Proceedings of the 2005 Winter Simulation Conference
M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, eds.

THE COMPUTATIONAL COMPLEXITY OF COMPONENT SELECTION IN SIMULATION REUSE

Robert G. Bartholet
David C. Brogan
Paul F. Reynolds, Jr.

Modeling and Simulation Technology Research Initiative
151 Engineer's Way, PO Box 400740
Department of Computer Science, University of Virginia
Charlottesville, VA 22904-4740, U.S.A.

ABSTRACT

Simulation composability has been much more difficult to realize than some initially imagined. We believe that success lies in explicit considerations for the adaptability of components. In this paper we show that the complexity of optimal component selection for adaptable components is NP-complete. However, our approach allows for the efficient adaptation of components to construct a complex simulation in the most flexible manner while allowing the greatest opportunity to meet all requirements, all the while reducing time and costs. We demonstrate that complexity can vary from polynomial, to NP, and even to exponential as a function of seemingly simple decisions made about the nature of dependencies among components. We generalize these results to show that regardless of the types or reasons for dependencies in component selection, just their mere existence makes this problem very difficult to solve optimally.

1 INTRODUCTION

Recently there has been a flurry of activity in the modeling and simulation (M&S) community regarding the complexity of simulation composability—the cost of building federations of simulations from component simulations. Constructing complex systems from off-the-shelf components has great appeal. It offers numerous benefits, not the least of which is potential order of magnitude savings in federation development time. Additionally if there are several components that meet the same set of requirements, then a broader set of design alternatives can be explored, providing greater flexibility to system designers and implementers. Dynamic Data Driven Application Systems (DDDAS) are a new class of applications in which simulations dynamically adapt to real-time, possibly streaming, data (Darema 2004). The dynamic composition of models has been listed as a key enabler for the DDDAS concept.

When it is difficult or impossible to construct a federation from preexisting immutable component simulations, we believe it is valuable to consider altering components so they will satisfy new requirements. Our COERCE research (Waziruddin, Brogan, and Reynolds 2003) reflects this perspective. For example we place emphasis on methods for rapidly adapting component simulations to meet new requirements as a substitute for any insistence that all desired end results be achievable without any adaptation of component simulations. The flexibility introduced by an efficient adaptability assumption has a cost however, in terms of the complexity of component selection. Exploration of that cost is the topic of this paper.

In this paper we show that under desirable relaxed assumptions, the complexity of optimal reuse of adaptable components is NP-complete. Rather than insisting that components are black boxes that must be used "as-is," we allow for adaptation, and even creation. (In this context, component creation can be considered a special case of adaptation, where creation is the adaptation of the null component.) The goal is to construct a complex federation in the most flexible manner while allowing the greatest opportunity to meet all requirements, all the while benefiting from the reduced time and costs of reusing components. We demonstrate that the complexity of composing alterable components can vary from polynomial, to NP, and even to exponential as a function of seemingly simple decisions made about the nature of dependencies among components.

2 COMPOSABILITY AND COMPONENT SELECTION

Composition and component selection have been researched comprehensively within the software engineering and simulation communities. In this section, we address the distinct contributions these two communities have made to the general composability problem and the specific challenge of selecting components.

2.1 Software Composability

In his widely read text on component software, Szyperski (2002) defines a composition as an "assembly of parts (components) into a whole (a composite) without modifying the parts." (Most software engineers would use the term "adapting" where Szyperski uses "modifying." We use "adapting" in keeping with established software engineering terminology.) Prohibiting adaptation of components is a recurring theme in software engineering literature. Many software engineers consider composition strictly in terms of functional composition, where each component possesses a clearly defined interface and functional description. We do not believe this restrictive approach suits the needs of the simulation community.

In the software engineering community, composability is typically discussed within the framework of component-based software design (CBSD), sometimes referred to as component-based software engineering. CBSD models include Microsoft's Component Object Model Plus (Microsoft 2005), the Object Management Group's Common Object Request Broker Architecture (OMG 2005), and Sun's Enterprise JavaBeans (Sun 2005). These technologies are similar because they enforce a binary structure for exposing public interfaces allowing components to provide services to clients, which may or may not themselves be components. The significant drawback to current CBSD technologies is that while they provide the facilities to communicate and provide services, they make no guarantees about the meaning, reliability, or consistency of the exchanged information.

2.2 Simulation Composability

Following extensive simulation community efforts to identify an acceptable definition, simulation composability has been defined as "the capability to select and assemble simulation components in various combinations into valid simulation systems to satisfy specific user requirements." (Petty and Weisel 2003) In contrast to Szyperski's definition this one allows consideration of adaptation. However Petty and Weisel reason that somewhere on the scale of integration effort, the problem changes from the composition of components to one of component interoperability when a "substantial effort" is required.

Within the simulation community both interoperability and composability have been topics of intense study. Composability is distinguished from interoperability in that composability requires an ability to combine and recombine components to meet different sets of requirements without substantial integration efforts. Interoperability, on the other hand, implies a one-time integration effort as is employed in distributed simulation architectures such as Distributed Interactive Simulation (IEEE a), Aggregate Level Simulation Protocol (Weatherly, Seidel, and Weissman 1991), and the High Level Architecture (IEEE b). These architectures have provided practical tools for federating simulations, but with the same kind of drawbacks seen in current CBSD technologies. Simply put, they provide little more than syntactic interoperability—the ability to exchange information. They make few if any guarantees about semantic interoperability because there are no guarantees about the meaning of exchanged data.

