Proceedings of the 2008 Winter Simulation Conference
S. J. Mason, R. R. Hill, L. Mönch, O. Rose, T. Jefferson, J. W. Fowler eds.

AUTOMATING THE CONSTRAINING PROCESS

Joel J. Luna

Dynamics Research Corporation
10 Carter Drive
Chelmsford, MA 01824, USA

ABSTRACT

The typical approach to finding minimum levels of resources that still allow support and operational performance goals to be met for military aircraft is based on a manual trial-and-error method. This is because the function to be minimized (total manpower levels) is an input, and constraints (in terms of support and operational goals) are outputs. An approach with algorithms for automating this optimization, called constraining, is presented, based on dividing the range of total manpower into partitions and then searching for the lowest partition that still returns a manpower allocation that meets the performance goals. A tool, which was developed implementing this approach, is also discussed and results are presented. The conclusion is that while analysts are surprisingly good at finding minimum levels of resources, an automated approach produces acceptable results which are also reproducible and reduce analyst workload.

1 BACKGROUND

In evaluating logistical support for military aircraft (or other operational systems for that matter), one must determine levels of manpower, parts, support equipment, and facilities so that specific support and performance goals can be met under certain constraining conditions. Performance goals might include sortie generation rate (SGR – the daily number of sorties flown per day normalized by the number of allocated aircraft), and support goals might include direct manpower spaces per aircraft (DMSpA) and logistics footprint (amount of support materiel taken when deployed). Constraining conditions could include manpower utilization rate (not over 70%, to avoid designing in the expectation of high manpower utilization to meet goals) and overfrag (the number of additional sorties scheduled beyond the number desired in the performance goal). In addition to the performance and support goals, there is the overarching goal (stated or unstated) of affordability – that operations and support be performed as economically as possible (read minimum cost translating into minimum resource levels).

The focus of this paper is primarily on manpower levels, since optimal parts allocation can be obtained using a variety of tools (in this case, the Aircraft Sustainability Model by the LMI (Kline et al. 2001) which uses marginal analysis coupled with difference equations to determine optimal spares levels to achieve stated availability goals). In particular, minimum manpower levels (by shift) are desired which still result in meeting support and operational goals, versus use of per unit cost factors to determine manpower requirements. Because the relationship between manpower levels and performance goals is difficult to adequately represent analytically, a discrete-event simulation approach is used.

The discrete-event simulation model used in this case is LCOM (Logistics Composite Model), a large-scale stochastic model which allows the analyst to define any number of operational, maintenance and support activities as interconnected tasks, usually at the squadron level (ASC/ENMS 2004). Analysts typically create tens of thousands of tasks for thousands of removable/replaceable items. Since LCOM is not a requirements determination model per se, the minimum manpower levels are defined through a trial and error process known as the 'constraining' process. In this process, an analyst will set the manpower levels, run the model (typically for one iteration), examine outputs, and then make adjustments to the manpower levels, and repeat until selected goals (such as SGR) can just be met. Then the model is run at higher iterations, and the process repeated if the goal cannot be met.

There are several disadvantages to this approach. The first is the obvious intensive use of analyst time to conduct the trial and error approach. The second, and actually more compelling disadvantage, is the lack of reliability of results, i.e. the lack of the ability to duplicate the same results from one analyst to another. This is particularly relevant when comparing the results from an original data version (that is, reliability and maintainability data in terms of failure rates and maintenance times) to an updated data version. Differences can be ambiguous – are they due to differences in the data or differences in the analyst performing the constraining? For this reason, the need for an automated approach which could be duplicated became more urgent.

2 INTRODUCTION

This paper introduces the approach to automating the constraining process, or autoconstraining, beginning with ground rules for the development of the approach and the specific implementation and results to date. Before adopting this approach, a search of the literature was conducted to see what approaches existed to solve the following problem:

$$\min G(\mathbf{x}), \text{ where } x \in I$$
$$\text{subject to}$$
$$\mathbf{x} \geq \mathbf{x}_{\min}$$
$$E[\mathbf{H}(\mathbf{x},\psi)] - \boldsymbol{\varepsilon} \geq \mathbf{h}$$

where each value x of **x** is a level of manpower resources on a shift (not summed over shifts) and is an integer and greater than some minimum shift level value. The function G(**x**) is linear and deterministic (i.e. $\Sigma\mathbf{x}$), while the expectation of performance measures **H** (based on the input values of **x** and stochastic effects ψ) minus some sampling error term $\boldsymbol{\varepsilon}$ must be greater than or equal to corresponding constants in **h**. A value of h might be an SGR requirement, and a mean of SGR from simulation output (H) is compared to it, taking into account the uncertainty, to determine if the requirement is met or not for the values of **x**.

