A Computational Intelligence Approach to System-of-Systems Architecting Incorporating Multi-Objective Optimization

David M. Curry

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

Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 44 (2015) 86 – 94

Procedia Computer Science

2015 Conference on Systems Engineering Research

A computational intelligence approach to system-of-systems architecting incorporating multi-objective optimization

David M. Curry[a,*], Cihan H. Dagli[a]

[a]*Department of Engineering Management and Systems Engineering, Missouri University of Science and Technology, Rolla, Missouri 65409*

Abstract

A computational intelligence approach to system-of-systems architecting is developed using multi-objective optimization. Such an approach yields a set of optimal solutions (the Pareto set) which has both advantages and disadvantages. The primary benefit is that a set of solutions provides a picture of the optimal solution space that a single solution cannot. The primary difficulty is making use of a potentially infinite set of solutions. Therefore, a significant part of this approach is the development of a method to model the solution set with a finite number of points allowing the architect to intelligently choose a subset of optimal solutions based on criteria outside of the given objectives. The approach developed incorporates a meta-architecture, multi-objective genetic algorithm, and a corner search to identify points useful for modeling the solution space. This approach is then applied to a network centric warfare problem seeking the optimum selection of twenty systems. Finally, using the same problem, it is compared to a hybrid approach using single-objective optimization with a fuzzy logic assessor to demonstrate the advantage of multi-objective optimization.

Keywords: System-of-Systems; SoS; Architecting; Genetic Algorithm; GA; Multi-Objective Optimization Problem; MOP; Multi-Objective Evolutionary Algorithm; MOEA; Meta-Architecture

1. Introduction

System architecting is the process of defining the form and underlying structure of a system to fulfill a stated need and a System-of-Systems (SoS) is a "supersystem" of other elements that are themselves independent operational systems that through interaction achieve a goal[1]. Therefore, System-of-Systems architecting may be defined as the

* Corresponding author. Tel.: +1-636-699-0997
E-mail address: dmcfh3@mst.edu

process of the defining form and underlying structure of a system comprised entirely of systems. The key difference being that System-of-Systems architecting becomes a combinatorial problem when the systems have common interfaces (e.g. Link-16). In fact, this is the goal in joint warfighting System-of-Systems[2]. This makes the architecting problem amenable to solution by computational means. However, the architecting process typically involves optimizing trade-offs between multiple objectives where the functions modeling the objectives are not differentiable. Therefore, this is a Non-Gradient Optimization (NGO) problem and a candidate for a Computational Intelligence (CI) approach. Computational Intelligence is comprised of algorithms that mimic intelligent behaviors in order to find optimal solutions to problems and include Evolutionary Algorithms (EA), Particle Swarm Optimization (PSO), and Artificial Neural Networks (ANN) to name of few[3].

As an optimization problem with multiple objectives, one choice that must be made is whether to solve the problem as a Multi-objective Optimization Problem (MOP) or as a hybrid method that reformulates the problem as a Single-objective Optimization Problem (SOP). It might seem obvious to just use a MOP solver, but this approach creates significant difficulties because instead of working with a single optimum, there is a now a set of optimum solutions (the Pareto or non-dominated set). To avoid these difficulties, the problem can be restated as a single objective problem by defining a function that maps all the objectives to a single value for use in a SOP solver. This may be accomplished many ways, the simplest being weighting and then summing the objectives. A more sophisticated approach is to use a Fuzzy Logic System (FLS)[4,5] to evaluate each solution (known as a fuzzy assessor)[6]. The advantage to the single objective approach is that it simplifies the solution space to a single "best" solution. This is also the disadvantage because it fails to provide any other information about the solution space.

Single objective approaches find an optimum (best) solution by reducing the dimension of the objective space to a single dimension, thereby losing information about the structure of the solution space. This information loss is avoided by using a multiple objective approach. However, multiple objective approaches often yield large sets of optimum solutions, none of which are a priori better than another. This can yield significant information about the solution space, but is difficult to make use of.

