01 Nov 2016

Multiobjective System of Systems Architecting with Performance Improvement Funds

Hadi Farhangi

Dincer Konur
Missouri University of Science and Technology, konurd@mst.edu

Cihan H. Dagli
Missouri University of Science and Technology, dagli@mst.edu

Recommended Citation

H. Farhangi et al., "Multiobjective System of Systems Architecting with Performance Improvement Funds," *Procedia Computer Science*, vol. 95, pp. 119-125, Elsevier, Nov 2016.
The definitive version is available at https://doi.org/10.1016/j.procs.2016.09.301

Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 95 (2016) 119 – 125

Procedia Computer Science

Complex Adaptive Systems, Publication 6
Cihan H. Dagli, Editor in Chief
Conference Organized by Missouri University of Science and Technology
2016 - Los Angeles, CA

Multiobjective System of Systems Architecting with Performance Improvement Funds

Hadi Farhangi[a], Dincer Konur[a]*, Cihan H. Dagli[a]

[a]*Missouri University of Science and Technology, Engineering Management and Systems Engineering, 600 W. 14th Street, Rolla MO, 65409, USA*

Abstract

A System of Systems architecting problem aims to determine a selection of systems, which are capable of providing a set of desired capabilities. A SoS architect usually has multiple objectives in generating efficient architectures such as minimization of the total cost and maximization the overall performance of the SoS. This study formulates a biobjective SoS architecting problem with these two objectives. Here, we consider that, by allocating funds to the systems, the SoS architect can improve the performance of the capabilities the systems can provide. The resulting architecting problem is a biobjective mixed-integer linear programming model. Specifically, the system selection decisions are binary while the fund allocation decisions are continuous. We first discuss the application of the adaptive epsilon-constraint method as an exact method for solving this model. Then, we propose an evolutionary method and compare its performance with the exact method. Finally, a numerical study demonstrates the benefits of fund allocation in the SoS architecting process.

Peer-review under responsibility of scientific committee of Missouri University of Science and Technology

Keywords: System of Systems; Multiobjective optimization; Performance Improvement

1. Introduction and Literature Review

The system of systems (SoS) is a system, whose components are systems themselves[1]. SoS needs a set of capabilities and these capabilities come from systems that form the SoS[2]. It is worth mentioning the variety of

* Corresponding author. Tel.: +1-573-341-7256.
E-mail address: konurd@mst.edu

Peer-review under responsibility of scientific committee of Missouri University of Science and Technology
doi:10.1016/j.procs.2016.09.301

applications of SoS in military, engineering, healthcare, and transportation[3,4,5]. During the construction of a SoS, the architect typically accounts for multiple objectives such as the minimization of the total cost and maximization of the overall performance of the constructed SoS[6]. This study assumes that the cost minimization and performance maximization are the SoS architect's objectives and accordingly formulates a biobjective SoS architecting problem. Here, we consider that the SoS architect can improve the performance of the capabilities that the selected systems can provide by allocating funds to them. A similar study of Konur and Dagli[6] investigates a related topic, where the systems negotiate with the SoS architect for fund allocation. In particular, Konur and Dagli[6] assume that the systems individually decide on how to utilize the allocated funds for achieving maximum performance improvements in their capabilities. Here, on the other hand, we consider that the SoS architect directs how the systems should use the allocated funds. Specifically, the SoS architect specifies how much of the allocated fund should be utilized in the improvements of the capabilities that a selected system can provide.

Note that it is possible to increase the overall performance of the SoS by allocating more funds to the systems; however, this will also increase the total cost of the SoS. We define the overall SoS performance as the sum of the performances of the capabilities provided by the selected systems. The total cost of the SoS is defined as the sum of the fixed capability costs charged by the systems, the funds allocated to the systems, and the cost of interfaces used to assure connectivity of the SoS architecture. The problem of interest in this study can be defined as follows: Which systems should be selected and how much funds should be allocated to each capability of the selected systems in order to minimize the total cost and maximize the overall performance of the SoS guaranteeing that the SoS is capable and connected? In Section 2, we give the formulation of this problem. Section 3 explains the solution analysis. The numerical studies are summarized in Section 4 and Section 5 concludes the paper.

