8	643	2	5	1	4	1
9	642	0	4	3	12	1
10	642	9	0	0	0	2
11	641	0	8	0	1	3

Table 3: GAP4-2: FoIs with high objective values; from `FoI(Obj)`

Since the sampled feasible solutions from the heaps can be ranked according to either their objective values or the slack on a certain resource, it is easy to obtain the information with regard to what-if questions involving reductions in one or more resources.[3] For example, in both of our optimal solutions, constraint 1 has a slack of 2 (see Table 3). To look for alternative solution(s) with a slack of at least 5 on resource 1, we can easily identify that solutions #1 and #6 both have a slack on constraint 1 of 5 and objective values of 643, requiring a reduction of less than 0.002% from optimality. We can also get relevant information on certain why-questions. For example, task 25 is assigned to machines 3 and 2 in the two optimal solutions respectively (see Tables 1 and 2). Why not assign task 25 to machine 1?, alternatively What is the best solution if we assign task 25 to machine 1? Solution #4 listed above has an objective value of 643 with the task 25 being assigned to machine 1. Its detailed assignment is given in Table 5.

Finally, on the feasible side, note that in `FoI(Obj)`, Table 3, there is one solution, #8 with objective value 643, that has slack of at least 1 on every constraint (resource). Neither of the optimal solutions have this property. In a stochastic world a decision maker may well prefer #8 to either of the optimal solutions. The `FoI(Slacks|MinObj)` heap, Table 4, predictably has more solutions with this property, although it is interesting that very many of the solutions there do not.

5.2 Infeasible solutions

When considering the infeasible solutions, perhaps the most frequently asked questions are about how one can best utilize extra resources (available say by

[3] In the linear programming context, the analog of this is the examination of shadow prices on the constraints.

	Obj.Val.	Slack Sum	R1	R2	R3	R4	R5
0	644	6	2	1	1	2	0
0	644	9	2	5	0	1	1
1	629	37	0	28	3	5	1
2	629	36	0	8	10	17	1
3	629	36	20	5	0	4	7
4	628	39	15	15	3	5	1
5	627	38	0	21	10	4	3
6	626	42	0	8	16	17	1
7	626	40	5	28	6	0	1
8	626	40	15	8	10	4	3
9	626	40	15	8	3	5	9
10	626	39	15	9	3	11	1
11	626	39	20	8	7	4	0
12	626	39	0	8	9	22	0
13	625	42	20	15	6	0	1
14	625	41	0	1	3	29	8

Table 4: GAP4-2: FoIs with large sums of slacks; from `FoL(Slacks|MinObj)`

	0	1	2	3	4	5	6	7	8	9
0	-	3	3	5	1	2	3	4	1	4
1	2	4	2	1	4	5	5	2	2	5
2	3	4	3	3	5	1	1	4	1	5
3	2	-	-	-	-	-	-	-	-	-

Table 5: GAP4-2: Best found solution with task 25 assigned to processor 1

purchase) to improve the objective value. In the case that an extra resource is very hard to get, one may be more interested in finding out what possible improvement can be achieved with minimal additional resource. In the case that a given amount of extra budget is available, one may be more interested to know how to allocate the extra resources to achieve the new optimal objective value. By sampling the infeasible solutions, the deliberation system offers the flexibility to accommodate these two different deliberation focuses.

Option one is to search the infeasible solutions which have the shortest distance to the feasible boundary, that is the `IoI(SumV)` heap. Since such search focusing on solutions with minimal constraint violation, the sampled infeasible solutions tend to have fitness value -1 (one distance unit). See Table 8. A rich collection of infeasible solutions which are just one step away from the feasible boundary are typically found by our approach. Three infeasible solutions (at least) are one unit short of either resource 2 or resource 4. If the one extra unit of required resource is obtained, the objective value will increase to 648 for all three alternative solutions (listed below in resource usage mode). See Table 6.

	R1	R2	R3	R4	R5
1	0	4	-1	0	1
2	2	1	0	-1	0
3	5	4	0	-1	0

Table 6: GAP4-2 IoIs: Three nearly-feasible solutions with objective values of 648

A second option in searching infeasible solutions in the IoIs is to look for solutions with high objective values, provided they are not too far away from feasibility. These are collected in the `IoI(Obj|MaxDist)` heap. With a given constraint violation threshold (a certain number of units away from the feasibility boundary, 5 in what we report here), the system offers different alternatives which achieve higher objective values then the one-away infeasible solutions do. For example, the deliberation system found the solutions (displayed in resource usage mode) shown in Table 7. If the required extra units of resources are supplied, we will be able to improve the objective value from the original 644 to 650 or even higher. Thus, the table can be used to answer what-does-it-take? questions. Interestingly, although Table 7 is just a small portion of the `IoI(Obj|MaxDist)`s, we see that there is considerable heterogeneity in the options displayed. For example, a decision maker could get an objective value of 650 by finding 5 units of R2 or 5 units of R5 or (2 units of R1 and 2 of R4), and so on.