A multiple objective approach is developed that makes the solution set more readily understood by choosing a small set of solutions from the Pareto set that provide a useful picture of the solution space. This approach begins by defining a meta-architecture to represent the actual architecture. A MOP solver is then used to generate and evolve architectures until a satisfactory approximation to the Pareto set is found. Finally, a method to choose the solutions best representative of the optimum solution space is presented. The MOP solver will be based upon a Genetic Algorithm (GA)[7,8] because of their wide-spread use, effectiveness, and simplicity.

2. Meta-Architecture

The meta-architecture is the representation of the architecture. It should be able to represent all possible architectures and, whenever possible, naturally enforce all constraints on the architecture. The meta-architecture should also fit with the algorithm chosen to optimize the architecture. For example, if a Genetic Algorithm is used as the optimizer, then a chromosome is the ideal meta-architecture because Genetic Algorithms use chromosomes to represent solutions.

The definition of the meta-architecture, just like the actual architecture of a system, will ultimately determine the limits on the performance, robustness, extensibility, and simplicity of the Computation Intelligence approach taken. If the meta-architecture is able to represent all possible feasible solutions without representing any non-feasible solutions (in other words, naturally enforces all feasibility constraints), then feasibility may be assumed. This simplifies the algorithm that generates and evolves the architecture greatly because it does not need to enforce constraints explicitly which not only complicates the algorithm, but can prevent the use of off-the-self standard implementations.

While a meta-architecture may represent the architecture explicitly, it is sometimes preferable to make some aspects implicit. For example, if an aspect of a solution, such as its topology, naturally follows based upon the other choices made, then there is no need to complicate the meta-architecture with a representation of the topology. The key benefit of keeping the chromosome as small as possible is that the GA will be able to better explore and exploit the solution space in a given amount of time.

3. Architecture Generation and Evolution

Since the meta-architecture is the representation of all possible architectures, to generate an architecture is simply to realize the meta-architecture with actual values. Genetic algorithms find optimal solutions given nothing more than a particular representation (the meta-architecture in this case), making them is a natural choice for this approach. To be clear, genetics algorithms are being employed here because they are easy to apply to this problem because of the met-architecture representation, not because GAs are faster or find better solutions than other algorithms for this type of problem. Because GAs find optimal solutions by generating random solutions (by populating the meta-architecture) and evolving them according to specific operations modeled after genetic ones, this process can be termed architecture generation and evolution. The GA used in this approach is the Canonical Genetic Algorithm (CGA)[9] minimally extended to work with a set of optimal (non-dominated) solutions instead of a single optimal solution. However, any Multi-objective Optimization Evolutionary Algorithm (MOEA) such as MOGA[10], NSGA-II[11], or SPEA2[12] could be used in its place.

The Canonical Genetic Algorithm can be considered the basic genetic algorithm and is comprised of four main elements[13]:

- A chromosome with a bitstring representation,
- Proportional selection to select parents for recombination,
- One-point crossover operator as the primary method to produce offspring, and
- Mutation operator.

For the purpose of this discussion the details of how the Canonical Genetic Algorithm works is not important. What is important is that the CGA creates an initial population of individuals. These individuals are simply randomly populated chromosomes. The fitness of each individual is then evaluated and half of the population is then selected to produce offspring. The selection is random, but the chance of selection increases proportionately with the fitness of the individual. After parents are chosen, offspring is produced though one point crossover which produces offspring by choosing a random point within the chromosome and copying one parent's chromosome though that point and the other parent's for the rest. The mutation operator then randomly changes the value of individual bits in the chromosome. The fittest individual is then recorded for this generation and the process of selecting parents and producing offspring is repeated until successive generations fail to produce a fitter individual.

Back to the chromosome. A chromosome is comprised of genes. Each gene represents a particular attribute of the whole and is represented in binary using the minimum number of bits required to represent the range of possible values of the particular attribute. Because the chromosome is comprised of a string of bits, it is called a bitstring representation. If n represents the number of choices (values) possible, then x = ceiling($\log_2 n$), where x is the number of bits required for the gene. For example, if a gene represents a system to be employed and there are fifteen different systems to choose from, then four bits would be required for this chromosome. The structure of a chromosome is shown in Fig. 1.