2.3 Software Reuse and Component Selection

The formal pursuit of software reuse has a history of nearly forty years (McIlroy 1968). Reuse can take one of many forms, ranging from functional composition as discussed in Section 2.1 to scavenging, often called accidental reuse. The forms of reuse are summarized in Krueger (1992). For our purposes, reuse is more general than composability because it allows for the adaptation of components as necessary, either to adjust to changing requirements, or because the set of components "as-is" do not completely satisfy all requirements.

Reuse research as it relates to our view of component selection generally takes one of three forms: 1) designing reuse metrics and models, 2) representing components and 3) selecting the best set of components. Reuse metrics and models are typically used to predict the costs and benefits, given a reuse strategy. They enable measurement of the tradeoffs associated with using a specific component. Many reuse metrics and models are summarized in Frakes and Terry (1996). Component representations support the description of components, and are necessary for mechanically mapping components to requirements. Component representations range from informal text-based descriptions, several of which are summarized in Frakes and Pole (1994), to formal descriptions (Mili, Mili, and Mittermeir 1994, Penix and Alexander 1999), and more recently to the use of ontologies and domain models (Sugumaran and Storey 2003). Finally, there is a need for a method to retrieve the best set of components given metrics and descriptions. Methods which have been investigated include model checking (Xie and Browne 2003), formal methods (Mili, Mili, and Mittermeir 1994, Penix and Alexander 1999), decision and utility theory (Alves et al. 2005, Kontio 1996), information retrieval (Maarek, Berry, and Kaiser 1991), and artificial neural networks (Merkl, Tjoa, and Kappel 1994).

2.4 Simulation Component Selection

Recent interest within the M&S community has focused on the simulation composability component selection problem (Page and Opper 1999, Petty, Weisel, and Mielke 2003,

Fox, Brogan, and Reynolds 2004). Informally, the component selection problem asks, given a set of simulation components and a set of objectives, does a subset of components of cardinality K or less exist that meets all objectives. Even with the existence of an oracle that can in one step determine which objectives have been met by any component or set of components, component selection has been shown to be NP-complete (Petty, Weisel, and Mielke 2003). More recently, a polynomial time approximation algorithm for component selection was discovered (Fox, Brogan, and Reynolds 2004).

The component selection problem, as it has been framed up to now, does not explicitly address changing objectives (requirements); it assumes all objectives are known at selection time. Also not explicitly considered are cost and time to adapt components to meet requirements. We believe a more flexible approach, recognizing the reality of requirements that can change and components that can be adapted, is needed.

3 PROBLEM DEFINITION

Stated concisely, our challenge is to select from a set of simulations a subset that can optimally (in terms of cost and utility) meet a set of target requirements, given that the component simulations are adaptable (at a cost). The novelty in our statement of the challenge resides in the flexibility introduced by assumptions of component adaptability.

3.1 Critical Assumptions

Consider R, the set of ρ requirements to be satisfied ("target requirements")

$$R = \{r_1, r_2, \ldots, r_\rho\}$$

by a subset of X, the set of γ components available for selection:

$$X = \{x_1, x_2, \ldots, x_\gamma\}$$

We assume a candidate subset of X does not have to meet all target requirements fully. Given sufficient time and effort, any component can be adapted to meet any requirement, so all components are potential candidates to meet some or all target requirements. Even the null component can meet a requirement, which is equivalent to creating a component anew. Since every component simulation can potentially satisfy every requirement, choosing an optimal subset of components creates an inherently intractable problem because the number of possible combinations of components grows exponentially with the number of components (given γ is the number of components, there are 2^γ possible subsets of components). Clearly, some components are better candidates for satisfying a given requirement than others and we assume this insight is available before composition begins. We find it reasonable to consider an upper bound on the number of requirements that can be satisfied by any one component. Call this bound θ. Furthermore, for every instance of component selection, it is known *a priori* which of up to θ requirements a component is assumed to be able to satisfy. We will show for constant values that θ prevents our formulation of the component selection problem from exhibiting exponential growth in complexity.

There is a utility associated with the pairing between every component, x_i, and the (up to) θ requirements, r_j, it satisfies. When performing component selection, the goal is to maximize the aggregate utility of the selected set of components. For a given set of components and a given set of target requirements we assume an appropriate utility function exists. Our formulation is consistent with formal utility theory, where utility is a real-valued measure applied to user preferences (von Neumann and Morgenstern 1947). We consider utility to be a function of the time and cost necessary to adapt a component to meet a requirement(s), where cost represents all potential resources (excluding time) necessary to adapt a simulation component. We know of no universal utility function. We assume a utility function can be defined for each instance of the component selection problem as we define it. While we have defined utility as a function of time and cost, it could just as easily be defined in terms of algorithmic efficiency, or any other metric we believe is important to capture.

We allow for the possibility that the utility of a component x_i satisfying requirement r_j might be a function of the number of other requirements x_i also satisfies. Thus, x_i may have one utility if it is ultimately selected to satisfy requirement r_j only, and it may have another utility if in addition to r_j, x_i is also chosen to satisfy another requirement. This is realistic, for example there can be benefit in employing fewer simulation components to satisfy a set of requirements.

Earlier we introduced θ, an upper bound on the number of requirements that can be met by a single component in a selected set of components. In the worst case, $\gamma(2^\theta - 1)$ sets of utility values would need to be computed to assign utilities to describe all possible ways γ components can satisfy their θ requirements. This follows from there being 2^θ possible combinations of the θ requirements a component can satisfy less one combination representing the case when a component satisfies zero requirements. Each of these sets of utility values represents a different scenario for a given component usage, with one or more utility computations necessary per scenario. The total number of utility computations required to complete all possible sets of utility values is $\gamma\theta(2^{\theta-1})$.

