Integration of COTS software products ARENA & CPLEX for an inventory/logistics problem

Mayur Vamanana, Qian Wangb, Rajan Batta b,c,*, Robert J. Szczerbad

aAgilisys Consultant, 9 Country View Road, Malvern, PA 19355, USA
bDepartment of Industrial Engineering, University at Buffalo, 342 Bell Hall, Buffalo, NY 14260, USA
cCenter for Global Enterprise Management, University at Buffalo, Buffalo, NY 14260, USA
dAdvanced Technology Department, Lockheed Martin Systems Integration-Owego, 1801 State Rt. 17C Mid 0210, Owego, NY 13827, USA

Received 1 December 2000; received in revised form 1 March 2002

Abstract

This paper describes the integration of two commercially available, highly specialized, off-the-shelf software packages—CPLEX and ARENA. We present the need for such integration and the ways in which it differs from the more traditional methods of problem solving. The two products are described in detail followed by an example that requires this integration. A detailed account of the integration is also presented together with drawbacks of this approach.

Scope and purpose

The purpose of this paper is to demonstrate the mechanics of integrating two commonly used Operations Research software packages, CPLEX and ARENA. It then illustrates such integration by analyzing a sample supply chain application in the area of inventory/logistics. A discussion of the paper’s findings and potential drawbacks to the approach are provided.

Keywords: Software integration; Simulation and optimization; Supply chain applications
1. Motivation

Commercial-Off-The-Shelf (COTS) software products are products made available in the market to industries with specialized needs. These products are, in most cases, stand-alone applications. While the manufacturers of COTS products usually allow interfacing with programming languages (C, C++, BASIC, Java, etc.) and give their customers detailed instructions on how they can achieve this interface, they seldom document the possibility of integrating their COTS products with those of other manufacturers.

Due to potential savings in development cost, COTS integration has attracted the attention of software engineers. We cite some examples of recent work in this area. Yakimovich et al. [1] discuss the importance of estimating the costs of COTS integration before deciding whether or not to use a COTS solution and also to aid in the selection of the best available COTS products. They present a classification scheme of software architectures with respect to the integration of COTS tools. Stavridou [2] considers the issues to be examined in the use of COTS tools in critical systems. The paper emphasizes the fact that appropriate use of COTS tools can have high payoffs in terms of improvements in functionality, lifecycle costs, schedule and risks. Verma [3] talks about the increasing need to design new systems that make maximum use of COTS products to cope with trends of increasing system complexity with the addition of new and improved technologies and also provides insight into the necessary adaptation of the systems engineering supportability approach, life-cycle cost and provisioning approaches. Stottlemyer and Hassett [4] discuss the role of standards in COTS integration projects. They develop a process that identified standards that are applicable to the evaluation and integration of products and assess how those standards should be applied.

While doing a supply-chain optimization project we were faced with a logistics problem that needed both optimization tools as well as simulation tools. To attack this problem we considered using a COTS integration approach. It was clear after the project was initially defined that the components involved would be—an optimizer of some kind to fulfill customer demand optimally; a simulation tool by using which different scenarios and order fulfillment policies could be tested. This paper talks about integrating two specific COTS products—CPLEX (mathematical solver manufactured by ILOG Inc.) and ARENA (simulation software manufactured by Systems Modeling Corp.), and applies such a framework to solve a simple supply chain problem. Several factors were considered before the choice of working with CPLEX and ARENA was arrived at—including, but not limited to—functionality, performance, platforms, reliability, usability, documentation and costs. We now elaborate on some of these attributes:

- **Functionality** of the product is an obvious selection factor. To the extent the product requires tailoring or workarounds for the integration, this will tend to increase interdependencies with other parts of the system and increase risk on subsequent product releases.

- **Full performance considerations** of the COTS products must be tested before a selection is made. There are numerous examples of projects that failed because performance parameters were not adequately defined, resulting in the selection of COTS products that were not sufficiently robust for the application demands.