2. Problem Formulation

The SoS architecting problem is to find a subset of the m available systems to provide the entire set of n capabilities such that the resulting SoS is connected and it shows high performance and low cost. In addition, a total fund amount of F is available to assign to the selected systems in order to improve their performances in providing capabilities. Therefore, in addition to which systems to select, SoS architect should also decide how to allocate this total fund among the selected systems. Particularly, let capabilities be indexed by i such that $i \in I$, where $I = \{1, \dots, n\}$, and systems indexed by j such that $j \in J$, where $J = \{1, \dots, m\}$. Let us define $x_j = 1$ if system j is included in the SoS and $x_j = 0$ otherwise, and let $\boldsymbol{X}$ be the $m \times 1$ −vector of x_j's. For SoS connectivity, a variable y_{qp} is defined such that $y_{qp} = 1$ if both systems p and q are included in the SoS, i.e. $x_p = 1$ and $x_q = 1$, and $y_{qp} = 0$ otherwise. Let $\boldsymbol{Y}$ be the $m \times m$ −matrix of y_{qp} values. For fund allocation decisions, we define continuous variables $f_{ij} \geq 0$ as the amount of funds that is being allocated to system j to improve its performance in providing capability i. Let $\boldsymbol{F}$ be the $n \times m$ −matrix of f_{ij} values.

A system can provide some or all of the capabilities required by the SoS. Let $A_{ij} = 1$ if system j can provide capability i and $A_{ij} = 0$ otherwise, and $\boldsymbol{A}$ be the $n \times m$ −matrix of A values. Moreover, we define c_{ij} and p_{ij} as the cost and the performance (without any additional improvement spending) of system j in providing capability i, respectively. Furthermore, to assure connectivity, interfaces should be used between any pair of selected systems. Let h_{pq} be the cost of connecting system p to system q with an interface. In this study, similar to Konur and Dagli[6], we assume that the performance of systems in providing capabilities can be improved linearly by the fund allocations. Specifically, let $\Delta p_{ij} = \alpha_{ij} f_{ij}$ be the increase over p_{ij} by allocating f_{ij} amount of funds to system j's capability i, where α_{ij} defines the rate of improvement in the performance of system j in providing capability i. Since, there should be a natural upper bound on the maximum performance achievable, we also define $\overline{f_{ij}}$ as the upper bound for the amount of funds that can be allocated to system j to improve capability i.

Based on the above discussion, the SoS problem of interest (**P-SoS**) can be formulated as follows:

$$\max \quad TP(\boldsymbol{X}, \boldsymbol{F}) = \sum_{i \in I} \sum_{j \in J} A_{ij} p_{ij} x_j + \sum_{i \in I} \sum_{j \in J} \alpha_{ij} f_{ij}$$

$$\min \quad TC(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{F}) = \sum_{i \in I} \sum_{j \in J} A_{ij} c_{ij} x_j + \sum_{p \in J} \sum_{q \in J, q > p} h_{pq} y_{pq} + \sum_{i \in I} \sum_{j \in J} f_{ij}$$

$$s.t. \quad \sum_{j \in J} A_{ij} x_j \geq 1 \qquad \forall i \in I \qquad (1)$$

$$y_{pq} + y_{qp} \geq x_p + x_q - 1 \qquad \forall p, q \in J, q > p \qquad (2)$$

$$\sum_{i\in I} f_{ij} \le F x_j \qquad \forall j \in J \qquad (3)$$

$$\sum_{i\in I}\sum_{j\in J} f_{ij} \le F \qquad (4)$$

$$0 \le f_{ij} \le A_{ij}\overline{f_{ij}} \qquad \forall i \in I, \forall j \in J \qquad (5)$$

$$x_j \in \{0,1\} \qquad \forall j \in J \qquad (6)$$

$$y_{pq} \in \{0,1\} \qquad \forall p,q \in J, q > p \qquad (7)$$

The first objective is the maximization of the total performance $TP(\boldsymbol{X},\boldsymbol{F})$, where the first part is the total performance of the included systems in providing their capabilities and the last part is the total improvement in performances after the allocation of funds.