In our formulation of the component selection problem we have explored the use of a scaling factor, β, associated with all components, where the utility of the component

in satisfying multiple requirements scales by the factor times the number of additional requirements satisfied. The scaling factor captures the change in utility encountered when a component satisfies multiple requirements. By using a scaling factor to model the utility of a component's satisfaction of multiple requirements, we reduce the cost of performing component selection considerably, and yet retain a useful level of practicality.

Consider the practicality argument first. Using component x_i to satisfy requirement r_j will realize utility $u(x_i, r_j)$. If x_i is used to satisfy any single additional requirement, the utility of using x_i to satisfy r_j will now be $\beta * u(x_i, r_j)$. The β scaling factor reflects that the cost of the programmers' learning about x_i to satisfy r_j is subsequently amortized across the application of x_i to the additional requirement. The scaling factor can be applied in a similar fashion when x_i satisfies many additional requirements.

We believe that in the course of performing a component selection, a user is much more likely to be comfortable estimating a scaling factor than describing context dependent functions. While the existence of a scaling factor is not essential to our complexity analysis, it represents the sort of relaxed assumption we find reasonable. It does have some cost benefits as we discuss next.

Using the scaling factor dramatically reduces the number of utility values that must be generated for most problem instances. Earlier in this section, we stated that $\gamma\theta(2^{\theta-1})$ utility values must be computed in the worst case when the scaling factor is not used. Consider a component x_i, a utility function u that evaluates the utility of using x_i to satisfy any requirement r_j, and a scaling factor that specifies the utility of using x_i to satisfy more than one requirement. For any one of the γ components x_i in a problem instance, a utility must be computed for each of the θ individual requirements x_i could satisfy. Additionally, for each of the θ requirements that x_i satisfies it is required to compute the θ utilities for ways x_i can satisfy that specific requirement and $\theta - 1$ additional requirements. The total number of utilities that must be computed is $\gamma\theta^2$, which becomes significantly smaller (for $\theta > 2$) than the number of utilities that must be computed when the scaling factor is not utilized.

3.2 Informal Problem Description

We refer to our formulation of the component selection problem as "Applied Simulation Component Selection" (CS-ASCS). An instance of CS-ASCS is graphically depicted in Figure 1.

In this figure, there are seven labeled bins, with each bin corresponding to requirements r_1 through r_7. Each item within a bin corresponds to a uniquely computed utility value u_{ijq} corresponding to the ith component satisfying requirement j while also satisfying $q-1$ other requirements. In this problem instance we are using four components, 1 through 4. We set θ, the upper bound on how many requirements a given component can be adapted to meet, to 3. Because θ is assumed to be 3, no q for any of the u_{ijq} is greater than 3. In Figure 1, x_1 is a candidate for satisfying r_1 and r_7. Note that the scaling factor, β, allows us to compute u_{113} and u_{173} even though x_1 was only chosen to satisfy two requirements and these two utility values will never be utilized.

r_1	r_2	r_3	r_4	r_5	r_6	r_7
u_{111}	u_{221}	u_{331}	u_{441}	u_{251}	u_{461}	u_{171}
u_{112}	u_{222}	u_{332}	u_{442}	u_{252}	u_{462}	u_{172}
u_{113}	u_{223}	u_{333}	u_{443}	u_{253}	u_{463}	u_{173}
	u_{321}	u_{431}			u_{261}	u_{371}
	u_{322}	u_{432}			u_{262}	u_{372}
	u_{323}	u_{433}			u_{263}	u_{373}

Figure 1: Example Instance of CS-ASCS

Given a problem instance, our goal is to select the set of components that meets all requirements while maximizing the total utility value. For the example problem instance, one entry should be selected from each bin. For any selected entry, u_{ijq}, where $q > 1$, there must be $q-1$ entries selected from $q-1$ other bins possessing the same values for i and q as the original selected entry. That is, for any component, x_i whose utility, u_{ijq}, is selected from the jth bin, $q-1$ other utilities for component x_i must be chosen from $q-1$ other bins.

In Figure 2, we show an equivalent view of the problem depicted in Figure 1. This equivalent view can be constructed for any problem instance.

The view in Figure 2 lends itself better to reasoning about the time complexity of component selection given our formulation of the problem.

In support of the problem representation in figure 2, R, ρ, X, γ and θ remain as we have defined them above. The set U corresponds to all possible computed utility values for a given problem instance (e.g. every item appearing in a bin in Figure 1). The set C is a collection of collections, where each C_k for k in the range $1..\gamma$, is a collection of subsets of U. Note that these subsets and collections are not constructed arbitrarily. Each C_k corresponds to all possible uses of a given component for the problem instance. For the example given in Figure 2, every possible usage of component x_2 is described by a subset in C_2. In a solution to this problem instance, we can either choose not to use x_2