- **Reliability** of a COTS package is a factor that is best assessed by understanding the range of other client applications and installations, as well as the vendor’s track record in building reliable COTS products.
• *Usability and proper documentation* are extremely important factors as well. With cost of support being a major consideration in lieu of academic software licenses, vendors with proven track records must be given primary importance.

These two products and their respective vendors ranked higher than their competitors while considering these major factors.

The rest of the paper is organized as follows: Section 2 describes the selected COTS products; Section 3 describes a supply chain application that requires the integration of the two products; the *primary contribution* of this paper is detailed in Section 4—it describes the method of integration that we used in detail; Section 5 discusses a simulation experiment to illustrate this integration for a sample supply chain application. Section 6 spells out some limitations and alternatives to COTS integration in this context.

2. CPLEX and ARENA

In this section, we introduce the reader to the two COTS products in question, beginning with a brief introduction of the mathematical solver CPLEX and followed by a discussion on the simulation software ARENA.

2.1. CPLEX

CPLEX [5,6] is a tool developed for solving large-scale linear optimization problems—it can also be used for quadratic programs. As shown in Fig. 1, CPLEX includes an Interactive Base System and a Callable Library. The stand-alone Interactive Base System consists of an executable file called *cplex.exe* on Windows platforms and *cplex* on Unix platforms. The Callable Library is a software library of C routines that can be used to call CPLEX from the application. The available CPLEX optimizers are the standard primal simplex, dual simplex, network, barrier and mixed-integer optimizers.

Data for the problem can either be interactively entered in via the Interactive Base System with the specified syntax or read from a file. CPLEX supports both the industry-standard Mathematical Programming System (MPS) file format as well as its own LP file format. Files can also be read from the Callable Library. It is usually more efficient, though, to pass data directly into CPLEX while using the Callable Library.

CPLEX is available on personal computers with Windows ’98 or Windows NT and on UNIX workstations. It operates the same way and provides the same facilities on each of those platforms.

2.2. ARENA

ARENA is a computer simulation tool with a graphical user interface. Users can model complex systems using the available modules, blocks and elements available in the ARENA templates using
simple click-and-drop operations into the model window. Examples of systems that can be modeled are:

- manufacturing plants with people, machines and transport devices,
- bank teller windows, ATMs, and deposit boxes, and
- a computer network with servers, clients and networking capabilities.

ARENA uses SIMAN as the underlying simulation language. So, every time the user places a module in the model window and fills in the required fields, ARENA actually writes its own SIMAN code in the background. The user can view the SIMAN code optionally in a separate window.

Each ARENA model and template also has a corresponding Microsoft Visual Basic for Applications (VBA) project. The VBA code is saved in the ARENA file with the `. doe` extension, every time the ARENA project is saved. A VBA project consists of a collection of modules. These are classified by their type and are grouped in the VBA project window. The types of modules in the ARENA VBA project are:

1. **ARENA objects**: Each model has one ARENA object named `ThisDocument`. It is an object reference to the model itself. This is not changeable by the user, but can be extended by adding properties and methods, which can be referenced by code associated with other modules.
2. **Code modules**: These are user-defined functions, subroutines and variables that can be inserted into a project and referenced from other VBA modules.
3. **Class modules**: These are similar to code modules, except they can be thought of as objects, rather than simply a collection of functions and properties.

4. **Forms**: These are user-defined dialogs with controls. They are used to display information or query the user for input.

The *ThisDocument* object has a collection of standard events that can each contain VBA code. In essence, the user can use event-oriented programming to exactly define the sequence of actions that need to be performed at each pre-defined event in the simulation. The standard events in the *ThisDocument* object are:

- **DocumentOpen**—when the model window is opened
- **DocumentSave**—when the document is saved
- **RunBegin**—prior to checking the model
- **RunEnd**—after the last replication and SIMAN has been terminated
- **RunBeginSimulation**—prior to the first replication
- **RunBeginReplication**—prior to each replication
- **RunEndReplication**—after each replication
- **RunEndSimulation**—after the last replication
- **RunStep**—before each run step
- **RunPause**—when a simulation is paused
- **RunResume**—when a run is resumed
- **RunFastForward**—when a run is resumed via the FastForward command
- **RunRestart**—when a run is restarted via the StartOver command
- **RunBreak**—when a break occurs during a run
- **OnKeyStroke**—when any key is pressed
- **OnFileRead**—when a read occurs for a file
- **OnFileWrite**—when a write occurs to a file
- **OnFileClose**—when a file is about to be closed
- **OnClearStatistics**—when the statistics are cleared at warm-up-end time
- **UserContinuousEquations**—when a continuous state change occurs

The **VBA block** is a SIMAN block with a corresponding module in the Blocks panel of ARENA and is the most important component in the linking of the actual ARENA model and VBA. This module allows an entity to fire a VBA event during a simulation run. When a VBA module is placed in a model, a corresponding object appears in the VBA editor. The corresponding event in the *ThisDocument* object is the *VBA_Block_.#_Fire()* event. (# is the VBA block cookie number in the ARENA model corresponding to the particular VBA block.) This event is called when an entity in the model enters the VBA block. To illustrate, consider the blocks from an ARENA model as shown in Fig. 2.

In this example, the *VBA_Block_1_Fire* event is called each time an entity moves from the Enter module to the first VBA module; the *VBA_Block_2_Fire* event is called when each entity leaves the Process module.

A more complete discussion of this integration is contained in the recent Master’s thesis of Vamanan [7].
3. A Sample supply chain problem requiring COTS integration

We look at a small segment of a Supply Chain with geographically dispersed distribution centers being replenished with product stock from the respective company’s manufacturing facilities. These distribution centers are required to fulfill the requirements of customers as the demands for the various product-groups enter a central order management station. The function of the central order management station is to organize the distribution of these product groups to the customers from the various distribution centers to satisfy demand and also due dates. The way the orders are allotted to the distribution centers (DCs) can vary according to the order fulfillment policy in place. We assume that each product/DC combination has a specified min/max inventory level, i.e., it operates as a \((s, S)\) inventory system. The list of references at the end of the paper contains several citations (Bashyam and Fu [8]; Bogataj and Bogataj [9]; Buffa [10]; Carter and Monczka [11]; Chakravarty and Martin [12]) in the general area of inventory/logistics management that relates closely to our sample supply chain situation.

We now formalize our example. Consider a situation where an order comes from a customer with demand \(D = (d_1, d_2, \ldots, d_m)\). Without loss of generality, assume that the DCs are labeled \(1, 2, \ldots, N\) in increasing order of distance from this customer. Consider the vector \((s_{1j}, s_{2j}, \ldots, s_{mj})\), where \(s_{ij}\) is the stock level of product \(i\) in DC \(j\). Let \(n\) be the closest DC which can fulfill the whole order \(D\), i.e. \(s_{in} \geq d_i, \forall i\), but \(\forall j < n\), there exists an \(i'\) such that \(s_{i'j} < d_i\). If no such \(n\) exists, then set \(n = N\). Therefore, we just need to consider DCs \(1, 2, \ldots, n\) for this order. Assuming that the transportation costs from the distribution center are a stepwise function of the truckload shipped from that particular distribution center, the problem to minimize transportation cost and shortage cost for that specific order could be represented as the following Mixed-Integer Program:

**Notation**

\(a_i\) volume of product \(i\) per unit
\(T\) time left to fulfill this order
\(L_j\) minimum truckload to guarantee that the shipment is within time period \(T\) from DC \(j\)
\(c_{jk}\) transportation cost from DC \(j\) when the truckload falls in the \(k\)th interval \([l_{jk-1}, l_{jk})\)
\(p_i\) penalty cost for shortage per unit of product \(i\)
\(M\) a very large number