Linear summation of the individual performances of systems, as the first part of this objective function, is rather a simplistic approach to capture the performance of SoS. In real world application, this part of the objective can be replaced by a weighted sum of individual performances given that the weights that are known beforehand by the decision maker. The second objective function is the minimization of the total cost $TC(\boldsymbol{X},\boldsymbol{Y},\boldsymbol{F})$, where the first part is the total cost of the selected systems in providing their capabilities, the second part is the cost of interfaces, and the last part is the total allocated funds. Constraints (1) guarantee that at least one system provides each capability. These constraints are covering constraints, which make **P-SoS,** $NP-hard$. Constraints (2) ensure that every pair of selected systems in the SoS are connected. Since this set of constraints are symmetric, in which $y_{pq}+y_{qp} \ge x_p + x_q - 1$ and $y_{qp}+y_{pq} \ge x_q + x_p - 1$ are equivalent, we only consider the first set by index relation $q > p$. Constraints (3) assure that if system j is selected, the total allocated fund to this system cannot exceed the maximum funds available; if it is not selected, we do not allocate any fund to it. Constraint (4) guarantees that the total allocated funds is less than the available fund. Constraints (5) define non-negativity and upper bound for f_{ij}; if $a_{ij} = 1$, the upper bound is $\overline{f_{ij}}$, otherwise the upper bound is zero, which makes $f_{ij} = 0$. Constraints (6) and (7) define the rest of variables as binary variables.

3. Solution Analysis

P-SoS is a biobjective mixed integer linear programming problem and due to the covering constraints (1), even the single objective case is $NP-hard$. To solve such a problem, one may reduce the two objectives into a single one by using a weighted sum approach or it is possible to find a solution that is at the lowest distance to the optimum of each objective[8]. In this work, however, our aim is to approximate the set of Pareto efficient solutions. A solution $\boldsymbol{S} = [\boldsymbol{X},\boldsymbol{Y},\boldsymbol{F}]$ is a Pareto efficient solution for **P-SoS** if and only if there exists no other solution $\boldsymbol{S}' = [\boldsymbol{X}',\boldsymbol{Y}',\boldsymbol{F}']$ such that $TP(\boldsymbol{X}',\boldsymbol{F}') \ge TP(\boldsymbol{X},\boldsymbol{F})$ and $TC(\boldsymbol{X}',\boldsymbol{Y}',\boldsymbol{F}') \le TC(\boldsymbol{X},\boldsymbol{Y},\boldsymbol{F})$ with at least one of the inequalities being strict. Due to the complexity of the problem, we next develop an evolutionary algorithm that approximates a set of Pareto efficient solutions in two stages. At the first stage, the method generates a feasible SoS, i.e., $\boldsymbol{X}$, say $\widehat{\boldsymbol{X}}$**,** and generates $\boldsymbol{Y} = \widehat{\boldsymbol{Y}}$ based on $\widehat{\boldsymbol{X}}$ considering the connections between any pairs of systems that are presented in $\widehat{\boldsymbol{X}}$. At the second stage, Pareto efficient fund allocations $\boldsymbol{F}$ for the given SoS $(\widehat{\boldsymbol{X}},\widehat{\boldsymbol{Y}})$ are generated by solving the following biobjective linear programming problem.