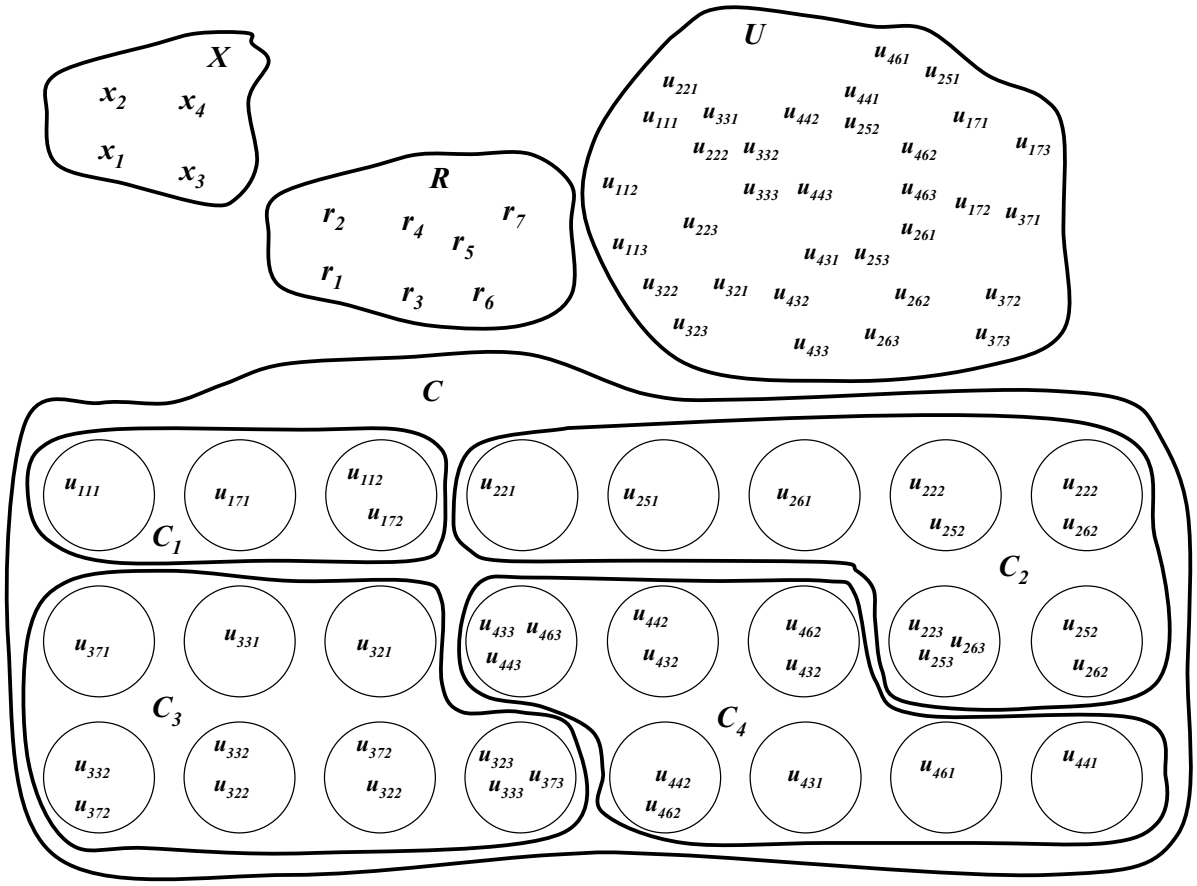

Figure 2: Set View Instance of CS-ASCS

at all (choose none of the subsets in C_2), or we can choose exactly one subset from C_2. More generally, we can choose no more than one subset from each C_k because each subset represents a different scenario for satisfying a requirement, and the computed utility values for each subset member are dependent on this chosen scenario. The utility value for a given component scenario (subset) is simply the sum of the subset members, and will be described in the formal definition of CS-ASCS as the function f.

Selecting subsets in C to satisfy the requirements in R is equivalent to searching for an exact cover of R. The goal is an exact cover because CS-ASCS allows only one component to meet any given requirement. There is no emergence (Page and Opper 1999) or duplication of requirement satisfaction among components. Intuitively, identifying an exact cover of R ensures each requirement is satisfied by exactly one component. Since the elements of U and R are not of the same type, we must construct a mapping from elements in U to elements in R in order to cover R. This mapping is simply a projection of the second subscript of each element of U to the corresponding unique requirement in R (e.g. u_{123} is mapped to requirement r_2).

3.3 Formal Problem Definition

Informally, the problem description in the set view is to choose no more than one subset from each C_k to exactly cover R while maximizing aggregate utility. More formally the problem is:

CS-ASCS
INSTANCE: Set $R = \{r_1, r_2, \ldots, r_\rho\}$ of requirements, set $X = \{x_1, x_2, \ldots, x_\gamma\}$ of components, set U of utility values where each $u_{ijq} \in U$ corresponds to the ith component satisfying requirement j while also satisfying $q-1$ other requirements (with an arbitrary upper bound θ on q), collection C constructed from U (per the construction in Section 3.2) where each C_k is a collection of subsets of U, function $f : 2^U \rightarrow \Re$ that sums utility values, $sum \in \Re$.
QUESTION: Is there an exact cover S of R constructed by choosing no more than one element z_k from each C_k such that $S = \bigcup_k z_k$ and $\sum_{s \in S} f(s) \geq sum$?

We have posed this as a decision problem (yes or no answer required). Alternatively, we could have asked the following optimization question:
QUESTION: What is the maximum utility of an exact cover

of R with no more than one element chosen from each C_k? In general, a given problem is more difficult to solve as an optimization problem than as a decision problem. In this instance, we will consider the complexity of the decision problem first.