**Decision variables**

\(x_{ij}\) amount of unit \(i\) fulfilled by DC \(j\)
\(y_j\) truckload paid by DC \(j\)
\(u_j\) 1 if \(y_j > 0\), 0 otherwise
\(z_{jk}\) 1 if \(y_j\) falls in interval \([l_{jk-1}, l_{jk})\), 0 otherwise
\(f_i\) units of shortage of product \(i\)
Formulation

Minimize \( \sum_{j=1}^{n} \sum_{k} c_{jk} z_{jk} + \sum_{i=1}^{m} p_{i} f_{i} \), \hspace{1cm} (0)

Subject to the constraints:

\[ \sum_{j} x_{ij} + f_{i} = d_{i}, \quad \forall i = 1, \ldots, m, \] \hspace{1cm} (1)

\[ \sum_{i} a_{ij} x_{ij} \leq y_{j}, \quad \forall j = 1, \ldots, n, \] \hspace{1cm} (2)

\[ L_{j} u_{j} \leq y_{j} \leq M u_{j}, \quad \forall j = 1, \ldots, n, \] \hspace{1cm} (3)

\[ \sum_{k} l_{jk} z_{jk} \geq y_{j}, \quad \forall j = 1, \ldots, n, \] \hspace{1cm} (4)

\[ \sum_{k} z_{jk} = 1, \quad \forall j = 1, \ldots, n, \] \hspace{1cm} (5)

\[ 0 \leq x_{ij} \leq s_{ij}, \quad y_{j}, f_{i}, \geq 0, u_{j}, z_{jk} \text{ binary}, \quad \forall i = 1, \ldots, m, \quad \forall j = 1, \ldots, n, \quad \forall k. \] \hspace{1cm} (6)

An explanation of the formulation is as follows: The objective function (0), seeks to minimize the total cost, which includes the cost of transportation (i.e., shipping) and the cost of shortage. Constraint (1) enforces the fact that the shortage of product \( i \) is the difference of demand and supply. Constraint (2) guarantees that the truckload paid is greater than or equal to the actual truckload used. Constraint (3) ensures that once some products are shipped from DC \( j \), the minimum truckload that DC \( j \) should pay is at least \( L_{j} \) to guarantee that the shipment will reach to customer before the due date. (We assume that a larger truckload takes less time than a smaller truckload.) Constraint (4) is written so as to determine the interval that the truckload falls in, and constraint (5) ensures that the truckload falls into one and only one interval.

It is apparent, in such a case, that to study this order fulfillment policy in detail and possibly compare the costs of operation under this policy, we would need to simulate the events taking place in the supply chain segment, including the solving of the MIP every time an order from one of the customers comes in. To do so, we need to be able to call a mathematical solver, such as CPLEX, to solve this MIP so that the central order management station can allot the customer requests to the appropriate distribution center so as to minimize transportation and shortage costs for that order. Thus, we need to be able to call CPLEX from within ARENA, to solve an MIP every time an order arrives at the order management station.

The sequence of events that take place in the simulation is as follows:

1. Demands arrive from customers according to a probability distribution based on historical data.
2. Demand for each product group is entered into the system at the order management station.
3. CPLEX is called to solve the MIP.
4. Order assignment (i.e., the actual number of Product A to be shipped from distribution center 1 to that customer, distribution center 2 to customer, etc.) takes place. The actual breakup of the order into various parts is got from the CPLEX solution to the MIP.
5. The stock levels at the distribution centers are updated; transportation and shortage costs are calculated based on the solution at the various distribution centers.
6. Inventory holding costs at the distribution centers are calculated for the past period.
7. Stock levels are checked so that replenishment from the manufacturing facilities can take place if and when the particular stock falls below the minimum \((s)\) level. Replenishment takes place with a certain fixed lead-time and stops when the stock reaches the maximum \((S)\) level. We note that when replenishment does take place it occurs at a constant rate as opposed to a single delivery with an amount equal to \((S—\text{current stock level})\).