$$\textbf{P-}\boldsymbol{SoS}_{\widehat{\boldsymbol{X}}}: \quad \max \ \widehat{TP}(\boldsymbol{F}) = \sum_{i\in I}\sum_{j\in J}\alpha_{ij} f_{ij}$$

$$\min \ \widehat{TC}(\boldsymbol{F}) = \sum_{i\in I}\sum_{j\in J} f_{ij}$$

$$s.t. \ \text{Eqs. (3)-(5).}$$

The problem **P-**$\boldsymbol{SoS}_{\widehat{\boldsymbol{X}}}$ is a biobjective linear programming problem and is the direct result of **P-SoS**, when $\boldsymbol{X} = \widehat{\boldsymbol{X}}$ and $\boldsymbol{Y} = \widehat{\boldsymbol{Y}}$. The objective functions of this problem are shown with $\widehat{TP}(\boldsymbol{F})$ as the total performance and $\widehat{TP}(\boldsymbol{F})$ as the total cost, which both are functions of $\boldsymbol{F}$. It will be discussed later that the efficient solutions to this problem provide a trade-off for fund allocation to systems considering both objectives of this problem.

The basic steps of the two-stage evolutionary algorithm are similar to the other works in the literature[8,9], which are explained next.

(1) *Chromosome representation and initialization:* We define the vector $\boldsymbol{X}$ as the chromosome. To generate a feasible chromosome, we randomly generate a binary vector of size m and check if all capabilities are provided. If a capability is missing, we select a non-selected system which can provide the missing capability (replace 0 by 1). Using this approach, we generate t_0 chromosomes as the initial population.

(2) *Chromosome evaluation and finding vectors* $\boldsymbol{Y}$ *and* $\boldsymbol{F}$*:* Given a feasible $\widehat{\boldsymbol{X}}$, vector $\widehat{\boldsymbol{Y}}$ is generated by considering the connections between any two pairs of systems that are selected in $\widehat{\boldsymbol{X}}$. However, to compute vector $\boldsymbol{F}$, **P-**$\boldsymbol{SoS}_{\widehat{\boldsymbol{X}}}$ should be solved. To do so, we use the classical ϵ-constraint method, which solves $\min\{\widehat{\mathrm{TC}}(\mathrm{F}): \widehat{\mathrm{TP}}(\mathrm{F}) \geq \epsilon, \text{Eqs. (4), (5), (8)}\}$ for increasing values of ϵ. This way, for a given chromosome $\widehat{\boldsymbol{X}}$, we generate *N* number of Pareto efficient fund vectors, namely $\boldsymbol{F_1}$, $\boldsymbol{F_2}$,…, $\boldsymbol{F_N}$ and get $\{(\widehat{\boldsymbol{X}}, \widehat{\boldsymbol{Y}}, \boldsymbol{F_1}), \ldots, (\widehat{\boldsymbol{X}}, \widehat{\boldsymbol{Y}}, \boldsymbol{F_N})\}$ solutions for **P-SoS**. Once these solutions are generated for each chromosome in the population, the evaluation step determines the set of Pareto efficient $(\widehat{\boldsymbol{X}}, \widehat{\boldsymbol{Y}}, \boldsymbol{F_i})$ solutions by pairwise comparison of all such solutions. The unique chromosomes generating at least one Pareto efficient $(\widehat{\boldsymbol{X}}, \widehat{\boldsymbol{Y}}, \boldsymbol{F_i})$ are accepted as the parent chromosomes for the next population.

(3) *Mutation:* Given a set of parent chromosomes, we perform adding, dropping, and swapping mutations. For the adding mutation, we consider each 0 in a given vector $\boldsymbol{X}$ and we replace it with 1 to generate a new chromosome. If there are α_a of 0's in a chromosome, adding mutation will generate t_0 new feasible chromosomes. For the dropping mutation, we consider each 1 in the vector $\boldsymbol{X}$ and we drop make it 0. If the resulting chromosome is still feasible, we accept it. For swapping mutation, we find all 0's and 1's. We swap any pair of 0's and 1's and if the resulting vector is feasible, we accept it. In addition to these mutation operators, we randomly generate α_1 new feasible solutions using the process described in step (1).

(4) *Termination:* The algorithm will stop if the parent chromosomes do not change for t_1 consecutive iterations.