First we show that a user can build an instance of CS-ASCS in polynomial time given a set of components and a set of requirements. Recall that γ is the cardinality of the set of components, and θ is an upper bound on the number of requirements a component can satisfy. To build an instance of CS-ASCS:

1. Iterate through each component, deciding for each component which requirements it can satisfy (not exceeding θ). In the worst case, every component is considered as a candidate for satisfying every requirement; this worst case requires time $\in O(\gamma\rho)$.
2. Compute utilities. In the worst case this activity requires time $\in O(\gamma\theta^2)$(using the scaling factor as shown in Section 3.1).
3. Form the collection C: γ collections of subsets. In the worst case the time required to form the subsets is $\in O(\gamma(2^\theta - 1) = O(\gamma 2^\theta)$. Recall that θ is bounded. Therefore the time required to form the subsets in the worst case is $\in O(\gamma)$.

From steps 1-3 we conclude that the overall time complexity to build an instance of CS-ASCS is $\in O(\gamma\theta^2)$, which is polynomial in the length of the input.

4 COMPUTATIONAL COMPLEXITY OF CS-ASCS

We prove that CS-ASCS is NP-complete, thus arguing its intractability. To do so, we use the EXACT COVER BY 3-SETS problem which is known to be NP-complete (Garey and Johnson 1979, Karp 1972). This problem is defined as follows:

EXACT COVER BY 3-SETS (X3C)
INSTANCE: Set X with $|X| = 3q$ and a collection C of 3-element subsets of X.
QUESTION: Does C contain an exact cover for X, i.e., a sub-collection $C' \subseteq C$ such that every element of X occurs in exactly one member of C'?

X3C asks the following question: Given a set X where $|X| \bmod 3 = 0$, and a collection of subsets of X, where each subset has exactly three members, can X be exactly covered by some or all of the subsets? By definition, an exact cover requires a one-to-one correspondence between the members of the subsets forming the exact cover and the elements of X. The subsets forming the exact cover will contain no elements that are "left over", and no element in X will be covered more than once.

We restate our original problem for convenience:

CS-ASCS
INSTANCE: Set $R = \{r_1, r_2, \ldots, r_\rho\}$ of requirements, set $X = \{x_1, x_2, \ldots, x_\gamma\}$ of components, set U of utility values where each $u_{ijq} \in U$ corresponds to the ith component satisfying requirement j while also satisfying $q-1$ other requirements (with an arbitrary upper bound θ on q), collection C constructed from U (per the construction in Section 3.2) where each C_k is a collection of subsets of U, function $f : 2^U \rightarrow \Re$ that sums utility values, $sum \in \Re$.
QUESTION: Is there an exact cover S of R constructed by choosing no more than one element z_k from each C_k such that $S = \bigcup_k z_k$ and $\sum_{s \in S} f(s) \geq sum$?

4.1 CS-ASCS is NP-complete

Theorem 1 *CS-ASCS is NP-complete.*

Proof By reduction from X3C. We first show that CS-ASCS is in NP. Then a function is constructed to transform an instance of X3C to an instance of CS-ASCS. We show this transformation is in P. Lastly, we show that a problem instance is an element of X3C if and only if the transformed instance is an element of CS-ASCS.

Given any set S, we can verify in polynomial time whether $S \in$ CS-ASCS. Recall that the members of S are subsets. We iterate through each of these subsets and verify which requirement is met by each subset member (second subscript in our notation). Additionally we keep a cumulative utility value using f, so for each member we add the utility to this running total. After we iterate through all subsets, we verify that each requirement was met once and only once, in addition to verifying that the result of f (cumulative utility) is $\geq sum$. In the worst case there are γ subsets, each with θ members. Therefore, this verification process is $\in O(\gamma\theta)$ and CS-ASCS $\in$ NP.

Next we define a transformation function g that transforms any instance I of X3C to an instance $g(I)$ of CS-ASCS. The transformation function g is defined as follows:

1. For every $c_i \in C_{X3C}$, form a new collection, d_i, that has one element, a set that also has only one element, the utility value 1 mapped to c_i, labeled 1_{c_i}. The result is a new collection $D_{X3C} = \{d_1, d_2, \ldots\}$ where each element of D_{X3C} is a collection containing one set, with that set containing a single utility value mapped to a member of C. Copy D_{X3C} to $C_{CS-ASCS}$. Add each 1_{c_i} to $U_{CS-ASCS}$.
2. Assign $sum = \frac{|X_{X3C}|}{3}$.
3. Copy X_{X3C} to $R_{CS-ASCS}$.

Let $r = |C_{X3C}|$ and $s = |X_{X3C}|$. Step 1 of the transformation requires time $\in O(r)$. Step 2 $\in O(s)$ assuming the

number of items in X_{X3C} must be counted. Step $3 \in O(s)$. Therefore, the overall time complexity of the transformation g is polynomial in the length of the input.

Next we show that $I \in Z_{X3C}$ if and only if $g(I) \in Z_{CS-ASCS}$.

$\longrightarrow$: Assume $I \in Z_{X3C}$. Then there exists a subset $C'_{X3C} \subseteq C_{X3C}$ that exactly covers X_{X3C}. By g, there exists a one-to-one correspondence between C'_{X3C} and a set $C'_{CS-ASCS} \subseteq C_{CS-ASCS}$. If C'_{X3C} exactly covers X_{X3C}, then by g, $C'_{CS-ASCS}$ will exactly cover $R_{CS-ASCS}$. Since through g, there is only one element in each $C_{i_{CS-ASCS}} \in C_{CS-ASCS}$, then no more than one element from each $C_{i_{CS-ASCS}}$ could have been chosen to form the exact cover. By g, $sum = \frac{|X_{X3C}|}{3} = |C'_{X3C}|$. Since there is a one-to-one correspondence between C'_{X3C} and $C'_{CS-ASCS}$, and each element of $C'_{CS-ASCS}$ possesses a utility of 1 in the exact cover, the total utility of the exact cover in $g(I)$ equals $|C'_{CS-ASCS}| = sum$. Therefore, $g(I) \in Z_{CS-ASCS}$.