4. Integration

A major goal of this paper is to illustrate the integration of the COTS software CPLEX and ARENA. In this section, we provide the detailed steps needed to achieve this integration: All entities (demand arrivals) are made to pass through a VBA block as soon as they arrive into the system. As discussed earlier, the entry of an entity into a VBA block triggers a VBA Block_Fire event. CPLEX is called from within the code for this event. Before CPLEX is called, however, certain attributes of the entity need to be obtained from the model. The ThisDocument object of the model has a SIMAN code associated with it, which in turn has various defined properties and methods. The method used to access attributes of an entity is the EntityAttribute method. Using this method, which has two arguments, i.e., entity location (the current module that the entity is in) and attribute number, attributes that have been assigned to the entity are easily transferred to the VBA code. Values of certain global model variables also need to be obtained when the entity enters the VBA block. This is done by using the VariableValue method of the SIMAN object.

Now that the required entity-specific attribute values (such as demand for each of the product groups, cost of truck loads from the distribution centers) and the global model variables (such as the stock levels at the distribution centers) have been obtained from the ARENA model, the MIP can be formulated. This is when CPLEX functions need to be used.

CPLEX is written in the C programming language and all of its prototype functions are found in the library cplex.h. Since, in this application, CPLEX needs to be called from within VBA, which differs from C in several ways; certain modifications need to be made:

1. **Accessing the DLL**: While calling standard CPLEX functions in Visual Basic, the function has to be declared in a module by referring to the CPLEX DLL. For instance, consider the following CPLEX Callable Library routine in C: 
   ```c
   CPXsolution(CPXptr env, CPXLPptr lp, int *lpstat_p, double *objval_p, double *x, double *pi, double *slack, double *dj).
   ```
   The corresponding routine in the DLL has the same name and argument list. A similar function declaration needs to be created in Visual Basic as follows:
   ```vba
   Declare Function CPXsolution Lib "cplex66.dll" (ByVal envptr as Long, ByVal lpptr as Long, stat as Long, objout as Double, x as Any, piout as Any, slack as Any, djout as Any) as Long.
   ```
   The above statement identifies CPXsolution as an exported function of the DLL that Visual Basic can call. Subsequent calls from Visual Basic are made using the name CPXsolution.

2. **Handling C structures from within VBA**: While using the CPLEX Callable Library, provision must be made to handle pointers of two types of C structures that are specific to CPLEX. In the
above example, these are the pointers to the CPLEX environment and the pointer to the problem itself (the first two arguments in the function). Since pointers are simply 4 byte integers, the Long type can be used in Visual Basic, since these are integers with 4 bytes too. This will suffice, since it is never actually required to access these pointers in Visual Basic. They simply need to be obtained while calling CPLEX routines like CPXopenCPLEXdevelop and CPXcreateprob that require them as arguments.

3. **Handling C pointers from within VBA**: As in handling addresses of special C pointers, addresses of standard C types like integers, characters and double precision arrays also need to be handled. In VBA, integer, character and double arrays can be declared and passed by reference to the functions in the DLL. In the above example, arguments 5–8 illustrate this aspect. The same applies for pointers to single integer, character or double items.

Once these three aspects of interfacing with the CPLEX DLL have been handled, accessing it is similar to accessing it from within C. For each new CPLEX function that needs to be called, the appropriate VB type for each of its arguments must first be determined. Then a declaration like the one discussed above can be established before calling the function.

While the application uses routines of the CPLEX Callable Library, it must first open the CPLEX environment, and then create the problem object before it solves the problem. Before it exits, the application must also free the problem object and release the CPLEX environment. In the application under consideration, we follow the exact sequence of steps:

1. CPLEX requires a number of data structures to execute properly. These data structures must be initialized before any call to the CPLEX Callable Library can be made. This is done using the CPXopenCPLEXdevelop() function. This routine checks for a valid CPLEX license and returns a pointer to the CPLEX environment.
2. The problem object is instantiated by calling the CPXcreateprob() routine.
3. The problem object is populated by assembling arrays of data and then calling the CPXcopylp() routine to copy the data into the problem object. This is the most suitable way to populate the problem object rather than using sequential calls to the CPXnewcols(), CPXnewrows(), CPXaddcols(), CPXaddrrows(), CPXchgcoeflist(). This is because the problem we have is static and does not require dynamic addition of new rows and columns.
4. A call to the CPXmipopt() routine is made, once the problem object has been populated and the CPXcopyctype() routine has been called to copy variable types (which may be continuous, binary or integer in the MIP), to solve the MIP.
5. Calls to the CPXgetstat(), CPXgetmipobjval() routines are made to get the solution status and objective value of the problem, respectively, after the optimization.
6. CPXgetmipx() is called to obtain the values of the decision variables. Slack variables can also be obtained by calling the CPXgetmipslack() routine.
7. These values are stored and then the problem instance is freed using the CPXfreeprob() routine. The CPLEX environment is then released using the CPXcloseCPLEX() routine.

CPLEX is now available for future calls to solve the different MIPs associated with each demand arrival. It is important though, to store the solution values obtained using the CPXgetmipx() routine. This can be easily done either linking VBA with a database such as MS Access or even with MS
Table 1
Data on min/max levels and replenishment times for example

<table>
<thead>
<tr>
<th></th>
<th>Product A</th>
<th></th>
<th>Product B</th>
<th></th>
<th>Units/Load</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Minimum</td>
<td>Maximum</td>
<td>Minimum</td>
<td>Maximum</td>
<td></td>
</tr>
<tr>
<td>DC #1</td>
<td>50</td>
<td>125</td>
<td>40</td>
<td>100</td>
<td>30</td>
</tr>
<tr>
<td>DC #2</td>
<td>50</td>
<td>125</td>
<td>40</td>
<td>100</td>
<td>30</td>
</tr>
</tbody>
</table>

Excel. To make things simpler, the MS FlexGrid control built in Visual Basic is used to temporarily store the data made available from the CPLEX solution. The key is to be able to access this data to make modifications to the simulation model and this is easily achieved as each cell of the MS FlexGrid is easily accessed using its TextMatrix(Row, Column) property. Once this has been done, necessary modifications can be made via stock updating at the distribution centers, calculation of inventory holding cost for the past period and also the calculation of transportation and shortage costs based on the demand allotment.

4.1. An example

Consider a simple example to illustrate the working of this simulation model integrated with CPLEX. In Fig. 3, we have two distribution centers and two customers. Each of the distribution centers has different minimum and maximum levels of two products—A and B, respectively. Fig. 3 also illustrates the transportation (full-truck-load) costs from the customers to the distribution centers.

We assume demands to arrive from the customers with a certain probability distribution. Demands from Customer #1 arrive every 9 out of 20 weeks and are uniformly distributed between (30,40) units of Product A and (40,50) units of Product B. Demands from Customer #2 arrive every 12 out of 20 weeks and are uniformly distributed between (28,36) units of Product A and (34,46) units of Product B. Other details are shown in Table 1.
The replenishment lead-time is 1 week and replenishment occurs at the rate of one truckload per week. Other assumptions made in the simulation model are as follows:

- The full truckload capacity is assumed to be 30 units.
- Partial truckload costs are $\frac{1}{2} *$ full load cost for up to $\frac{1}{3}$ truckloads, and are $\frac{3}{4} *$ full-load cost for between $(1/3, 2/3)$ truckloads. For anything more than $2/3$ loads, the company has to pay the full-load cost. Partial truckloads less then a single truckload take a longer time (3 weeks) than full truckloads (2 weeks) to reach the customer.
- Both the customers have due dates that are 2 or 3 weeks away.
- A customer satisfaction index (CSI) is defined to measure the level of customer satisfaction. Whenever a customer demand is met on time the CSI is incremented by 1 unit. When the demand is not met on time that demand is lost and the CSI is decremented by 2 units.
- Shortage cost is $30/unit and inventory-holding cost is $0.3/unit/week.

When the model representing this system is run in ARENA, the first instance of the demand coming in has the following attributes:

1. Customer = 2,
2. Demand for product A = 30 units,
3. Demand for product B = 37 units,
4. Due date = 3 weeks.

The MIP in this instance is provided in the appendix.