To verify the performance of the above evolutionary algorithm, we compute a subset of the set of efficient solutions for **P-SoS** using the adaptive $\epsilon-$constraint method. The method is a variation of the adaptive $\epsilon-$constraint method of Laumann, 2006[7]. The adaptive $\epsilon-$constraint version of **P-SoS**, i.e. $\boldsymbol{SoS-\epsilon}$, is created by adding objective TP in constraints.

$$\boldsymbol{SoS-\epsilon}: \begin{cases} \min & TC(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{F}) \\ s.t. & TP(\boldsymbol{X}, \boldsymbol{F}) \geq \epsilon \\ & \text{Eqs. (1)} - \text{(7)} \end{cases}$$

By varying ϵ values within bounds of objective TP and updating it based on the latest evaluation of this function, we can find a subset of Pareto efficient solutions. That is, different than increasing ϵ with equal steps as in the classical ϵ-constraint method, we increase it based on the latest solution. The reason for this as follows. Since, this problem is a mixed-integer model, the bound on *TP* is not necessarily tight. Therefore, by considering the *TP* of the latest solution, we avoid solving unnecessary mixed-integer models.

4. Numerical Analysis

In this section, we perform two sets of numerical study. First, we compare the adaptive $\epsilon-$constraint method with the evolutionary method we presented in the previous section. Second, we use the evolutionary algorithm to investigate the effects of allocating funds in the SoS architecture. The analysis is performed mainly on the integer part of solutions, i.e. vector $\boldsymbol{X}$. The rationale behind the decision is the fact that one can generate $\boldsymbol{Y}$ and $\boldsymbol{F}$ very easy for a given $\boldsymbol{X}$, as discussed in the step (2) of the evolutionary algorithm.

4.1. Comparison of the Algorithms

To compare the adaptive $\epsilon-$constraint method with the evolutionary method, we consider 9 problem classes corresponding to each combination of $n \in \{3,6,9\}$ and $m \in \{3,6,9\}$. For each class, 10 instances are randomly generated, in total 90 instances, and the averages of the results over these 10 problem instances are considered. All

90 instances are solved by both algorithms. For every instance, parameters are randomly generated as $c_{ij} \in UI[20,40]$, $p_{ij} \in UI[20,40]$, $h_{pq} \in UI[1,5]$, $\overline{f_{ij}} \in UI[5,10]$, and $\alpha_{ij} \in UI[1,4]$, where $UI[a,b]$ is a uniform discrete distribution between a and b. For this study, we assumed that the total available fund is equal to the summation of the maximum individual funds that we can assign to each system, i.e. $F = \sum_{i \in I} a_{ij} \overline{f_{ij}}$ as we do not allocate funds to a system to improve a capability that it originally cannot provide. Finally, we generated the matrix $\boldsymbol{A}$ as a binary random matrix of size $n \times m$. Then, every row of the matrix is checked to see if there is a system that can provide the corresponding capability. If not, we randomly select a system to provide that capability. Furthermore, the settings of evolutionary algorithm are $t_0 = n$ and $t_1 = n$.

Table 1 summarizes the results of comparison between the algorithms. We only compared the number of unique integer parts, i.e. vector $\boldsymbol{X}$ that we refer to as the Unique Integer Part (UIP), in the final set of returned solutions by the algorithms. Superscripts e and a are used to address adaptive $\epsilon-$constraints and evolutionary algorithm, respectively. In this table, $|\boldsymbol{X}|$ is the number of unique integer parts (NUIP) of the set of solutions and cpu records the computational time in seconds. In addition, $\#pop$ and $|pop|$ show the number of evaluated populations and the average size of the population of the evolutionary algorithm, respectively. The results show that in average 97.22% of NUIP of the solutions returned by the evolutionary algorithm are indeed integer parts of the Pareto efficient solutions returned by the adaptive $\epsilon-$constraint method. Furthermore, evolutionary algorithm requires less computational time on average. These confirm the quality of evolutionary algorithm.

