Multi-objective Optimization for Series–parallel Systems using Genetic Algorithm

Kriangkrai Chaonithi (57070701702)
CPE614 Optimization Design and Reliability Engineering
King Mongkut's University of Technology Thonburi

Project Description and Scope

This project aims to implement genetic algorithm (GA) to optimize reliability, cost, and weight of series-parallel systems based on the input data from table 1 in input data section. Each subsystem has different number of component and at least one component must be chosen to make the system functional. This project use weighted sum approach to calculate fitness value of the system which provides 3 weight parameters to adjust for the system reliability, cost and weight.

Programming Challenge

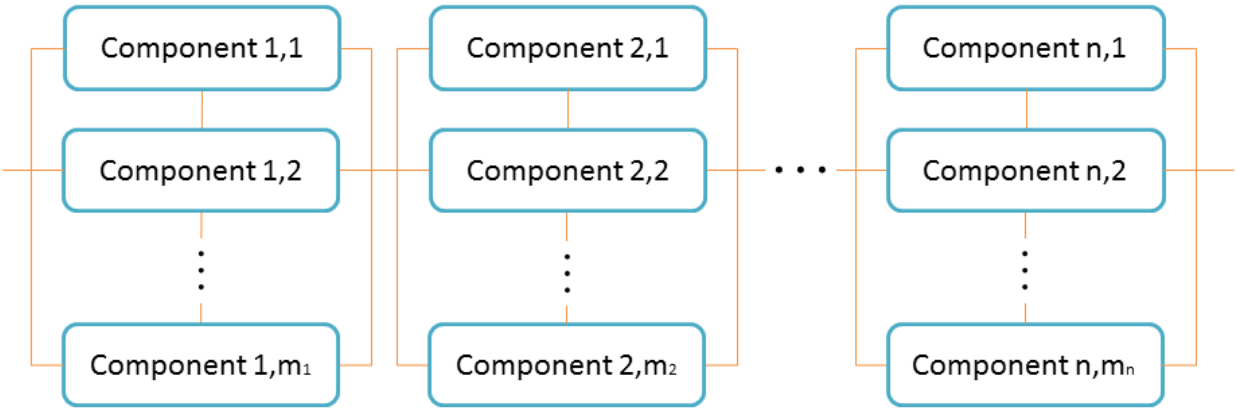
The problem space of this project is large and varies on weight parameters which cause GA to be easily trapped in local optimum. The algorithm must be able to increase population diversity of to keep exploring the problem space.

Assumptions

- The possible states of each component are functional or fail.
- Reliability, cost, and weight of each component are given and fixed.
- A component can be selected only once for the whole system.
- No repairing if a component fails.
- Failure of a component is independent from other components

Model Formulation

System Model



Notation

Z	Fitness of a system
X_{ij}	Architecture X for component j at subsystem i
n	Number of subsystem within the system (n = &0
m_i	Number of component choices available for subsystem i (m_i = 4,3,4,3,4,4,3)
R_i	Estimated reliability of the subsystem i
R_{ij}	Estimated reliability of using component j at subsystem i
C_i	Normalized cost of subsystem i
C_{ij}	Normalized cost of using component j at subsystem i
W_i	Normalized weight of subsystem i
W_{ij}	Normalized weight of using component j at subsystem i
α	Fitness weight of system reliability
β	Fitness weight of system cost
γ	Fitness weight of system weight

Objective Function

$$Max\ Z = \alpha \prod_{i=1}^{n} R_i + \beta \left(1 - \sum_{i=1}^{n} C_i\right) - \gamma \left(1 - \sum_{i=1}^{n} W_i\right)$$