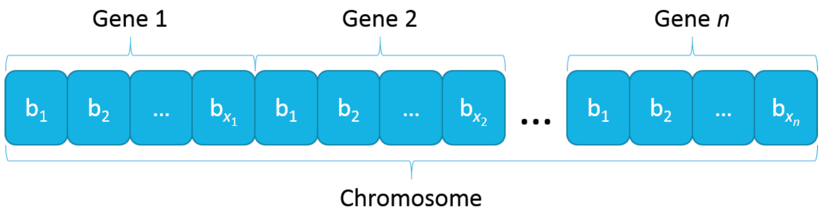

Fig. 1. The structure of a chromosome. The chromosome is comprised of a number of genes where each gene has its required number of bits.

There is a downside to genetic algorithms. Genetic algorithms have several parameters that must be tuned to each particular problem. These parameters are:

- The population size,
- Crossover probability,
- Mutation probability, and
- Number of successive failed generations before termination.

Unfortunately, tuning a genetic algorithm is more art than science and finding effective tuning parameters is essentially trial and error where their quality is determined by the rate of convergence and quality of solutions found when using them.

4. Architecture Evaluation

Ultimately, a solution must be evaluated. This was glossed over in the discussion of evolving the solution population but will be addressed here. For the purposes of optimization, evaluation used for comparing one solution to another in order to decide which one is more optimal (fitter in GA terminology). This is easy with single objective optimization, but with multiple objectives it is more complex. In order to compare two solutions with multiple objectives, the concept of Pareto dominance is usually employed. Assuming a minimization problem, a solution $x=(x_1,x_2,\ldots,x_n)$ is said to dominate a solution $y=(y_1,y_2,\ldots,y_n)$, written $x\prec y$, if, and only if, for all $1\leq i\leq n$, $x_i\leq y_i$ and for some $1\leq i\leq n$, $x_i<y_i$. Since a single solution generally cannot be expected to dominate all other solutions, an optimal solution is then considered to be any solution that is not dominated by any other solution. The set of non-dominated solutions is called the Pareto set.

The difficulty with the Pareto set is that it can be very large (effectively infinite), so it is not very useful without a method to identify a small subset of solutions of particular interest. Using a corner search method[14] (see Fig. 2), the corners of the Pareto set are identified. In this approach (assuming a minimization problem), corners are defined to be the solution with the minimum value for an objective. Hence, there is one corner for each objective. Another point of interest is the ideal point. The ideal point is not part of the Pareto set, but shows the direction in which the solutions would be more optimal. Using just the corners and the solution that is nearest to the ideal, the solution set can be reduced to a manageable size while still retaining information about the optimal solution set as a whole.

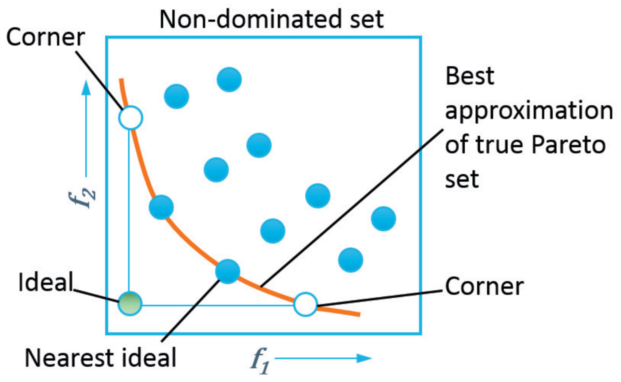

Fig. 2. The corner and ideal points identified in a Pareto (non-dominated) set.