Table 1 Comparison of Exact and Approximated Algorithms

n	m	$\lvert X^e \rvert$	$\lvert X^a \rvert$	% X^a in X^e	cpu^e	cpu^a	$\#\ pop$	$\lvert pop \rvert$
3	3	3.3	3.3	100%	0.71	0.05	3.8	9.91
	6	15.7	20.7	95.59%	2.68	0.39	5.2	54.99
	9	27.9	48.6	93.75%	4.87	2.46	7	228.25
6	3	2.3	2.3	100%	0.74	0.05	3.2	7.98
	6	13.7	19.4	97.22%	4.28	0.32	4.7	43.22
	9	33.3	57.7	94.87%	11.86	2.66	6.1	249.42
9	3	1.5	1.5	100%	0.58	0.08	3	5.43
	6	9.7	11.9	98.75%	5.67	0.20	4.1	25.25
	9	32.3	62	94.79%	20.37	2.76	5.7	251.61
Mean		15.52	25.27	97.22%	5.75	0.99	4.76	97.34

4.2. Analysis on Funds

In this section, we investigate the case of a SoS architecting with funds and the case of SoS architecting without fund allocations. Setting of the numerical analysis for this part is very similar to Section 4.1, except the size of problem instances is increased. Here, we consider every combination of $n \in \{4,8,12\}$ and $m \in \{4,8,12\}$. The notation is as follows: superscripts n and f refer to the no-funding case and the funding case, respectively. The notations for $|X|$, cpu, $\#pop$, and $|pop|$ are similar to the previous subsection. In addition, the following tables have a column for $|X^U|$, which is the NUIP that we get by combining all the non-dominated solutions of the two cases. We refer to X^U as the Union of Unique Integer Parts (Union-UIP) and its size as the Union-NUIP.

Table 2 summarizes the quantitative comparison of non-funding SoS versus funding SoS, while Table 3 demonstrates the qualitative comparison. The following observations are based on these tables:

- On average, SoS architecting without funds generates less Pareto efficient SoS architectures compared to the SoS architecting with fund allocations (see Table 1 for $|X^n|$ vs. $|X^f|$). This is because, some of the SoS architectures may be dominated unless improvements are achieved by fund allocations. Also, as expected, evolutionary algorithm evaluates more SoS architectures in case of fund allocations are allowed, which also is the reason for higher computational time (see Table 1 for $|pop^n|$ vs. $|pop^f|$).
- In all of the 90 problem instances, 100% of UIP returned by SoS with funds appear within Union-UIP, while this percentage for SoS without funds is 75%. That is, some of the Pareto efficient SoS architectures returned by

SoS architecting without funds are dominated by Pareto efficient SoS architectures enhanced with funds returned by SoS architecting with funds. In addition, NUIP that is shared between SoS with and without funds and Union-UIP is 71% of the NUIP of the Union-UIP. Finally, 99.94% of NUIP of the Union-UIP comes from SoS with funds, while this percentage for SoS without funds is 71.85% in average.

These observations show that SoS architecting with funds can improve both objectives of SoS architecting and it is recommended to allocate funds to systems to improve their performances.

Table 2 Quantitative Comparison of Non-funding vs. Funding

n	m	$\lvert X^U \rvert$	$\lvert X^n \rvert$	$\lvert X^f \rvert$	cpu^n	cpu^f	$\#pop^n$	$\#pop^f$	$\lvert pop^n \rvert$	$\lvert pop^f \rvert$
4	4	5.1	4.9	5.1	0.04	0.07	3.70	3.80	5.62	13.94
	8	37.2	35.5	37.2	0.72	1.30	6.00	6.00	50.60	141.10
	12	88.5	80.4	88.4	4.73	13.07	7.60	7.50	264.38	1017.98
8	4	4.2	4.1	4.2	0.05	0.06	3.20	3.30	5.46	9.47
	8	44.5	39.8	44.5	0.79	1.45	5.60	5.70	52.00	129.93
	12	128.4	101.4	128.4	6.30	21.30	7.30	7.00	319.09	1197.38
12	4	2.4	2.4	2.4	0.04	0.04	3.10	3.10	3.13	7.49
	8	45	43.6	45	0.86	1.46	5.40	5.30	55.59	132.17
	12	141.9	138.3	141.5	8.53	25.25	7.20	6.60	349.37	1226.51
Average		55.24	50.04	55.19	2.45	7.11	5.46	5.37	122.80	430.66