6 Populating the Heaps

We have identified four heaps of solutions, two on the feasible side and two on the infeasible, whose members are of particular interest for post-solution analysis and deliberation with COModels. Further, we have presented and

	Distance to Boundary×-1	Objective Value	R1	R2	R3	R4	R5
1	-4.1231	652	-2	4	-2	-3	0
2	-2.8284	651	-2	4	0	-2	1
3	-2.8284	651	-2	-2	0	5	2
4	-5	650	-0	4	0	1	-5
5	-5	650	2	-5	1	0	1
6	-2.8284	650	-2	4	0	-2	1
7	-3.4641	650	-2	4	0	-2	-2
8	-3.6056	650	-3	-2	0	1	0
9	-4.2426	650	-3	1	1	-3	0

Table 7: GAP4-2 IoIs: Ranked by objective value

	Distance to Boundary×-1	Objective Value	R1	R2	R3	R4	R5
1	-1	648	5	4	0	-1	0
2	-1	648	0	4	-1	0	1
3	-1	645	8	1	1	-1	0
4	-1	645	0	4	1	-1	5
5	-1	644	0	4	10	-1	0
6	-1	644	0	2	4	-1	0
7	-1	643	5	-1	0	0	0
8	-1	640	-1	9	0	9	0
9	-1	639	0	8	-1	10	1
10	-1	638	5	-1	0	11	3
11	-1	637	5	1	0	16	-1
12	-1	637	16	1	0	2	-1
13	-1	636	-1	7	4	1	0
14	-1	635	5	18	1	-1	1
15	-1	635	5	8	14	-1	1
16	-1	634	5	8	2	-1	7

Table 8: GAP4-2: IoIs with with minimum constraint violations; from `IoI(SumV)`

demonstrated one way of populating these heaps. This raises a number of researchable questions. Let us say that one of our heaps is *complete* if it has no missing solutions. That is, for example, the `FoI(Obj)` heap is complete if it finds the `MaxHeapSize` feasible solutions with the best objective values; the `IoI(Obj,MaxDist)` heap is complete if it finds the `MaxHeapSize` infeasible solutions with the best objective values, provided they are within `MaxDist` of feasibility. And so on. Research questions arising include:

1. How can we determine whether a given heap is complete or reasonably so?
2. What are the most efficient/effective ways of populating the heaps?
3. When are we better off obtaining presumably very incomplete heaps, which we use to identify areas of interest and which we follow up on with focused efforts at heap population, perhaps with local search near solutions of particular interest? And how might we do this best?
4. Are some solutions of interest typically harder to find than others?

With these questions in mind, we offer a benchmark method. In our prioritized solutions algorithm we keep track of when (which generation of which trial) a solution enters the heap. Each trial begins with a random start. We can thus expect that over multiple trials the heaps will move towards completion, since no found solution that is in the (ideally) complete heap ever leaves it once it is found and enters. This would indicate a falling off by trial in the number of new heap entries.

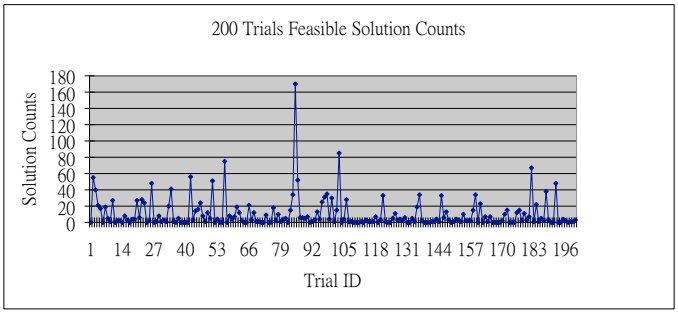

Figure 4: GAP5-1. Number of feasible solutions with superior objective values entering the `FoI(Obj)` heap by replication (trial). Heap size 2000, 200 trials.

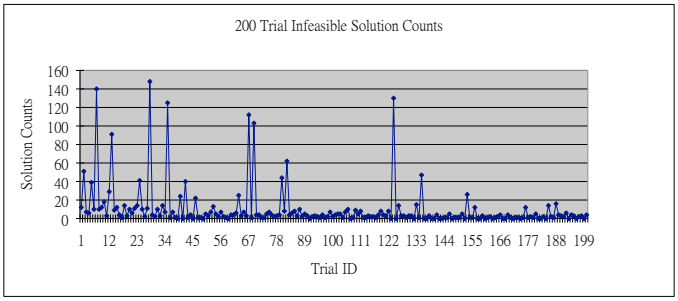

Figure 5: GAP5-1. Number of infeasible solutions with superior objective values entering the `IoI(Obj|MaxDist)` heap by replication (trial). `MaxDist`=5. Heap size 2000, 200 trials.