5. The Network Centric Warfare Problem

The network centric warfare problem (see Fig. 3 for the OV-1) is an extension of the five node aerospace problem[15,16]. The five node aerospace problem makes a good basis with which to illustrate how this computational intelligence approach works. In this problem, there are five nodes: a ground station, a carrier, and three others chosen from two types of satellites and one type of UAV. Each type of system has an associated cost and measure of overall capability. There is also a matrix of values, α_{ij} and β_{ij}, that are used for Functional Dependency Network

Analysis (FDNA)[17]. FDNA is a measure of resilience in that it measures the degradation of each system due to the degradation of other systems in the network. The parameter α_{ij} represents the strength of dependency of system type j on system type i while β_{ij} represents the criticality of dependency of system type j on system type i.

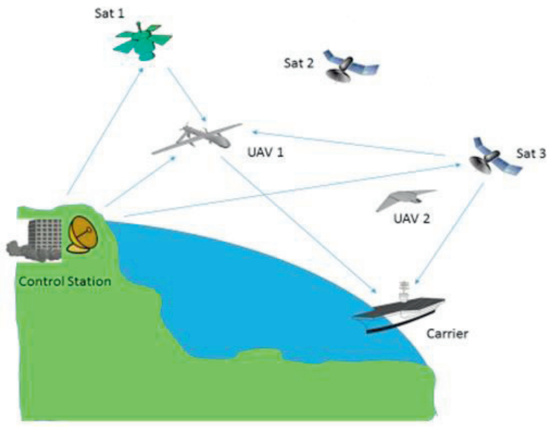

Fig. 3. High level operational view (OV-1) for the network centric warfare problem.

The network centric warfare problem is the same as the five aerospace node problem except that there are:

- Twenty-two nodes instead of five,
- Three types of satellites instead of two, and
- Two types of UAVs instead of one.

This means that there are twenty systems that must be chosen from five available options for each. Whereas the five node aerospace problem's optimum architecture is easy to obtain by a brute-force exhaustive search, the network centric warfare problem's solution is not amenable to exhaustive search. This is because the aerospace problem has only $(2+1)^{5-2} = 27$ possible solutions, but the network centric warfare problem has $(3+2)^{22-2} = 95{,}367{,}431{,}640{,}625$ possible solutions. In each case there are two fixed nodes (the aircraft carrier and the ground station) and the options available for the remaining nodes are the some of the available types of satellites and UAVs.

In this problem the ground station and aircraft carrier are fixed, but we are free to choose any of five types of systems for each of the other twenty nodes. Since a genetic algorithm is being employed as the optimizer, the meta-architecture should be a chromosome. The chromosome should represent all the feasible architectures but not any that are infeasible. Since twenty systems must be chosen, the chromosome should be comprised of twenty genes where each gene represents a system choice. And since there are five types of systems to choose from, each gene requires three bits. Since three bits represent eight unique values and only five are valid, modulo five should be applied to each gene to ensure all chromosome values represent feasible solutions. The resulting meta-architecture is shown in Fig. 4.

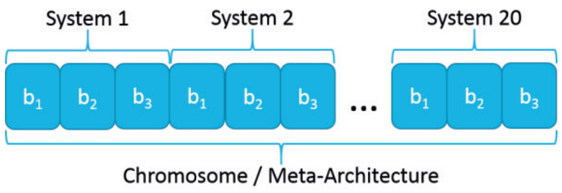
System 1
System 2
System 20
b1 b2 b3 b1 b2 b3 ... b1 b2 b3
Chromosome / Meta-Architecture

Fig. 4. The chromosome representing the network centric warfare problem meta-architecture.

In order to evaluate the architecture, three objectives are employed. Those objectives are cost, coverage, and resilience. Cost is to be minimized while coverage and resilience are to be maximized. The cost and coverage values used for the illustrating the approach are shown in Table 1. The resilience is a calculated value and is taken to be the value of the final receiver in the network, which in this case is the carrier. The values used for the strength of dependence, α, and the strength of criticality, β, are not shown because they must also be assigned for all possible interactions between the ground station and carrier as well as the five available system types, making for a rather unwieldy sized matrix in print. Because the problem must be strictly either a minimization or maximization problem, it will be treated as a minimization problem by subtracting the coverage and resilience values from their respective maximum values (in this case, 100).