A review of current research on simulation optimization methods did not readily yield an applicable existing approach which emphasized uncertainty in the constraint conditions versus the objective function. Several approaches were considered, however, particularly stochastic Nested Partitions (Shi and Ólafsson 2000).

Meanwhile, a separate effort to build an optimization tool was conducted by the National Center for Supercomputing Applications (NCSA) with the USAF Aeronautical Systems Center (ASC). This tool, the ASC Optimizer (Boughton 2006, Boughton et al. 2006), employs a directed search methodology (alternately reducing step size and partitioning) from very large levels for each shift resource (i.e. 1000) down to the minimum levels which still result in meeting the constraints on performance goals. While improving considerably in runtimes from initial implementations, the Optimizer can in some cases be significantly outperformed by manual constraining in terms of total manpower levels. This may be due to the assumption of monotonicity, that is, that reducing manpower will reduce or at least not change output measures such as SGR. Experience has shown that this is not strictly true, and it could result in not searching in more promising regions. The potential effect of monotonicity on the Optimizer is under investigation.

In addition, Sandia National Laboratories also conducted an initial investigation, comparing two approaches, based on optimization work for a related simulation model (Watson 2006). The first approach was based on an iterative descent, taking advantage of manpower utilization statistics to reduce underutilized resources, while the second approach was based on the use of genetic algorithms. Sandia evaluated both methods with a simple LCOM dataset, but no further progress with more complex datasets has been reported.

3 PROBLEM DISCUSSION

The focus of the problem to be solved is not finding an optimal result in a feasible solution space, but finding a feasible solution in the solution space where the optimal result is most likely to be. With $G(\mathbf{x}) = \Sigma\mathbf{x}$, lower values of G will be more likely to have an *optimal* feasible solution but also more infeasible solutions, while higher values of G will be more likely to have feasible solutions but not the *optimal* feasible solution. Specifying lower and higher bounds for G would reduce the total solution space, eliminating infeasible solutions below the low bound and nonoptimal feasible solutions above the high bound. Values of **x** exist ($\mathbf{x}_{\text{low}}$) for which a $G_{\text{low}}(\mathbf{x}_{\text{low}})$ can be specified as a lower bound (i.e. no feasible solutions exist below it) and a set of values for $\mathbf{x}_{\text{high}}$ for which a $G_{\text{high}}(\mathbf{x}_{\text{high}})$ can be specified that has at least one feasible solution as a higher bound (i.e. feasible but nonoptimal solutions exist above it). While $\mathbf{x} = 0$ could specify a G_{low} value (i.e. 0), values of $x > 0$ can be determined for each x by setting all other x values to very high levels (in essence, ∞) and reducing a given x until performance measures are not met. It is very unlikely then that reducing all other x values (into a normal range) will produce feasible solutions. The resulting set of $\mathbf{x}_{\text{low}}$ values can be used to determine G_{low}. Executing LCOM with a sample data set and producing a MATRIX report, which provides the maximum levels of manpower used in the simulation run, can determine a G_{high} value. These values for **x** can be adjusted for sampling error and used to compute G_{high}. With G_{low} and G_{high}, the feasible solution space of **x** can be loosely bounded (loose in the sense that infeasible solutions will still likely exist between the bounds).

Because G is linear, the range from G_{low} to G_{high} can be easily partitioned and searched, where each partition has a corresponding G value. Sets of **x** corresponding to a given G value (i.e. all of the different values for **x** which total to G) can be used to determine whether constraints are met or not by running the simulation for the given allocation. Any **x** for which constraints are not met is rejected for being an infeasible solution. If a partition of G has at least one feasible solution and no lower partition of G has a feasible solution, then the partition of G is the optimal G value (G*). Because there can be a number of allocations of **x** which total to G*, there can be more than one **x** which is the optimal solution.