Figures 4 and 5 display the number of new entries in two of the heaps (see figure captions) by trial over 200 trials. There is an apparent high degree of completion on the infeasible heap (Figure 5), and perhaps a trend towards completion in the feasible heap (Figure 4).

A little modeling is helpful for insight. If, for example, every completed heap member was equally easy/hard to find and on average a trial would find 10 of the 1000 members, then the probability that a given member is not found in n independent trials is $(1 - 0.01)^n = 0.99^n$. If we want to have an expected completion score of say 90% then we need $1-0.99^n = 0.9$ or $n = \ln 0.1/\ln 0.99 \rightsquigarrow n > 229$. Alternatively we can think of this as follows. Suppose a solution of interest will only be found, say, in 1 in 100 trials and we want a 90% chance of finding it. Well, then we need to run at least 229 trials.

Table 9 is also helpful for understanding heap completion issues. (Data on the infeasible side are similar.) The (problematic) assumption that every SoI is equally easy or hard to find would suggest a Poisson distribution for the number of times the solutions in the final heap are encountered during the run (of multiple random-start trials). And indeed the results in Table 9 broadly

suggest Poisson-like distributions, although they are too thin in the tail for Poisson. Even so, we can gain some insight with a Poisson assumption. On the feasible side, for example, Table 9 would suggest a mean of 0.55 for a Poisson distribution, implying that the heap is about 1- 0.58 = 42% complete.

What we are reporting here with respect to GAP 5-1 is similar to what we have found with other problems in the GAP test suite Beasley (2009). Broadly speaking, there are (at least to us) surprisingly large numbers of SoIs, both feasible and infeasible. This implies important opportunities, currently not well exploited, of supporting post-solution deliberation. Of course, much more and much broader empirical work needs to be done. We can hope, however, that the methods here on display, of estimating the completion percentages of the heaps by internal means, will prove fruitful in the sequel and in practice. (Data was generated with GA runs with a mutation rate of 0.09, crossover rate of 0.7, population size 250, 5000 generations, 200 replications or trials, and heap sizes of 2000.)

7 Summary and Discussion

To summarize the contributions of this paper:

1. We identify the non-optimal solutions of interest (SoI) problem as important for the study of constrained optimization by metaheuristics, including particularly evolutionary computation. Further, we identify two related sub-problems: identifying FoIs and IoIs for support of deliberation and post-solution analysis.
2. We propose two collections of solutions to constitute the FoIs: the `FoI(Obj)`s and the `FoI(Slacks|MinObj)`s. Similarly, we propose two collections of solutions to constitute the IoIs: the `IoI(SumV)`s and the `IoI(Obj|MaxDist)`s.
3. We demonstrate the usefulness of the FoIs and the IoIs for post-solution analysis on a prototypical problem, GAP4-2. We assert, based on many test runs on different problems, that these results are representative.
4. We begin to investigate the question of how the four collections of solutions of interest may be most effectively populated. We find that multiple replications (trials) of the FI2Pop GA display a gradual, noisy leveling off in finding improved entries for the four heaps. The computational task, that is the number of trials needed, is typically larger than if we simply want the optimal solution (in expectation). This approach serves as a benchmark for other methods.

Regarding future research, besides exploring other kinds of models and gaining practical experience, we see a number of especially important agenda items.

1. The FI2Pop GA is a credible, and in our experience very good, approach to finding FoIs as well as IoIs. Other approaches that work as well or better need to be sought and investigated. (Our experience with penalty

function GAs suggests that they are not very good at finding IoIs, and this is hardly surprising.)

2. In our searches we used two heaps each for the feasibles and the infeasibles, one heap for each "objective" (fitness value, slack value). Perhaps a combined, weighted search would suffice or be preferable, i.e. to construct a heap that integrates the degree of infeasibility into the objective function. One way to implement this is via Guided Local Search Voudouris and Tsang (2003) in which different features of infeasibility are treated as penalty terms that augment the objective function.
3. There is no reason to require that the SoIs are found by a single method. Multiple heuristics might be employed. Hyperheuristics Burke et al. (2003), for example, offers a framework for selecting, combining, generating or adapting multiple heuristics (or components of such heuristics) to obtain the Sols.
4. Interpolation, neighborhood search, simulated annealing, et cetera among the FoIs and IoIs found by the GA or whatever method(s) are initially employed to find SoIs, may well be effective ways of focusing on regions of interest and providing the decision maker with a more complete inventory of the options available.