Every time a demand arrives from a customer, a new MIP has to be formulated and then solved by CPLEX. Optimal values are obtained for the decision variables and the necessary changes are made to the model, as described earlier. This process continues till the end of each replication.

We mention here, that orders are sequentially fulfilled by the successive calls to CPLEX to solve the respective MIP for each incoming order. This is a greedy policy, in which each instance of the demand is fulfilled optimally, but there is no guarantee that this would be the overall optimal solution. An alternative way of dealing with orders would be to solve an MIP that provides for the batching of orders. This would allow for the DCs to hold inventory without fulfilling demand upon its arrival in anticipation of future demand. Detailed discussions on this strategy are contained in a recent Master’s thesis by Rana [13].

5. Simulation experiment

The first point to be considered while designing a simulation experiment is deciding upon whether the system being modeled is best analyzed via a terminating simulation or a steady-state simulation. See Law and Kelton [14] for an extended discussion.

A terminating simulation is one that has a natural event (E) that specifies the length of each run. A non-terminating simulation, on the other hand, is one for which there is no natural event E to specify the length of a run. It is important to determine the length of the warm up period from which point in time data can be collected for the steady-state parameter of interest. To decide on the length of warm up period, plots of relevant outputs must be examined. Another alternative would
be to examine the moving averages of parameters of interest. ARENA allows the user to perform such tasks in the built-in Output Analyzer (Kelton et al. [15] and Hossein [16]).

For the example discussed in Section 4.2, a warm up period of 100 weeks was decided upon after running the simulation for 5 replications of 3000 weeks each. Plots and moving averages of the total inventory on hand (IOH) were used to infer the length of the warm up. The steady-state parameters of interest are transportation cost, shortage cost and CSI incurred per order and the total inventory on hand. The factor we chose was the inventory level. It was set at three values:

1. Base −25%
2. Base (corresponding to \((s, S)\) levels discussed in the example)
3. Base +25%

The sum of the transportation and shortage cost goes down as the inventory level increases (Fig. 4), though the decrease that occurs from increasing the base inventory by 25% is very small. On the other hand, there is a significant increase in CSI value only when the base inventory is increased by 25% (Fig. 5). Thus the optimal inventory level would depend on the relative importance of CSI and transportation/shortage costs.

6. Summary and limitations & alternatives to COTS integration

The COTS integration approach detailed in this paper is efficient in certain cases (e.g., the example discussed above). In this case the integration approach performs better than the traditional heuristic based approach (greedy order fulfillment policy) of Order Fulfillment.
There are, however, limitations to the use of COTS integration. As discussed in Nemhauser and Wosley [17], solution time for a MIP using the Branch-and-Bound scheme (such as that used by CPLEX) usually increases exponentially with problem size (i.e., increase in the number of variables and/or number of constraints). Since the customer orders are assigned sequentially and CPLEX is called separately each time a demand enters the system, the number of customers does not increase the problem size. An increase in distribution centers does, however, greatly increase the number of elements in the objective function. Increasing the product groups does not have the same drastic effect on solution time because there are a greater number of variables added for each distribution center. We conclude that CPLEX could take too long to solve the related MIP for realistic problem sizes, thus making the simulation model too time intensive.

Alternatives to COTS integration include specialized mathematical algorithms to solve the Mixed Integer Program described in Section 3. CPLEX uses a linear programming based Branch-and-Bound algorithm to arrive at a solution after a series of linear programming relaxations. This may not be the most efficient way to solve the MIP. Solution of the MIP using Lagrangian Relaxation and Duality principles is often more efficient. Cutting-Plane algorithms may also be used in solving the MIP. To incorporate such features, one can write application’s code in a high-level programming language like C, and call ARENA from C. This approach, however, is likely to be far more time intensive. It is also less versatile than COTS integration, since changing the MIP formulation to account for other constraints, for example, would be much more easily handled using the COTS integration approach.