Table 1. Cost and coverage area for each type of system.

System	Cost	Coverage area
Satellite type 1	90	80
Satellite type 2	100	90
Satellite type 3	100	100
UAV type 1	5	10
UAV type 2	10	20

6. Results

The approach developed here was implemented using MATLAB® and the comparison approach was implemented in MATLAB® with the Fuzzy Logic Toolbox™. The Genetic Algorithm was set to a crossover probability of 0.9, a mutation probability of 0.02, and to terminate after 10 unsuccessful generations when using the multiple objective method and 50 when using the fuzzy assessor method. In order to assess the resilience of a solution, each node is randomly degraded between 30% and 70% before the FDNA analysis is performed. Runs were performed with population sizes of 100, 250, 500, 1,000, 2,000, 4,000, 6,000, 8,000, and 10,000.

First, the total time required for each run and the average time required per generation is shown in Fig. 5. Because the number of generations can vary greatly depending on the given parameters, a more stable measure of time complexity is the time complexity of each generation. The time complexity for the fuzzy assessor method is much better, $O(n)$, than that of the multiple objective method, $O(n^3)$. However, as noted in the section on future work, there is opportunity for improvement.

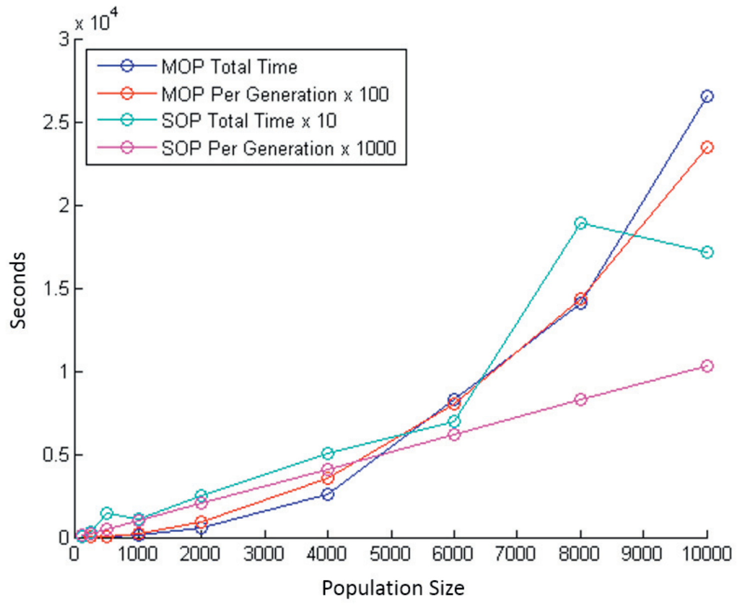

Fig. 5. Total time required for each run and the average time required per generation scaled for visibility.

Next, the Pareto set growth as a function of population and the Pareto set found with a population of 10,000 are shown in Fig. 6. The Pareto set size grows linearly with the population size and, for this algorithm, averaged about 60% of the population size.