$\longleftarrow$: Assume $g(I) \in Z_{CS-ASCS}$. Then there exists a subset $C'_{CS-ASCS} \subseteq C_{CS-ASCS}$ that exactly covers $R_{CS-ASCS}$. By g, there exists a one-to-one correspondence between $C'_{CS-ASCS}$ and a set $C'_{X3C} \subseteq C_{X3C}$, and a one-to-one correspondence between $R_{CS-ASCS}$ and X_{X3C}. If $C'_{CS-ASCS}$ exactly covers $R_{CS-ASCS}$, then by g, C'_{X3C} exactly covers X_{X3C}. Therefore, $I \in Z_{X3C}$.

We have shown that $I \in Z_{X3C}$ if and only if $g(I) \in Z_{CS-ASCS}$. This completes the proof of Theorem 1. □

4.2 The Optimization Problem

We can informally restate an instance of the decision problem for CS-ASCS as an optimization problem as follows:

CS-ASCS-OPT
QUESTION: What is the maximum utility of an exact cover of R with no more than one element chosen from each C_i?

Theorem 2 *CS-ASCS-OPT is NP-hard.*

Proof (informal) We argue that all problems in NP reduce to CS-ASCS-OPT. As stated in the proof for Theorem 1, X3C is NP-complete, therefore all problems in NP can be reduced to X3C. X3C can be reduced to CS-ASCS-OPT as follows. Run a transformed instance of X3C through a machine that solves CS-ASCS-OPT. If the maximum utility returned is zero, then the answer to the X3C decision problem is NO because an exact cover does not exist. Any other answer from the CS-ASCS-OPT machine implies an exact cover exists, and therefore the answer to the decision problem is YES. This completes the proof of Theorem 2. □

We have now shown that all instances of X3C can be reduced to CS-ASCS-OPT, and therefore CS-ASCS-OPT is NP-hard. However, note that CS-ASCS-OPT cannot be proven NP-complete because there is no known way to verify in polynomial time the maximum utility for any given problem instance.

4.3 Dealing with Complexity

Proving a problem to be NP-complete provides convincing evidence that an optimal polynomial-time algorithm is unlikely to be found. However, alternatives exist. First, if the input size is sufficiently small and remains small, then the problem could be approached using brute force. Second, while the worst case may be NP-complete, the average case may be shown to be much less complex. Lastly, approximation algorithms and heuristics are a class of solutions that can give very good, reasonably close to optimal, results in a reasonable amount of time.

Since most simulations are of reasonable complexity, we believe that those built from components, especially small and simple components as advocated in (Bartholet et al. 2004), will have enough requirements and components from which to choose that brute force will not be an option for component selection. It is most likely that success will be found in a semi-automated approach, where a combination of heuristics and subject matter expert insight find a reasonable first approximation that can be further refined if necessary.

4.4 The Effect of One Assumption on the Problem Complexity Class

The constant θ is critical in the definition of CS-ASCS. Recall θ is an arbitrary bound on the maximum number of requirements a given component can meet, and is applied before the component selection process is carried out (i.e. no component appears in the ρ bins for requirements more than θ times. The nature of CS-ASCS changes considerably as θ takes on values of 1, 2, 3 or greater than 3. We discuss how this range of values for θ affects the complexity of the problem.

For $\theta = 1$ (CS-ASCS-1), this is a problem instance where each component can satisfy at most one requirement. A trivial algorithm for optimally solving CS-ASCS-1 is to place all components that meet the same requirement into a bin, sort the bin in order of decreasing utility value, and select the first component in each bin. Clearly this is an optimal polynomial time algorithm for CS-ASCS-1.

For $\theta = 2$ (CS-ASCS-2) the problem complexity is unknown. Many, but not all, problems have polynomial time solutions when independent variables associated with the problem have a value of two. To show that CS-ASCS-2 is of polynomial time complexity, we must either show a reduction of CS-ASCS-2 to a problem such as Exact-Cover-by-2-Sets (X2C), which is known to be of polynomial time complexity (Garey and Johnson 1979), or we must construct a polynomial time algorithm directly. A procedure

for reducing to X2C is not obvious, mainly because the weights in CS-ASCS-2 have no equivalent in X2C. We will continue to explore this question.

For $\theta \geq 3$ (in effect, bounding the number of requirements any component can satisfy), this is an NP-complete problem because when $x \geq 3$ an X3C to CS-ASCS reduction as shown in Section 4.1 is possible. However, if θ is unbounded, i.e. there is no limit on the number of requirements a given component can meet, then CS-ASCS requires an exponential time algorithm. This conclusion is derived from the analysis in Section 3.3 where we showed the need to consider $O(\gamma 2^{\theta})$ subsets in the problem solution. In this case θ is growing unbounded making CS-ASCS a problem of exponential time complexity.