Table 3 Qualitative Comparison of Non-funding vs. Funding

n	m	$\frac{\lvert X^n \cap X^U \rvert}{\lvert X^n \rvert}$	$\frac{\lvert X^f \cap X^U \rvert}{\lvert X^f \rvert}$	$\frac{\lvert X^n \cap X^U \rvert}{\lvert X^U \rvert}$	$\frac{\lvert X^f \cap X^U \rvert}{\lvert X^U \rvert}$	$\frac{\lvert X^f \cap X^n \cap X^U \rvert}{\lvert X^U \rvert}$
4	4	96.25%	100 %	94.17%	100%	94.17%
	8	72.71%	100 %	71.93%	100%	71.93%
	12	52.64%	100 %	47.87%	99.87%	47.74%
8	4	96.33%	100 %	96.25%	100%	96.25%
	8	74.20%	100 %	66.79%	100%	66.79%
	12	57.88%	100 %	46.63%	100%	46.63%
12	4	100%	100 %	100%	100%	100%
	8	72.44%	100 %	70.53%	100%	70.53%
	12	53.87%	100 %	52.45%	99.57%	52.02%
Average		75.15%	100 %	71.85%	99.94%	71.78%

5. Conclusion

This study investigates a SoS architecting problem, which allows allocating funds to the systems for performance improvements. The problem is formulated as a biobjective mixed integer linear programming model. An evolutionary method for Pareto front approximation is proposed and the quality of algorithm is compared to the ϵ-constraint method. Through the comparison, the quality of evolutionary algorithm is confirmed. The next section of the numerical study demonstrates that by fund allocation, better solutions can be achieved that can improve both objectives.

Acknowledgements

This work is partially supported by the US Department of Defense through the Systems Engineering Research Center (SERC) under Contract HQ0034-13-D-0004. SERC is a federally funded University Affiliated Research Center managed by Stevens Institute of Technology.

References

1. Maier MW. Architecting principles for systems-of-systems. In: INCOSE International Symposium; 1996 Jul 1. p. 565-573.
2. Agarwal S, Pape LE, Dagli CH, Ergin NK, Enke D, Gosavi A, Qin R, Konur D, Wang R, Gottapu RD. Flexible and Intelligent Learning Architectures for SoS (FILA-SoS): Architectural Evolution in Systems-of-Systems. Procedia Computer Science. 2015;**44**:76-85.
3. Jamshidi MO. System of systems engineering-New challenges for the 21st century. Aerospace and Electronic Systems Magazine, IEEE; 2008 May;23(5): p.4-19.
4. Jamshidi M. Introduction to system of systems. System of Systems Engineering. Innovations for the 21st Century; Jamshidi, M., editor. 2008; 13: 1-43.
5. Jamshidi M, editor. System of systems engineering: innovations for the twenty-first century. John Wiley & Sons; 2011.
6. Konur D, Dagli CH. Military system of systems architecting with individual system contracts. *Optimization Letters*. 2015;**9**(8):1749-67.
7. Laumanns M, Thiele L, Zitzler E. An efficient, adaptive parameter variation scheme for metaheuristics based on the epsilon-constraint method. *European Journal of Operational Research*. 2006;**169**(3):932-942.
8. Konur D, Farhangi H, Dagli CH. On the Flexibility of Systems in System of Systems Architecting. *Procedia Computer Science*. 2014;**36**:65-71.
9. Konur D, Farhangi H, Dagli CH. A multi-objective military system of systems architecting problem with inflexible and flexible systems: formulation and solution methods. *OR Spectrum*. 2016:1-40.