The number of ways that **x** can be allocated for any given partition of G can be quite large. Thus, some means of searching or sampling within a partition (i.e. for a given G value) is required since it is not practical to enumerate every possible combination of **x**. It is assumed that only the first feasible solution found for a given partition is sufficient to define that value of G as the new G_{high} (i.e. that the solution accounts for sampling error sufficiently to give high confidence that constraints will be met). The search then is directed to values of G below the new G_{high}. It is also assumed that a search or sample within a partition can be conducted such that if no feasible solutions are found, it is likely with some confidence that no feasible solutions exist in this partition. Furthermore, if one is able to conclude that it is likely no feasible solutions exist in a given partition, then it is likely that no feasible solutions exist in any lower partitions as well. This last assumption allows one to set the value of G for which there are no feasible solutions as the new G_{low}. Given these assumptions, a search can successively reduce the range between G_{low} and G_{high} (such as with a bisection method) until they meet, at which point G* is found.

4 AUTOCONSTRAINING ALGORITHMS

4.1 Basic Autoconstraining Algorithm

The basic autoconstraining algorithm can now be stated:

Algorithm: Autoconstrain

Step 0: Find $\mathbf{x}_{low}$ and $\mathbf{x}_{high}$ as described earlier so that lower and upper bounds for each x and for G (G_{low} and G_{high}) can be obtained. Set the initial value for the current G value being considered (G_{curr}) to G_{high}.

Step 1: Obtain a sample allocation of **x** (where $\Sigma\mathbf{x} = G_{curr}$) and obtain $E[\mathbf{H}(\mathbf{x},\psi)]$, where ψ represents the stochastic inputs from several stochastic samplings (such as iterations of a simulation). Evaluate whether $E[\mathbf{H}(\mathbf{x},\psi)] - \varepsilon \geq \mathbf{h}$. If so, **x** is a feasible solution and continue to **Step 3**, otherwise, go to **Step 2**.

Step 2: Repeat **Step 1** until some maximum number of samples has been taken. If no feasible solutions have been found, go to **Step 3**.

Step 3: If a feasible solution has been found, set G_{high} to G_{curr}. If no feasible solutions have been found, set G_{low} to G_{curr}. Set G_{curr} to the integer midpoint between G_{low} and G_{high}. If $G_{curr} = G_{low}$ or $G_{curr} = G_{high}$ (i.e. G_{low} and G_{high} are adjacent partitions), **Stop** and return $G^* = G_{high}$ and corresponding **x***. Otherwise, go to **Step 1**.

4.2 Obtaining Allocations of Shift Resources (x)

Random sampling is the primary method chosen to obtain allocations of **x**. The algorithm which obtains random samples of **x** for partition G_i which satisfies $\Sigma\mathbf{x} = G_i$ is as follows for all G_i:

Algorithm: Random Allocate

Step 0: Given initial $\mathbf{x}_{low}$, $\mathbf{x}_{high}$, G_{low} and G_{high} values, and G_i, set initial $\Delta = G_i - G_{low}$. Additionally, since each manpower type resource can have a minimum crew size on at least one shift for that resource, do the following: for each manpower type resource with a minimum crew size specified, randomly select one of its shifts and set x_{low} to the minimum crew size unless x_{low} is already greater than the minimum crew size. Reduce Δ by the amount x_{low} has been increased. Set $\mathbf{x}_{curr} = \mathbf{x}_{low}$. Also, randomly select the starting x in **x**.

Step 1: For each x in **x**, if x_{curr} is less than x_{high} and $\Delta > 0$, randomly draw a value X_i from between x_{curr} and the lesser of x_{high} and G_i. Decrement Δ by $(X_i - x_{curr})$. If $\Delta < 0$, add $|\Delta|$ to x_{curr} and set $\Delta = 0$.

Step 2: Repeat **Step 1** until $\Delta = 0$. Return $\mathbf{x}_{curr}$ as the sample allocation of **x**. **Stop**.

Because it is unlikely in successive random sampling for each x in **x** to produce allocations that are more or less the same proportionally between each x, a deterministic algorithm was added. For each x in **x**, a value of x is obtained which is the same proportion to x_{low} as G_i is to G_{low}. Only one proportional allocation of **x** was obtained for each G_i.