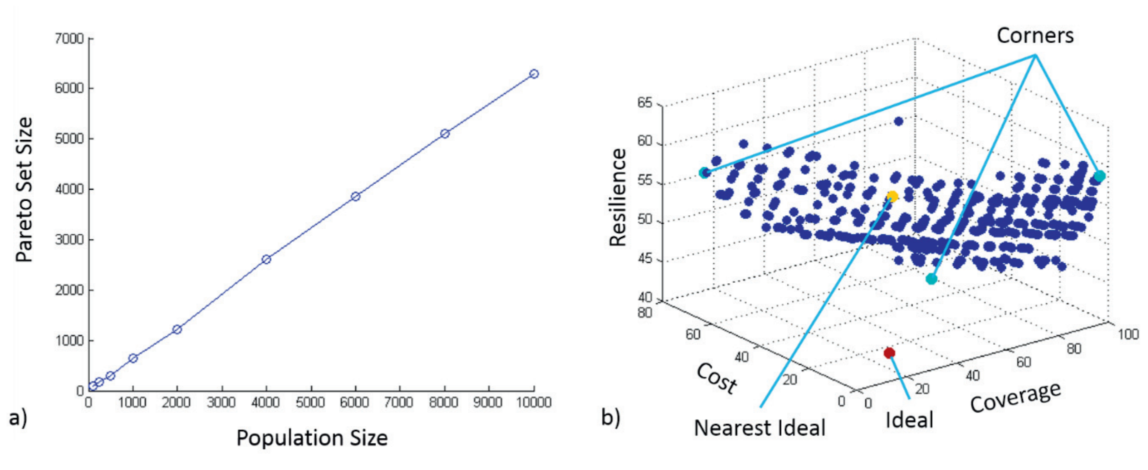

Fig. 6. (a) Pareto set size versus population size; (b) Pareto set found with a population of 10,000.

Next, the solutions found by each method are shown in Fig. 7. The solutions found by the fuzzy assessor method are perfectly in line with those found by the multiple objective method, however they are basically one point in the Pareto set and fail to yield as much information about the solution space as does the multiple objective method. The value of reducing the Pareto set is clearly seen in the uncluttered approximation to the true Pareto set compared to the mass of solutions seen in Fig. 6b.

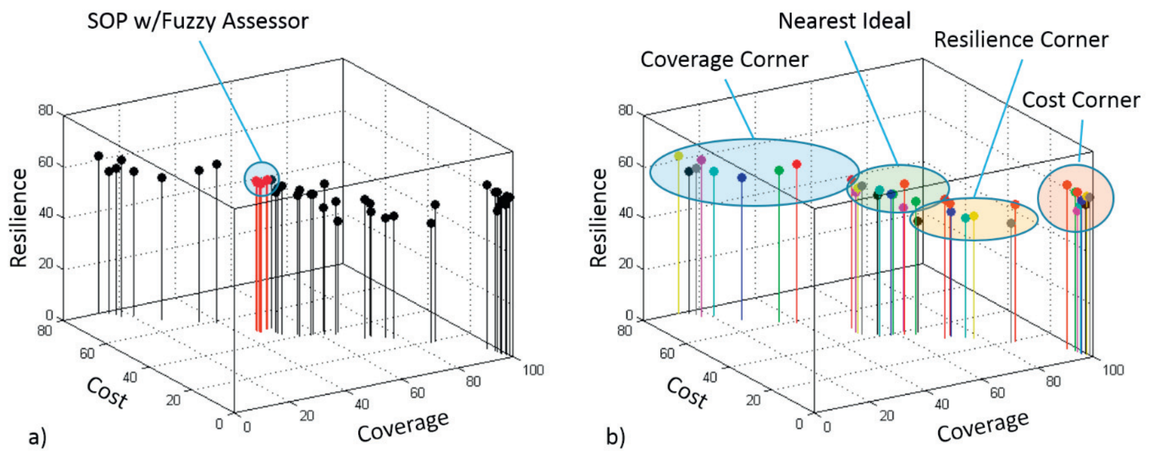

Fig. 7. (a) Multi-objective and fuzzy assessor method solutions for comparison; (b) Multi-objective method solutions broken-down.

7. Conclusion

The multiple objective computational intelligence approach demonstrated a significant improvement in the overall information available about the solution space. The solutions weren't better than those found by the fuzzy assessor method, but overall conveyed a picture that would allow an architect to better understand the optimal trade-space of a problem. The downside is that the multiple objective approach is significantly slower than the fuzzy assessor approach.

8. Future Work

This is a bare-bones approach incorporating multiple objective optimization and there is significant room for improvement. A few areas include:

- Use of potentially faster and more effective algorithms such as MOGA[10], NSGA-II[11], or SPEA2[12],
- Use of many-objective optimization algorithms such as Corner Sort[18], and
- Improving the information extracted from the Pareto set.