Where

$$R_i = 1 - \prod_{j=1}^{m_i} \left(1 - X_{ij} R_{ij}\right)$$

$$C_i = \sum_{j=1}^{m_i} X_{ij} C_{ij}$$

$$W_i = \sum_{j=1}^{m_i} X_{ij} W_{ij}$$

Constrains

System must be functional

$$\prod_{i=1}^{n} R_i > 0$$

Sum of the fitness weight must equal to 1

$$\alpha + \beta + \gamma = 1$$

The number of selected components must be in between 1 and m_i

$$\sum_{j=1}^{m_i} X_{ij} = 1,2, \dots, m_i$$

$$X_{ij} = 0,1$$

$$i = 1,2, \dots, n$$

Input Data

Configurable Parameters

- Fitness weight of system reliability (α)
- Fitness weight of system cost (β)
- Fitness weight of system weight (γ)

Component Input Data

Note: Comp = component, R = reliability, C=cost, W=weight,
the symbol "-" means that a design alternative is not available.

Table 1 input data consists of design choices, component reliability, cost, and weight. [3]

Sub-system	Design Alternative											
	Comp Choice 1			Comp Choice 2			Comp Choice 3			Comp Choice 4		
	R	C	W	R	C	W	R	C	W	R	C	W
1	0.90	1	3	0.93	1	4	0.91	2	2	0.95	4	5
2	0.95	4	8	0.94	2	10	0.93	1	9	-	-	-
3	0.85	2	7	0.95	3	5	0.87	1	6	0.92	4	4
4	0.83	3	5	0.87	4	6	0.85	5	4	-	-	-
5	0.94	2	4	0.93	2	3	0.95	5	5	0.94	2	4
6	0.99	6	5	0.98	4	4	0.97	2	5	0.96	2	4
7	0.91	4	7	0.92	4	8	0.94	5	0	-	-	-

In this experiment, there are 5 sets of weight parameters which are
(α, β, γ) = {(0.2, 0.4, 0.4), (0.5, 0.3, 0.2), (0.5, 0.2, 0.3), (0.8, 0.1, 0.1), (1.0, 0.0, 0.0)}.

Problem Size

The problem size is form as equation below

$$Problem\ size = \prod_{i=1}^{n} \sum_{j=1}^{m_i} \binom{m_i}{j}$$

For the input data we used, the problem size is

$$Problem\ size = \prod_{i=1}^{7} \sum_{j=1}^{m_i} \binom{m_i}{j}$$

$$Problem\ size = \left[\binom{4}{1} + \binom{4}{2} + \binom{4}{3} + \binom{4}{4}\right]^4 \times \left[\binom{3}{1} + \binom{3}{2} + \binom{3}{3}\right]^3$$

$$Problem\ size = 50{,}625 \times 343$$

$$Problem\ size = 17{,}364{,}375$$

Algorithm and Parameter Setting

This project combines of two main approaches, weighted sum and genetic algorithm. This section will describe the algorithm and parameter setting in detail.

Weighted Sum

Weighted sum is a simple and most widely used method to solve multi-objective problems. The method simply weights the value of each objective and sums them up as a main objective value. Summation of weight parameters must equal to 1 and the objective value to be weight must be normalized to the range of 0 and 1, the cost and weight values are divided by total cost and weight respectively.

In this project, there are 3 objectives which are system reliability (α), costs (β), and weight (γ). When the weights changed, the result of the system also changed as shown in Figure 1. The higher weight means the more concern of the objective. The values of system cost and weight are better when lower. Therefore, the value of β is high means the low of system cost and γ means the same for system weight.