4.3 Constraining over Multiple Periods

It may be necessary to find multiple sets of **x*** corresponding to different periods of operation. For example, for military fighter aircraft, SGR goals can be set for surge, sustained surge, and sustained periods (say for 7, 23, and 60 days, respectively). It can be seen that the basic algorithm can be applied to each period to obtain **x*** for that period. Such an approach, however, must coordinate **x*** between the periods, since experience has shown that the periods cannot be considered independent of each other and that later periods depend on the earlier periods (this is especially true for surge and sustained surge, where the levels of manpower for sustained surge can be driven largely by the transition from surge to sustained surge). To coordinate the autoconstraining algorithm between the periods, the following algorithm was developed:

Algorithm: Period Autoconstrain

Step 0: For each period, perform **Step 0** of the **Autoconstrain** algorithm. Start with period 1 (e.g. surge).

Step 1: For a given period, perform the **Autoconstrain** algorithm with the following exception: before performing **Step 3** for the given period, perform **Step 1** (and **Step 2** if necessary) only for the subsequent periods (i.e. only G_{high} for each of the subsequent periods is being considered). If no feasible solutions are found for any of the subsequent periods, then it is as if no feasible solutions had been found for the given period.

Step 2: Set **x*** for the given period, and repeat **Step 1** for the next period until all periods have been processed (**Stop**).

The basic idea of the algorithm is to find **x*** for each period that still meets the goals of subsequent periods for G_{high} for that period, assuming that it would not for lower values of G if it did not meet them for G_{high}. This approach, however, does not guarantee an optimal answer across the periods – a better algorithm would adjust earlier periods in a search for an optimum, for example, if lower levels in period 2 might require higher levels in period 1 resulting in lower levels overall. The search for additional solutions somewhat mitigates this, as described below.

4.4 Finding Additional Solutions

At the point that **x*** are found, there are likely other **x** (where $\Sigma\mathbf{x}$ = G* or even $\Sigma\mathbf{x}$ near G*) that are feasible solutions. These are important to the analyst, since **x** are evaluated for face validity. Even **x*** may be rejected if the allocations are not considered 'realistic'. For this reason, a set of 'good' answers, rather than a 'best' answer, is what the analyst is wanting. The approach taken was to perform the **Period Autoconstrain** algorithm, and then repeat it for each feasible solution of **x** for period 1 starting from G* in the direction of G_{high} until some specified number of feasible solutions for **x** had been obtained, as described below:

Algorithm: Find More Feasible Solutions

Step 0: Perform the **Period Autoconstrain** algorithm. Set G_{curr} to G* for period 1.

Step 1: Perform **Step 1** of **Autoconstrain** for period 1, except: if **x** is a feasible solution, then go to this **Step 2**, otherwise go to this **Step 3**.

Step 2: Perform **Period Autoconstrain** starting with period 2.

Step 3: Repeat **Step 1** until:
a) a prespecified number of solutions has been obtained, at which point **Stop**;
b) some stopping criteria for this value of G have been met (corresponding to the efficiency of obtaining additional feasible solutions). Increment G to the next G partition, and repeat **Step 1**.

It can be seen that in cases with only one period, **Step 2** is not performed. It can also be observed that with higher period 1 G values, it is possible to obtain lower overall manpower levels as described earlier, so the lack of a true optimization approach across periods is somewhat mitigated when the analyst chooses to obtain more than one feasible solution.

5 AUTOCONSTRAINING TOOL

A prototype tool was developed (Luna 2007) to implement the algorithms described above in a set of DOS batch files, Awk scripts, and Simscript II.5 programs. The tool essentially acts as a wrapper to the LCOM program, running it for the evaluation step where $E[\mathbf{H}(\mathbf{x},\psi)]$ values are required. Specifics of how the algorithms were implemented are described below.

