

A Description

This problem demonstrates how to solve an architecting problem in the software domain using SoS Explorer using a toy example. The problem is one where a software architecture is needed for a new data analytics application. The necessary functionality required is known, but the problem is how to obtain it optimally for a given set of business objectives. For this problem, a component is a software implementation delivering functionality and a package is a set of one or more components. There are a number of available off-the-shelf (OTS) components that may be used in the overall solution. There is also the option to develop certain components in-house. All in-house development must be in Java, Scala, or Python. Each component may be written in one any of these languages regardless of the languages used for other components. The application itself must also be written in one of these languages.

The performance of the software development effort will be evaluated by the following objectives: affordability, performance, scalability, flexibility, and robustness. The functionality required by the application falls under four categories and has been identified as follows:

- Report generation
 - Office Open XML (OOXML)
 - Portable Document Format (PDF)
- Visualization
 - Graph
 - Flow
 - Multi-dimensional (MD)
 - Geospatial (GS)
- Clustering
 - BIRCH
 - DBSCAN
- Predictive Modeling
 - Support vector machines (SVM)
 - Generalized linear models (GLM)
 - Deep belief networks (DBN)

The following four tables detail the software application and component options along with their implementation language, cost, performance, scalability, and functionality. The performance and scalability are relative indexes where larger numbers are better. Table 1 lists the options for the report generating components, table 2 is for the visualization components, and table 3 is for the clustering and predictive modeling components. Table 4 is for the application development itself.

B Modeling

B.1 Overview

When using SoS Explorer, it is important to note that it seeks to maximize objectives. Therefore entities that need to be minimized, such as cost, must be recast as a maximized entity such as affordability. Also, entities that are selectable need to be cast as systems. In this case, software packages act as systems. The other items must be treated as either characteristics, capabilities,

Table 1: Report Generation Software Packages

Package	Language	Cost	Perf	Scaling	OOXML	PDF
Report A	Java	\$2,000	2	2	✓	✓
Report B	Python	\$1,000	1	2	✓	✓
Report C	Java	\$500	1	1	✓	
Report D	Java	\$500	1	1		✓

Table 2: Visualization Software Packages

Package	Language	Cost	Perf	Scaling	Graph	Flow	MD	GS
Visual A	Java	\$5,000	4	3	✓	✓	✓	✓
Visual B	Python	\$2,000	1	2	✓	✓	✓	✓
Visual C	Python	\$400	1	1	✓	✓	✓	
Visual D	Java	\$2,000	5	3				✓
Visual E	Python	\$1,500	2	3				✓

or interfaces. In SoS Explorer, characteristics are represented using real numbers and describe a property of a system. Therefore, the characteristics are language (Java, Scala, or Python), cost, performance, and scalability. The capabilities are boolean values and are the elemental components of functionality such as PDF support. The interfaces are a boolean matrix indicating which systems are capable of interfacing with one another.

The characteristics matrix, (\mathbf{C}), has dimensions $N_C \times N_S$ where N_C is the number of characteristics and N_S is the number of systems. C_{ij} is defined as the i th characteristic of the j th system. Likewise, the capabilities matrix, (\mathbf{C}'), has dimension $N_{C'} \times N_S$ where $N_{C'}$ is the number of capabilities. Finally, the feasible interface matrix, (\mathbf{F}), has dimension $N_S \times N_S$.

Together, the systems, characteristics (\mathbf{C}), capabilities (\mathbf{C}'), and feasible interfaces (\mathbf{F}) define the problem's meta-architecture. From an optimization standpoint, the meta-architecture is static and stands apart from an architecture. In SoS Explorer, an architecture is simply a set of systems and interfaces. These sets are defined in a vector called a chromosome by the evolutionary algorithms used to optimize the architecture. The functions S and I extract the system and interface information from a chromosome and are defined as

$$S(\mathbf{X}, i) = \begin{cases} 1 & \text{if the } i\text{th system is selected in } \mathbf{X} \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

and

$$I(\mathbf{X}, i, j) = \begin{cases} 1 & \text{if the } i\text{th and } j\text{th systems have an interface in } \mathbf{X} \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

where \mathbf{X} is the chromosome.