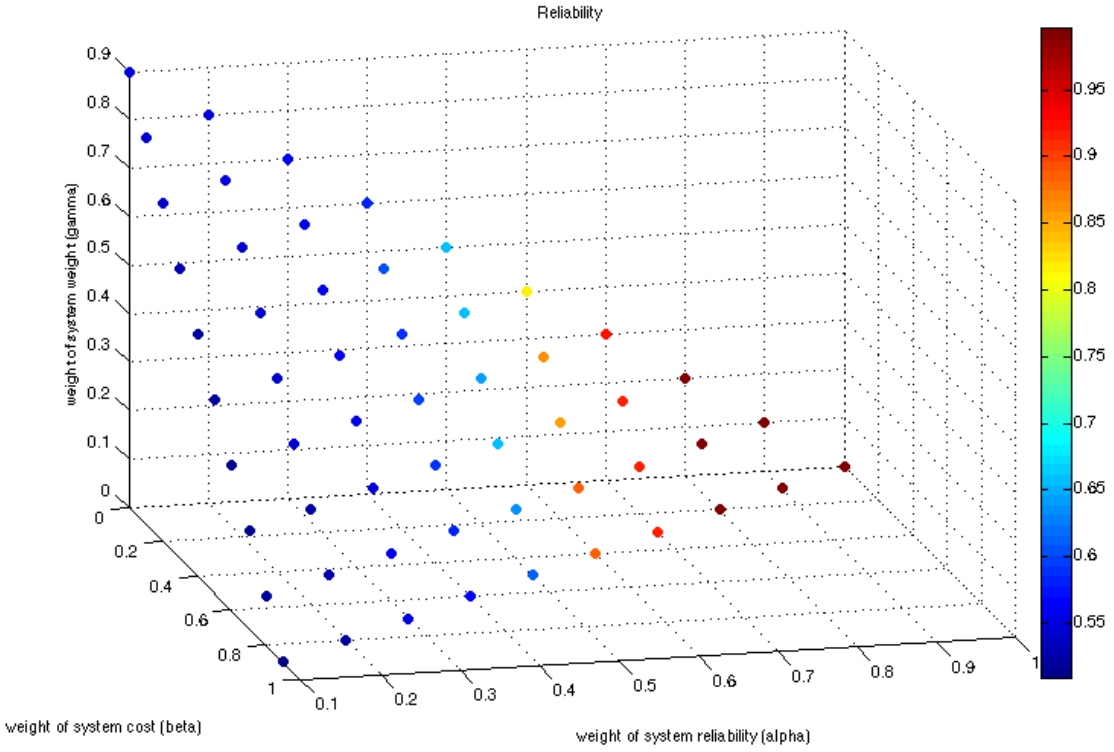


Figure 1 system reliability changes because of weights change

Genetic Algorithm

This project uses genetic algorithm to search for the optimum point of the weighted objectives. In the experiment, the string encoding is a set of bits that expresses the selected components, one bit per one component which results in 25 bits for one chromosome. The maximum number of generation is set to 5,000 and the population size is 50. Top 3 fittest chromosomes are passed to the next generation for elitism. Tournament selection has been chosen for selecting parents for crossing over. The tournament size is 10 randomly pick from the population and select the first two fittest chromosomes. The crossing over rate is set at 80%, and one-point crossover is used in the experiment. The mutation rate is 20% which allow 5-6 bits to be mutated for a single chromosome.

A subsystem must have at least one component, so repairing algorithm is implemented. When the genes are changed, it will be checked if any subsystem has zero components. The algorithm will randomly select a component to fill a blank subsystem.

In order to improve exploration, 10 of the worst offspring are reinitialized in each evolution.

As parameters setting described above, when the values of weights (α, β, γ) is (0.2, 0.4, 0.4), we obtain the step of fitness value of the algorithm shown in Figure 2.

This project is developed based on creating a genetic algorithm for beginners [2], and the final code is openly available on the author's GitHub [1]