5.1 Simulation Iterations and Defining Performance Measures (H)

For a given **x**, each run of the LCOM model generates values for **H**. In a typical analysis with LCOM, 30 iterations are considered adequate for an estimate of the mean of **H**. (Currently no adjustment for standard error is made to conclude with high confidence that the true mean is at or above the requirement, but a value of ε based on the standard error could be derived). Since LCOM can be quite a sizeable model, 30 iterations for each sample can take quite a bit of time. A graduated approach was implemented which runs the LCOM model with iterations of 1, then 9 (for a total of 10), and then 20 (for a total of 30). After each set of iterations, $E[\mathbf{H}(\mathbf{x},\psi)] - \boldsymbol{\varepsilon} \geq \mathbf{h}$ is evaluated for different values of $\boldsymbol{\varepsilon}$. For initial iterations, we want to minimize the Type I error (rejecting a good solution), but we are willing to allow a higher Type II error (accepting a bad solution) since we are more likely at higher iterations to then reject the bad solution (but at a reduced efficiency). Earlier on, then, $\boldsymbol{\varepsilon}$ can be negative and large with respect to the standard error at 30 iterations (say by a factor of two or three), so that an allowable value for $E[\mathbf{H}(\mathbf{x},\psi)]$ could be below **h**. As the iterations increase, $\boldsymbol{\varepsilon}$ can be decreased as the standard error decreases. After the final set of iterations, $\boldsymbol{\varepsilon}$ should be positive and correspond to a one-tailed confidence level (for results provided in this paper, however, $\boldsymbol{\varepsilon}$ was set to zero for the purpose of comparison with manual results since this is the normal practice for LCOM analysts).

The analyst enters the following information to define **H**, **h** and $\boldsymbol{\varepsilon}$ for 1, 10, and 30 iterations: the name of the LCOM performance measure H (e.g. 'C15'); the period; the level (whether the measure is reported each day in the given period, for each period, or over all periods); resource (aircraft type for SGR or manpower resource type for manpower utilization); the number allowed to exceed the goal plus ε (for specifying percentiles, if desired); goal (the value h); and delta (the value for $-\varepsilon$, since delta is subtracted from the goal in the code). In the case of SGR, a value of H that meets or exceeds h is desired, while for utilization, a value of H that meets or is less than h is desired. The sign of the goal entry handles this. If the goal is positive (e.g. h = 3.0 for SGR), then the test used is $E[\mathbf{H}(\mathbf{x},\psi)] - \boldsymbol{\varepsilon} \geq \mathbf{h}$. If 'C15' = 2.8 and delta is 0.2 (so ε = -0.2), then the test is $2.8 + 0.2 \geq 3.0$, which is true (thus, this would be a feasible solution for

SGR). If the goal is negative (e.g. h = -70.0 for utilization), then the test used is E[$\mathbf{H}(\mathbf{x},\psi)$] ≤ | $\mathbf{h} + \boldsymbol{\varepsilon}$ |. If 'D2' = 80.0 and delta is 5.0 (so ε = -5.0), then the test is 80.0 ≤ | -70.0 + -5.0 |, which is false (thus, this would not be a feasible solution for manpower utilization).

The purpose of specifying values for **ε** for either 1 or 10 iterations is to provide some early culling of clearly bad answers prior to running all 30 iterations for greater efficiency. The analyst is not required to specify them, however, any earlier than at the 30 iteration point. Clearly, a systematic way of developing values for **ε** (such as tied to the factors of the sample standard error for the given number of iterations based on some prespecified desired confidence level) would be preferable to the ad hoc approach taken now.

5.2 Partitioning Total Manpower (G) – Step Size

Two approaches were taken to partition the range of total manpower into searchable chunks. The first (and default), is to specify a size for the partition, or step. For example, by specifying a step size of two, the partitions for a range of 50 to 100 for G will be {50, 52, 54, …100}. The second approach, developed primarily for focusing the search of G, is defined by a total number of samples desired and start and end values for G. For example, after a run of the autoconstraining tool using the default partitioning, the analyst may want to focus on the range of G from 61 to 75. The analyst can specify 100 samples to be taken between 61 and 75, which at 5 samples per partition would yield {61, 61.75, 62.5, 63.25, …75} as the partitions of G, which when rounded, become {61, 62, 63, 63, …75}.

5.3 Number of Samples per Step

For the results provided in this paper, a constant value of five samples per step was used (1 deterministic and 4 random allocations). This was considered adequate since the user can refine the step size as discussed above to provide more intensive sampling in a specified range and because adequate tool performance was obtained. A more thorough approach would address how to establish the likelihood with a specified confidence level that no feasible solutions exist in a given partition and that no feasible solutions exist in any lower partitions as well. Likely this will involve specifying some criteria (such as a pass/fail ratio) and determining the total number of combinations of **x** for a given value of G.