Table 3: Clustering and Predictive Modeling Software Packages

Package	Language	Cost	Perf	Scaling	BIRCH	DBSCAN	SVM	GLM	DBN
Cluster A	Java	\$1,000	4	3	✓	✓			
Cluster B	Scala	\$1,000	4	6	✓	✓			
Cluster C	Python	\$500	1	2	✓	✓			
PM A	Java	\$3,000	4	3	✓	✓	✓	✓	✓
PM B	Scala	\$3,000	4	6	✓	✓	✓	✓	✓
PM C	Python	\$1,000	1	2	✓	✓	✓	✓	✓
In-house A	Java	\$200	2	2	✓	✓			
In-house B	Scala	\$200	2	4	✓	✓			
In-house C	Python	\$100	1	1	✓	✓			
In-house D	Java	\$400	2	2			✓	✓	✓
In-house E	Scala	\$400	2	4			✓	✓	✓
In-house F	Python	\$200	1	1			✓	✓	✓

Table 4: Application Software

Package	Language	Cost	Perf	Scaling
Application A	Java	\$10,000	3	2
Application B	Scala	\$15,000	3	6
Application C	Python	\$5,000	1	1

B.2 Objectives

The objectives are individually modeled as a function of the chromosome containing the selected systems and interfaces along with the characteristics, capabilities, and feasible interfaces. The first objective, affordability (O_1), can be modeled as

$$O_1(\mathbf{X}, \mathbf{C}) = - \sum_{i=1}^{N_S} S(\mathbf{X}, i) \left(\mathbf{C}_{\text{Cost}, i} + \sum_{\substack{j=1 \\ j \neq i}}^{N_S} S(\mathbf{X}, j) \mathbf{C}_{\text{Cost}, j} \beta_{\text{Lang}, i, \text{Lang}, j} \right) \quad (3)$$

where \mathbf{X} is the chromosome and $\beta_{i,j}$ is the cost factor to create an interface between components with languages i and j . These cost factors are defined in Table 5. The second objective, performance (O_2), can be modeled by averaging the performance values of the selected components:

$$O_2(\mathbf{X}, \mathbf{C}) = \frac{\sum_{i=1}^{N_S} S(\mathbf{X}, i) \mathbf{C}_{\text{Perf}, i}}{\sum_{i=1}^{N_S} S(\mathbf{X}, i)} \quad (4)$$

The next objective, scalability (O_3), can be modeled by averaging the scalability values:

$$O_3(\mathbf{X}, \mathbf{C}) = \frac{\sum_{i=1}^{N_S} S(\mathbf{X}, i) C_{\text{Scale}, i}}{\sum_{i=1}^{N_S} S(\mathbf{X}, i)} \quad (5)$$

The fourth objective, flexibility (O_4), is the surplus of capabilities (components) and can be modeled by

$$O_4(\mathbf{X}, \mathbf{C}') = -N_{C'} + \sum_{i=1}^{N_S} S(\mathbf{X}, i) \sum_{j=1}^{N_{C'}} C'_{ji} \quad (6)$$

Hence, flexibility is related to the ability to perform substitutions within the architecture. The final objective, robustness (O_5), is a measure of interface redundancy and can be modeled as

$$O_5(\mathbf{X}) = -N_S + \sum_{i=1}^{N_S} S(\mathbf{X}, i) \sum_{j=i}^{N_S} S(\mathbf{X}, j) I(\mathbf{X}, i, j) \quad (7)$$

This predicts the maximum impact of an interface between two components.

Table 5: Inter-Language Interface Costs

Language	Java	Scala	Python
Java	0.01	0.02	0.04
Scala	0.02	0.01	0.04
Python	0.04	0.04	0.01

B.3 Constraints

The evolutionary algorithms allow constraints to be enforced via a mechanism known as chromosome fixing. Using this technique, the chromosomes are modified to meet feasibility requirements. An issue with using fixing is that it can work against the evolutionary process. To mitigate this, the actual chromosomes are not modified, but rather a function, \mathbf{G} , is used to map the chromosome to its feasible compliment which is then passed into the objective function. In other words, when constraints are enabled then the objectives are passed $\mathbf{G}(\mathbf{X})$ instead of \mathbf{X} . This way, the evolutionary operations are not undermined by the enforcing of constraints.

There are two constraints required by this problem. The first constraint is that each capability (component) must be present in the architecture. This may be performed by Algorithm 1. The second constraint is that all the interfaces must be feasible and that the required paths are present. The required paths are listed in Table 6 and are represented by the matrix γ . The associated algorithms for removing infeasible interfaces and adding missing ones are in Algorithms 2 and 3 respectively. Missing paths are remedied in the simplest manner possible, which is to add an interface directly between the systems (packages). Since a path is required, feasibility of this interface is assumed.