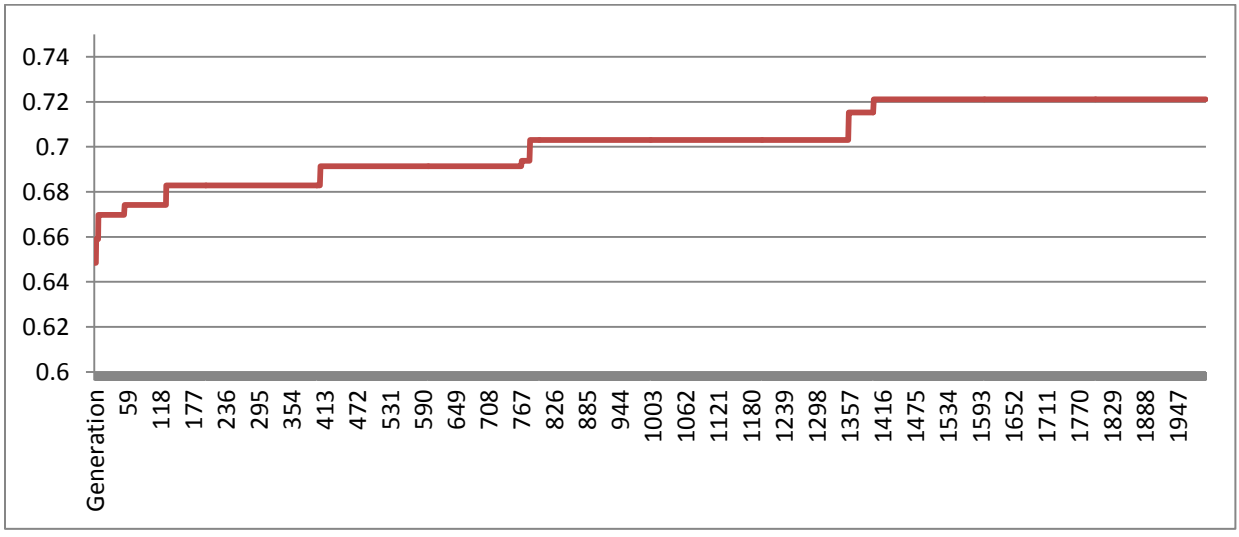


Figure 2 Example of Fitness Value Convergence Graph when at (α, β, γ) = (0.2, 0.4, 0.4)

Verification

The optimum solutions of each set of weight parameters are calculated by using brute force approach, the result and comparison are shown in the next section.

Result and Discussion

The selected result of genetic algorithm and brute force approach are shown in Table 2 and Table 3 respectively. The values of system reliability, cost, and weight are varying by the values of α, β, and γ as described before. CPU times of genetic algorithm are depended on the number of generation and the frequency in need of repairing method. On the other hand, brute force approach, which guaranteed the optimum solution, uses large amount of computing time compare to the genetic algorithm.

Table 2 result of genetic algorithm

α	β	γ	Fitness	Reliability	Cost	Weight	Generation	CPU Time (sec)
0.2	0.4	0.4	0.7108	0.524907	16	37	1009	0.5555
0.5	0.3	0.2	0.6990	0.835923	32	63	2933	0.8449
0.5	0.2	0.3	0.6973	0.782103	29	52	1724	0.4965
0.8	0.1	0.1	0.8411	0.986849	55	102	3179	0.1185
1.0	0.0	0.0	0.9958	0.995846	75	136	10	0.0457

Table 3 result of brute force approach

α	β	γ	Fitness	Reliability	Cost	Weight	CPU Time (sec)
0.2	0.4	0.4	0.7151	0.50741	14	38	89.1709
0.5	0.3	0.2	0.7159	0.915248	30	75	87.8117
0.5	0.2	0.3	0.7273	0.879627	36	58	86.5529
0.8	0.1	0.1	0.8603	0.969234	40	84	84.3733
1.0	0.0	0.0	0.9958	0.995846	75	136	83.2641

Comparing the fitness values of genetic algorithm to brute force in Figure 3, there are some differences when the weight parameters are closed to each other. It is because there are more similar options there, and the GA is trapped in local optima as a result. Even tough, the result is acceptable especially when we concern about CPU time. The algorithm can search for the solution almost instantly, with in a second. When the weight of system reliability (α) is set to 1, which is to enable all components in the system, GA can find the optimum solution easily.