6 RESULTS

The autoconstraining tool (version A2.0) was used to constrain three different but related LCOM models (the three Joint Strike Fighter variants) that had already been manually constrained. In the case of the first variant, the tool obtained total manpower levels (G) that were between 5-10% lower than the manual results for three periods. For the second variant, the levels were much closer – the tool results were higher by one in the first period, the same in the second period, and lower by two for the third period. For the third variant, the tool results were higher in the first two periods but lower in the third period. Closer examination revealed that the manual results from the third variant did not satisfy all of the SGR and utilization goals (**h**), therefore, one would expect that the manual results would be higher (and thus commensurate with the tool results) if they met all of the goals. Runtimes on a Pentium III PC ranged from 2.5 to 8.5 hours for the tool from start to finding G* (i.e. the first **x***).

The tool was also distributed to two other analysts for use with different LCOM models. In both cases only one period was modeled. The first model was a Joint Strike Fighter variant modified for the Royal Australian Air Force. This model was constrained manually, using the autoconstraining tool, and also using the ASC Optimizer. The manual constrained results performed better than the ASC Optimizer, and the autoconstraining tool performed better than the manual constraining. Runtimes were also less for the autoconstraining tool. The second model was a UK modified carrier vehicle variant for the Joint Strike Fighter. This analysis added daily SGR and flight hour requirements, in addition to constraints on how x was allocated. In this case, the autoconstraining tool did not perform better than the manual constraining, although only by a few percent at best. This was particularly true for scenarios with very demanding goals. It also became evident that with very small increments of G (on the order of 1-5) that the random sampling approach was not adequate for generating enough samples. In this case, a more enumerative approach would work better.

7 CONCLUSIONS

An approach for automating the process of constraining resources to lowest possible levels that still meet performance goals was introduced and defined. A prototype tool was developed to implement this approach, and results so far indicate that it has potential for more widespread use by LCOM analysts. There are aspects of the approach that could use a more solid theoretical foundation, which likely would improve its capability. This work also shows that analysts are quite good at manual constraining, and that automated approaches will not likely provide significant levels of improvement (on the order of 25% or more). On the other hand, such automated approaches are repeatable and thus facilitates comparisons between results (such as in sensitivity analyses).

ACKNOWLEDGMENTS

This work was sponsored by the Joint Strike Fighter Program Office, Autonomic Logistics Modeling Simulation & Analysis, contract number 25298AFZ-01.

REFERENCES

Aeronautical Systems Command/Engineering Directorate (ASC/ENMS). 2004. ASC LCOM 2.6 User's Manual. ASC/ENMS, Wright-Patterson Air Force Base, Ohio.

Boughton, G. 2006. LCOM auto-constraining tool: The optimization wrapper. LCOM Summit presentation, May 2006, El Segundo, California.

Boughton, G. J., E. A. Linder, G. J. Dierker, F. J. Erdman, and M. T. Sipniewski. 2006. Legacy Air Force simulation model enhanced by optimizer. In 74th Military Operations Research Society Symposium Working Groups and Abstracts. Available via <www.mors.org/publications/abstracts/74th_abs.pdf> [accessed April 2, 2008].

Kline, R. C., S. P. Ford, T. Meeks, F. M. Slay, R. M. King. 2001. User's Manual for the Aircraft Sustainability Models (ASM, ISAAC, SSM): Version 7.0. Logistics Management Institute, McLean, Virginia. Available via <http://citrix1.lmi.org/asm/ASMManual.pdf> [accessed April 2, 2008].

Luna, J. 2007. LCOM auto-constraining tool: Alpha version 2.0 (A2.0). Presentation, Dynamics Research Corporation, Andover, Massachusetts.

Shi, L., and S. Ólafsson. 2000. Nested partitions method for stochastic optimization. *Methodology and Computing in Applied Probability*, 2:271-291.

Watson, J. 2006. An exploratory analysis of optimization methods for LCOM. LCOM Summit presentation, May 2006, El Segundo, California.

AUTHOR BIOGRAPHY

JOEL J. LUNA is a Staff Operations Research Analyst with Dynamics Research Corporation. He is the lead programmer for ASC LCOM, and also conducts analyses of logistics support systems for the Joint Strike Fighter Program Office in the areas of prognostics and health management (PHM) applications, reliability and maintainability (R&M) assessments, Verification and Validation (V&V), and special topics as they arise. His e-mail address is <jluna@drc.com>.