References

1. M. Jamshidi, Editor, System of Systems Engineering - Innovations for the 21st Century, John Wiley & Sons, 2009, pp. 1–2.
2. W. H. Manthorpe, The Emerging Joint System of Systems: A Systems Engineering Challenge and Opportunity for APL, John Hopkins APL Technical Digest, 17(3):305–310, 1996.
3. A. P. Engelbrecht, Computational Intelligence: An Introduction, 2nd Edition, Wiley Publishing, 2007, p. 4.
4. L. A. Zadeh, Fuzzy Sets, Information and Control, 8:338–353, 1965.
5. L. A. Zadeh, Soft Computing and Fuzzy Logic, IEEE Software, 11(6):48–56, 1994.
6. L. Pape, K. Giammarco, J. Colombi, C. Dagli, N. Kilicay-Ergin, G. Rebovich, A Fuzzy Evaluation method for System of Systems Meta-architectures, Procedia Computer Science 16 (0) (2014) 245–254, 2013 Conference on Systems Engineering Research. doi:http://dx.doi.org/10.1016/j.procs.2013.01.026. URL http://www.sciencedirect.com/science/article/pii/S1877050913000276.
7. A. S. Fraser, Simulation of Genetic Systems by Automatic digital Computers I: Introduction, Australian Journal of Biological Science, 10:484–491, 1957.
8. A. S. Fraser, Simulation of Genetic Systems by Automatic digital Computers II: Effects of Linkage on Rates of Advance Under Selection, Australian Journal of Biological Science, 10:492–499, 1957.
9. J. H. Holland, Adaptation in Natural and Artificial Systems, University of Michigan Press, 1975.

10. C. M. Fonseca and P. J. Fleming, Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization, in Proc. 5th Int. Conf. Genet. Algorithms, 1993, pp. 416–423.
11. K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Trans. Evol. Comput., vol. 6, no. 2, pp. 182–197, Apr. 2002.
12. E. Zitzler, M. Laumanns, and L. Thiele, SPEA2: Improving the strength Pareto evolutionary algorithm, in Proc. Evol. Methods Des., Optimization Control Applicat. Ind. Problems (EUROGEN), 2002, pp. 95–100.
13. A. P. Engelbrecht, Computational Intelligence: An Introduction, 2nd Edition, Wiley Publishing, 2007, pp. 143–144.
14. H. Singh, A. Isaacs, and T. Ray, A Pareto corner search evolutionary algorithm and dimensionality reduction in many-objective optimization problems, IEEE Trans. Evolut. Comput., vol. 15, no. 4, pp. 539–556, Aug. 2011.
15. N. Davendralingam, D. DeLaurentis, Z. Fang, C. Guariniello, S. Y. Han, K. Marais, A. Mour, P. Uday, An Analytic Workbench Perspective to Evolution of System of Systems Architectures, Procedia Computer Science 28 (0) (2014) 702–710, 2014 Conference on Systems Engineering Research. doi:http://dx.doi.org/10.1016/j.procs.2014.03.084. URL http://www.sciencedirect.com/science/article/pii/S1877050914001471.
16. C. Guariniello, D. DeLaurentis, Communications, Information, and Cyber Security in Systems-of-Systems: Assessing the Impact of Attacks through Interdependency Analysis, Procedia Computer Science 28 (0) (2014) 720–727, 2014 Conference on Systems Engineering Research. doi:http://dx.doi.org/10.1016/j.procs.2014.03.086. URL http://www.sciencedirect.com/science/article/pii/S1877050914001495.
17. P. R. Garvey, C. A. Pinto, Introduction to Functional Dependency Network Analysis, The MITRE Corporation, 2009.
18. H. Wang, X. Yao, Corner sort for Pareto-based many-objective optimization, IEEE Trans. Cybernetics, vol. 44, no. 1, pp. 92–102, Jan. 2014.