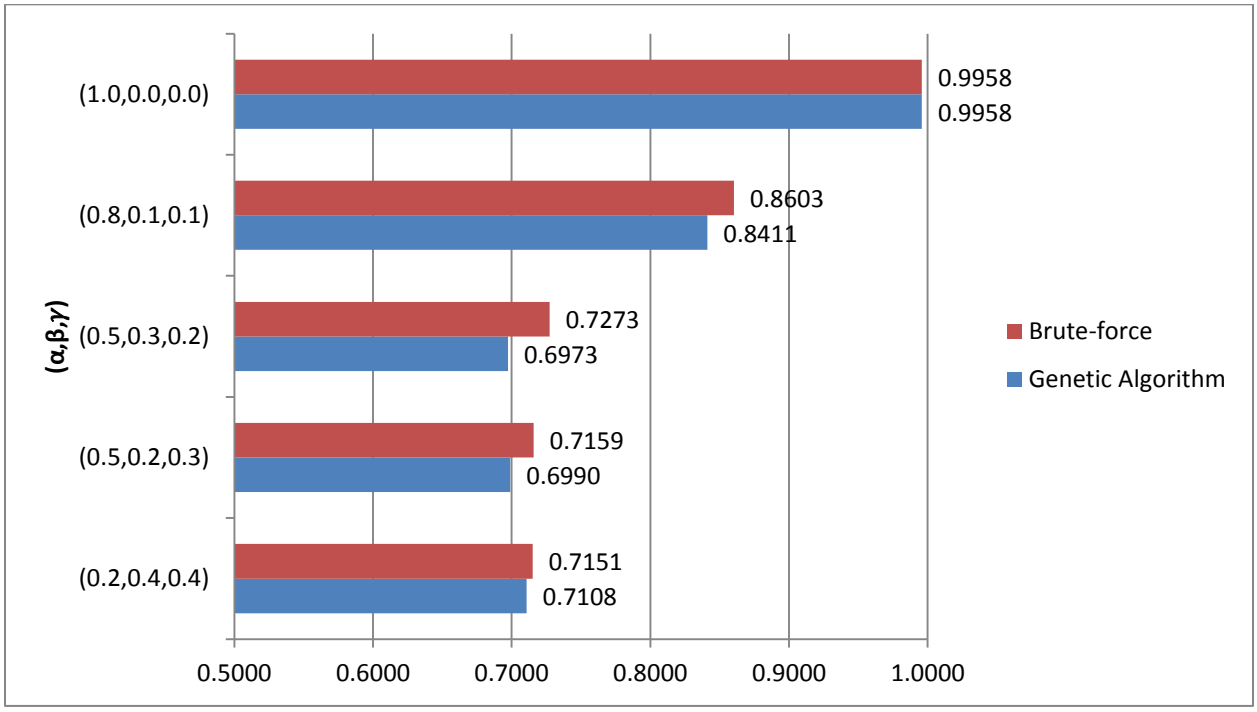


Figure 3 result comparison between GA and brute force

Conclusion

Genetic algorithm with weighted sum can be used to solve multi-objective optimization for series-parallel systems. The experiment shows the acceptable results of using GA which are quite close to the optimum solutions when using brute force approach. On the other hand, GA uses significantly less processing time which makes it more appropriate to apply for real world problem.

This work can be improved further by advance the mechanism to increase exploration of the genetic algorithm. This, as a result, will prevent GA from trapping in local optimum.

Reference

[1] Chaonithi, K. (2014, November 24). CPE614-Component-Allocation-GA. Retrieved (cfc9fffb64) from https://github.com/spicydog/CPE614-Component-Allocation-GA/tree/series-palallel

[2] Jacobson, L. (2012, February 12). Creating a genetic algorithm for beginners. Retrieved from http://www.theprojectspot.com/tutorial-post/creating-a-genetic-algorithm-for-beginners/3

[3] Wattanapongsakorn, N. Input data consists of design choices, component reliability, cost, and weight.