We close this paper by mentioning an alternative to deal effectively with unacceptable solution times for CPLEX to solve large MIPs. One can terminate CPLEX with a solution that is within a certain acceptable level to optimality; say 95% or 98% optimal solution. Even though the time to find the optimal solution increases exponentially with problem size, the time to find a solution that is close to optimal remains reasonable in most cases (Nemhauser and Wosley). It is also noted that CPLEX offers its own set of cutting plane features as well as tuning opportunities for its algorithm. These can all reduce the time needed to reach an acceptable solution. An implementation of COTS products with premature termination of CPLEX calls (and while using these advanced features of CPLEX) could thus be a viable alternative.

Acknowledgements

This work was supported by a grant from Lockheed Martin Systems Integration—Owego. This support is gratefully acknowledged. The authors would also like to thank three anonymous referees for their helpful comments on earlier versions of this paper.

Appendix.

The MIP formulation for the example in Section 4.1 is as follows:

\[
\text{Minimize} \quad 30 f_A + 30 f_B + 400 z_{10} + 600 z_{11} + 700 z_{12} + 800 z_{13} + 1000 z_{14} + 1100 z_{15} + 250 z_{20} + 375 z_{21} + 437.5 z_{22} + 500 z_{23} + 625 z_{24} + 687.5 z_{25}
\]
Subject To:
c1: \( f_A + x_{A1} + x_{A2} = 30 \)
c2: \( f_B + x_{B1} + x_{B2} = 37 \)
c3: \( 0.0333x_{A1} + 0.0333x_{B1} - y_1 \leq 0 \)
c4: \( 0.0333x_{A2} + 0.0333x_{B2} - y_2 \leq 0 \)
c5: \( 0.33u_1 - y_1 \leq 0 \)
c6: \( 0.33 u_2 - y_2 \leq 0 \)
c7: \( -10000 u_1 + y_1 \leq 0 \)
c8: \( -10000 u_2 + y_2 \leq 0 \)
c9: \( z_{10} + 1.33 z_{11} + 1.67 z_{12} + 2z_{13} + 2.33 z_{14} + 2.67 z_{15} - y_1 = 0 \)
c10: \( z_{20} + 1.33 z_{21} + 1.67 z_{22} + 2z_{23} + 2.33 z_{24} + 2.67 z_{25} - y_2 = 0 \)
c11: \( z_{10} + z_{11} + z_{12} + z_{13} + z_{14} + z_{15} = 1 \)
c12: \( z_{20} + z_{21} + z_{22} + z_{23} + z_{24} + z_{25} = 1 \)

Bounds
\( f_A \geq 0 \)
\( f_B \geq 0 \)
\( 0 \leq z_{10} \leq 1 \)
\( 0 \leq z_{11} \leq 1 \)
\( 0 \leq z_{12} \leq 1 \)
\( 0 \leq z_{13} \leq 1 \)
\( 0 \leq z_{14} \leq 1 \)
\( 0 \leq z_{15} \leq 1 \)
\( 0 \leq z_{20} \leq 1 \)
\( 0 \leq z_{21} \leq 1 \)
\( 0 \leq z_{22} \leq 1 \)
\( 0 \leq z_{23} \leq 1 \)
\( 0 \leq z_{24} \leq 1 \)
\( 0 \leq z_{25} \leq 1 \)
\( 0 \leq x_{A1} \leq 100 \)
\( 0 \leq x_{A2} \leq 100 \)
\( 0 \leq x_{B1} \leq 125 \)
\( 0 \leq x_{B2} \leq 125 \)
\( 0 \leq y_1 \leq 1 \)
\( 0 \leq y_2 \leq 1 \)

Binaries
\( z_{10} z_{11} z_{12} z_{13} z_{14} z_{15} z_{20} z_{21} z_{22} z_{23} z_{24} z_{25} y_1 y_2 \)

Generals
\( f_A f_B x_{A1} x_{A2} x_{B1} x_{B2} \)

End

References