A second independent variable we consider in our formulation of the component selection problem is the scaling factor (β). Recall β is used in determining the utility of a component for satisfying a requirement when that component is also used to satisfy other requirements. The component's final utility would be its utility for satisfying the first requirement times the scaling factor raised to the power of the remaining number of requirements the component satisfies. There would be one scaling factor for each component, requirement pair. If we assume that all scaling factors are equal to one, then our formulation of the component selection problem has a polynomial time complexity independent of the value of θ. This result is important because it means there is an interesting class of component selection formulations for which there is a polynomial time solution. We intend to explore this insight further.

4.5 Generalizing from CS-ASCS

We chose utility, computed as a function of the pair (x_i, r_j), as the measure of goodness in our search for an optimal component selection method. We can generalize our results to other metrics, and the problem remains NP-complete.

We further generalize our formulation as follows: assume there are dependencies in the component selection process such that component-requirement pairs are required to be selected together in groups of size 1 or more. A selection possesses an associated weight. As an example, assume that component x_{i1} is selected to satisfy requirement r_{j1}. Owing to a dependence this selection requires component x_{i1} to be selected to satisfy r_{j2}, component x_{i2} to satisfy requirements r_{j3} and r_{j4}, and component x_{i3} to satisfy requirement r_{j5}. We assume this set of component-requirement pairs possesses a weight computed by a function not specified here. The weight could be used for maximizing or minimizing an objective (e.g. minimize if the weight is a cost, maximize if the weight is a benefit). More formally:

CS-ASCS-X
INSTANCE: Set $R = \{r_1, r_2, \ldots, r_\rho\}$ of requirements, set $X = \{x_1, x_2, \ldots, x_\gamma\}$ of components, set $Y = X \times R$, collection C where each C_i is a weighted subset of Y with maximum cardinality of θ, function $f : C_i \rightarrow \Re$ that sums subset weights, $k \in \Re$.
QUESTION: Is there an exact cover $Z \subseteq C$ of R such that $\sum_{z \in Z} f(z) \geq k$? (Without loss of generality, in this problem description the weight is maximized.)

From this problem description we state Corollary 1 to Theorem 1:

Corollary 1 *If $\theta \geq 3$, then CS-ASCS-X is at least NP-complete.*

Corollary 1 is proven by showing that X3C is a restricted case of this problem in the same way that X3C was shown to be a restricted case of CS-ASCS. The corollary is stated "at least NP-complete" to account for those instances where the number of dependencies is so large that the input to the problem grows exponentially (e.g. the case where $|C| = |2^Y|$).

Informally, Corollary 1 is interpreted to mean that once dependencies are considered in a weighted component selection problem, finding an optimal solution quickly becomes intractable as the problem scales. However, as shown in Section 4.4, there are cases where constraining dependencies makes the problem tractable.

For a typical set of components, a typical set of requirements, subject matter expert knowledge and experience and other factors as yet not considered in the literature, the component selection problem may not be as seriously constrained as theoretical treatment suggests. While the results presented here are theoretically similar to the results in (Page and Opper 1999, Petty, Weisel, and Mielke 2003), the flexibility provided to the component selection problem by adaptable components gives encouragement for future progress toward acceptable solutions. For a given metric that facilitates the selection of components, the acceptable number of dependencies among components that an efficient analysis can support is still not determined. While the theoretical limit appears to be no more than two, other factors may support a larger limit. The nature of those other factors remains to be explored.

5 CONCLUSIONS

In this paper, we have shown that the complexity of optimal reuse of adaptable components is NP-complete. We were motivated to pursue this study of adaptable components because it is impossible to anticipate future requirements when creating component sets and because the most efficient way to construct a complex simulation may be to adapt existing components. This work is unique in its formal treatment of how components can be changed or adapted to satisfy multi-

ple requirements. However, the ways in which a component can be adapted are constrained according to inter-component dependencies that permit precise analysis of multiple component reuse scenarios. As a function of seemingly simple decisions made about the nature of dependencies among components, we demonstrated that the complexity of the component selection problem can vary from polynomial, to NP, and even to exponential. We generalized these results to show that regardless of the types or reasons for dependencies in component selection, just their mere existence makes this problem very difficult to solve optimally. Our future work will explore the existence of approximation algorithms and heuristics that can achieve good, but not necessarily optimal, results in component selection while retaining the flexibility that component adaptation provides to the practitioner.

ACKNOWLEDGMENTS

We gratefully acknowledge support from the DDDAS program at the National Science Foundation (ITR 0426971), the U.S. Army, as well as from our colleagues in the Modeling and Simulation Technology Research Initiative (MaSTRI) at the University of Virginia.

REFERENCES

Alves, C., X. Franch, J. P. Carvallo, and A. Finkelstein. 2005, February. Using goals and quality models to support the matching analysis during COTS selection. In *Proceedings of the 4th International Conference on COTS-Based Software Systems*. Bilbao, Spain.

Bartholet, R. G., D. C. Brogan, P. F. Reynolds, Jr., and J. C. Carnahan. 2004, September. In search of the philosopher's stone: Simulation composability versus component-based software design. In *Proceedings of the Fall 2004 Simulation Interoperability Workshop*. Orlando, FL.

Darema, F. 2004, June. Dynamic data driven application systems: A new paradigm for application simulations and measurements. In *Proceedings of the 2004 International Conference on Computational Science*. Krakow, Poland.

Fox, M. R., D. C. Brogan, and P. F. Reynolds, Jr.. 2004, December. Approximating component selection. In *Proceedings of the 2004 Winter Simulation Conference*, ed. R. G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters, 421-426. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Frakes, W. B., and T. P. Pole. 1994. An empirical study of representation methods for reusable software components. *IEEE Transactions on Software Engineering* 20 (8): 617–630.