Algorithm 1 Add missing capabilities

```

1: procedure REQUIREALLCAPABILITIES( $\mathbf{X}, \mathbf{C}'$ )
2:   for  $i \leftarrow 1$  to  $N_{\mathbf{C}'}$  do                                     ▷ For each capability
3:      $j \leftarrow 0$                                                ▷ System index
4:      $k \leftarrow -1$                                              ▷ Non-selected system with capability  $i$ 
5:      $hasCapability \leftarrow \text{false}$ 
6:     while  $\neg hasCapability \wedge (j \leq N_S)$  do
7:       if  $\mathbf{C}'_{ij}$  then                                           ▷ If system  $j$  has capability  $i$ 
8:         if  $S(\mathbf{X}, j)$  then                                       ▷ If system  $j$  is present
9:            $hasCapability \leftarrow \text{true}$                              ▷ Capability  $i$  is present
10:        else
11:           $k \leftarrow j$                                            ▷ Remember non-selected system with capability  $i$ 
12:        end if
13:      end if
14:       $j \leftarrow j + 1$                                            ▷ Next system
15:    end while
16:    if  $\neg hasCapability \wedge (k \neq -1)$  then                       ▷ If capability  $i$  is missing
17:       $\mathbf{X}' \leftarrow \text{SETSYSTEM}(\mathbf{X}, k, \text{true})$                  ▷ Add system  $k$  with capability  $i$ 
18:    else
19:       $\mathbf{X}' \leftarrow \mathbf{X}$                                            ▷ No changes to chromosome
20:    end if
21:  end for
22:  return  $\mathbf{X}'$ 
23: end procedure

```

Algorithm 2 Remove infeasible interfaces

```

1: procedure REMOVEINFEASIBLEINTERFACES( $\mathbf{X}, \mathbf{F}$ )
2:    $\mathbf{X}' \leftarrow \mathbf{X}$                                              ▷ Copy chromosome
3:   for  $i \leftarrow 1$  to  $N_S$  do                                     ▷ For each system  $i$ 
4:     for  $j \leftarrow 1$  to  $N_S$  do                                     ▷ For each system  $j$ 
5:       if  $i \neq j$  then                                           ▷ Only consider different systems
6:         if  $I(\mathbf{X}, i, j)$  then                                       ▷ If interface is present
7:           if  $\neg(S(\mathbf{X}, i) \wedge S(\mathbf{X}, j) \wedge \mathbf{F}_{ij})$  then         ▷ If not feasible
8:              $\mathbf{X}' \leftarrow \text{SETINTERFACE}(\mathbf{X}', i, j, \text{false})$      ▷ Remove interface
9:           end if
10:        end if
11:      end if
12:    end for
13:  end for
14:  return  $\mathbf{X}'$ 
15: end procedure

```

Algorithm 3 Add required paths

```

1: procedure ADDREQUIREDPATHS( $\mathbf{X}, \mathbf{C}$ )
2:    $\mathbf{X}' \leftarrow \mathbf{X}$  ▷ Copy chromosome
3:   for  $i \leftarrow 1$  to  $N_S$  do ▷ For each system  $i$ 
4:     if  $S(\mathbf{X}, i)$  then ▷ If system is selected
5:       for all  $m \in \{\text{RG, Vis, Cluster, PM, App}\}$  do ▷ For each major function  $m$ 
6:         if  $C_{mi} \neq 0$  then ▷ If system  $i$  has function  $m$ 
7:           for all  $n \in \{\text{RG, Vis, Cluster, PM, App}\}$  do ▷ For each major function  $n$ 
8:             if  $(m \neq n) \wedge \gamma_{mn}$  then ▷ If path required
9:               for  $j \leftarrow 1$  to  $N_S$  do ▷ For each system  $j$ 
10:                if  $(i \neq j) \wedge S(\mathbf{X}, j) \wedge (C_{nj} \neq 0)$  then ▷ If system  $j$  has function  $n$ 
11:                  if  $\neg \text{HASPATH}(\mathbf{X}', i, j)$  then ▷ If path is missing
12:                     $\mathbf{X}' \leftarrow \text{SETINTERFACE}(\mathbf{X}', i, j, \text{true})$  ▷ Add interface
13:                  end if
14:                end if
15:              end for
16:            end if
17:          end for
18:        end if
19:      end for
20:    end if
21:  end for
22:  return  $\mathbf{X}'$ 
23: end procedure

```

Table 6: Required Paths By Function

	Reporting	Visualization	Clustering	Predictive	Application
Reporting			✓	✓	✓
Visualization			✓	✓	✓
Clustering	✓	✓			✓
Predictive	✓	✓			✓
Application	✓	✓	✓	✓	✓