Clearly, very much additional research is called for on the SoI problem. Our aim in this paper has been to motivate and frame such prospective research, not to culminate it, an impossible task for any single paper. We hope, however, that others will become interested in investigating the non-optimal solutions of interest problem, and that soon progress on it will be seen as progress of the first kind.

References

Beasley, J. E. (Accessed July 27, 2009). OR-Library. World Wide Web. `http://people.brunel.ac.uk/~mastjjb/jeb/info.html`.

Branley, B., Fradin, R., Kimbrough, S. O., and Shafer, T. (January 1997). On heuristic mapping of decision surfaces for post-evaluation analysis. In Nunamaker, Jr., J. and Sprague, Jr., R. H., editors, *Proceedings of the Thirtieth Hawaii International Conference on System Sciences*. IEEE Computer Press.

Burke, E. K. et al. (2003). Hyper-heuristics: An emerging direction in modern search technology. In Glover, F. and Kochengerger, G., editors, *Handbook of Metaheuristics*, pages 457–474. Kluwer.

Geoffrion, A. M. and Nauss, R. (1977). Parametric and postoptimality analysis in integer linear programming. *Management Science*, 23(5):453–466.

Greenberg, H. J. (1993a). How to analyze the results of linear programs—Part 1: Preliminaries. *Interfaces*, 23(4):56–67.

Bin	Frequency
1	1503
2	273
3	103
4	53
5	25
6	15
7	14
8	5
9	4
10	1
11	2
12	2
More	0
Total	2000

Table 9: GAP5-1. Number of times the feasible solutions with superior objective values entering the `FoI(Obj)` heap are found. Heap size 2000, 100 trials. Frequency = number of solutions in the final heap that were found Bin number of times during the run.

Greenberg, H. J. (1993b). How to analyze the results of linear programs—Part 2: Price interpretation. *Interfaces*, 23(5):97–114.

Greenberg, H. J. (1993c). How to analyze the results of linear programs—Part 3: Infeasibility diagnosis. *Interfaces*, 23(6):120–139.

Greenberg, H. J. (1994). How to analyze the results of linear programs—Part 4: Forcing substructures. *Interfaces*, 24(1):121–130.

Greenberg, H. J. (1998). An annotated bibliography for post-solution analysis in mixed integer programming and combinatorial optimization. In Woodruff, D. L., editor, *Advances in Computational and Stochastic Optimization, Logic Programming, and Heuristic Search*, pages 97–148. Kluwer Academic Publishers.

Hall, N. G. and Posner, M. E. (2004). Sensitivity analysis for scheduling problems. *Journal of Scheduling*, 7:49–83.

Kellerer, H., Pferschy, U., and Pisinger, D. (2004). *Knapsack Problems*. Springer, Berlin, Germany.

Kimbrough, S. O., Koehler, G. J., Lu, M., and Wood, D. H. (2008). On a feasible–infeasible two–population (FI-2Pop) genetic algorithm for constrained optimization: Distance tracing and no free lunch. *European Journal of Operational Research*, 190(2):310–327. DOI information: 10.1016/j.ejor.2007.06.028.

Kimbrough, S. O., Kuo, A., Lau, H. C., Lindawati, and Wood, D. H. (2009). On using genetic algorithms to support post-solution deliberation in the generalized assignment problem. MIC 2009: The VIII METAHEURISTICS INTERNATIONAL CONFERENCE, conference CD. `http://opimstar.wharton.upenn.edu/~sok/sokpapers/2009/MIC/submitted/Deliberation.pdf`.

Kimbrough, S. O., Lu, M., Wood, D. H., and Wu, D. J. (2003). Exploring a two-population genetic algorithm. In Cantú-Paz, E. and et al., editors, *Genetic and Evolutionary Computation (GECCO 2003)*, volume 2723 of *LNCS: Lecture Notes in Computer Science*, pages 1148–1159. Springer-Verlag, Berlin, Germany.

Kimbrough, S. O., Oliver, J. R., and Pritchett, C. W. (May-June 1993). On post-evaluation analysis: Candle-lighting and surrogate models. *Interfaces*, 23(7):17–28.

Kimbrough, S. O. and Wood, D. H. (2008). A note on the deliberation problem for constrained optimization and how evolutionary computation may contribute to its solution. In *Proceedings of the Conference on Information Systems & Technology (CIST 2008)*. INFORMS. `http://www.rhsmith.umd.edu/doit/cist2008/`.

Martello, S. and Toth, P. (1990). *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, New York, NY.

Miller, J. H. (1998). Active nonlinear tests (ANTs) of complex simulation models. *Management Science*, 44(6):820–830.

Voudouris, C. and Tsang, E. (2003). Guided local search. In Glover, F. and Kochengerger, G., editors, *Handbook of Metaheuristics*, pages 185–218. Kluwer.

`$Id: deliberation-short-generic.tex 1275 2010-01-29 13:52:12Z sok $`