Frakes, W., and C. Terry. 1996. Software reuse: Metrics and models. *ACM Computing Surveys* 28 (2): 415–435.

Garey, M. R., and D. S. Johnson. 1979. *Computers and intractability: A guide to the theory of NP-completeness*. W.H. Freeman and Company.

Institute of Electrical and Electronics Engineers a. IEEE Standard for Distributed Interactive Simulation - Application Protocols. IEEE std. 1278.1.

Institute of Electrical and Electronics Engineers b. IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA). IEEE std. 1516-2000.

Karp, R. 1972. *Complexity of computer computations*, Chapter Reducibility Among Combinatorial Problems. Plenum Press.

Kontio, J. 1996, March. A case study in applying a systematic method for COTS selection. In *Proceedings of the 18th International Conference on Software Engineering*. Berlin, Germany.

Krueger, C. W. 1992. Software reuse. *ACM Computing Surveys* 24 (2): 131–183.

Maarek, Y. S., D. M. Berry, and G. E. Kaiser. 1991. An information retrieval approach for automatically constructing software libraries. *IEEE Transactions on Software Engineering* 17 (8): 800–813.

McIlroy, M. 1968, October. Mass-produced software components. In *Report on a Conference by the NATO Science Committee*. Garmish, Germany.

Merkl, D., A. M. Tjoa, and G. Kappel. 1994, November. Learning the semantic similarity of reusable software components. In *Proceedings of the IEEE International Conference on Software Reusability*. Rio de Janeiro, Brazil.

Microsoft 2005. *COM: Component Object Model technologies*. <www.microsoft.com/com/default.mspx> [last visited February 2005]: World Wide Web.

Mili, A., R. Mili, and R. Mittermeir. 1994, May. Storing and retrieving software components: A refinement based system. In *Proceedings of the 16th International Conference on Software Engineering*. Sorrento, Italy.

Object Management Group 2005. *Corba*. <http://www.omg.org/corba/> [last visited February 2005]: World Wide Web.

Page, E. H., and J. M. Opper. 1999, December. Observations on the complexity of composable simulation. In *Proceedings of the 1999 Winter Simulation Conference*, ed. P. A. Farrington, H. B. Nembhard, D. T Sturrock, and G. W. Evans, 553-560. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Penix, J., and P. Alexander. 1999. Efficient specification-based component retrieval. *Automated Software Engineering* 6 (2): 139–170.

Petty, M. D., and E. W. Weisel. 2003, April. A composability lexicon. In *Proceedings of the Spring 2003 Simulation Interoperability Workshop*. Orlando, FL.

Petty, M. D., E. W. Weisel, and R. R. Mielke. 2003, September. Computational complexity of selecting components for composition. In *Proceedings of the Fall 2003 Simulation Interoperability Workshop*. Orlando, FL.

Sugumaran, V., and V. C. Storey. 2003. A semantic-based approach to component retrieval. *ACM SIGMIS Database* 34 (3): 8–24.

Sun Microsystems 2005. *J2EE Enterprise Javabeans technology*. http://java.sun.com/products/ejb/ [last visited February 2005]: World Wide Web.

Szyperski, C. 2002. *Component software: Beyond object-oriented programming*. 2d ed. Addison-Wesley.

von Neumann, J., and O. Morgenstern. 1947. *Theory of games and economic behavior*. second ed. Princeton University Press.

Waziruddin, S., D. C. Brogan, and P. F. Reynolds, Jr.. 2003, September. The process for coercing simulations. In *Proceedings of the Fall 2003 Simulation Interoperability Workshop*. Orlando, FL.

Weatherly, R., D. Seidel, and J. Weissman. 1991, July. Aggregate level simulation protocol. In *Proceedings of the 1991 Summer Computer Simulation Conference*. Baltimore, MD.

Xie, F., and J. C. Browne. 2003, September. Verified systems by composition from verified components. In *Proceedings of the 9th European Software Engineering Conference*. Helsinki, Finland.

AUTHOR BIOGRAPHIES

ROBERT G. BARTHOLET is a Ph.D. Candidate in Computer Science and a member of MaSTRI at the University of Virginia. Robert earned his B.S. in Electrical Engineering at the U.S. Military Academy at West Point, and his Masters in Computer Science at the University of Virginia. Robert is an active duty Lieutenant Colonel in the U.S. Army with 9 years experience in the Field Artillery. For the past 8 years he has served in the Signal Corps as an Information Systems Engineer supporting battlefield communication and automation systems in tactical units. His email address is <bartholet@cs.virginia.edu>.

DAVID C. BROGAN earned his Ph.D. from Georgia Tech and is currently an Assistant Professor of Computer Science and a member of MaSTRI at the University of Virginia. For more than a decade, he has studied simulation, control, and computer graphics for the purpose of creating immersive environments, training simulators, and engineering tools. His research interests extend to artificial intelligence, optimization, and physical simulation. His email address is <dbrogan@cs.virginia.edu>.

PAUL F. REYNOLDS, JR. is a Professor of Computer Science and a member of MaSTRI at the University of Virginia. He has conducted research in modeling and simulation for over 25 years, and has published on a variety of M&S topics including parallel and distributed simulation, multiresolution models and coercible simulations. He has advised numerous industrial and government agencies on matters relating to modeling and simulation. He is a plank holder in the DoD High Level Architecture. His email address is <reynolds@cs.virginia.edu>.