

# Multi-objective Optimization for Series-parallel Systems Using Genetic Algorithm

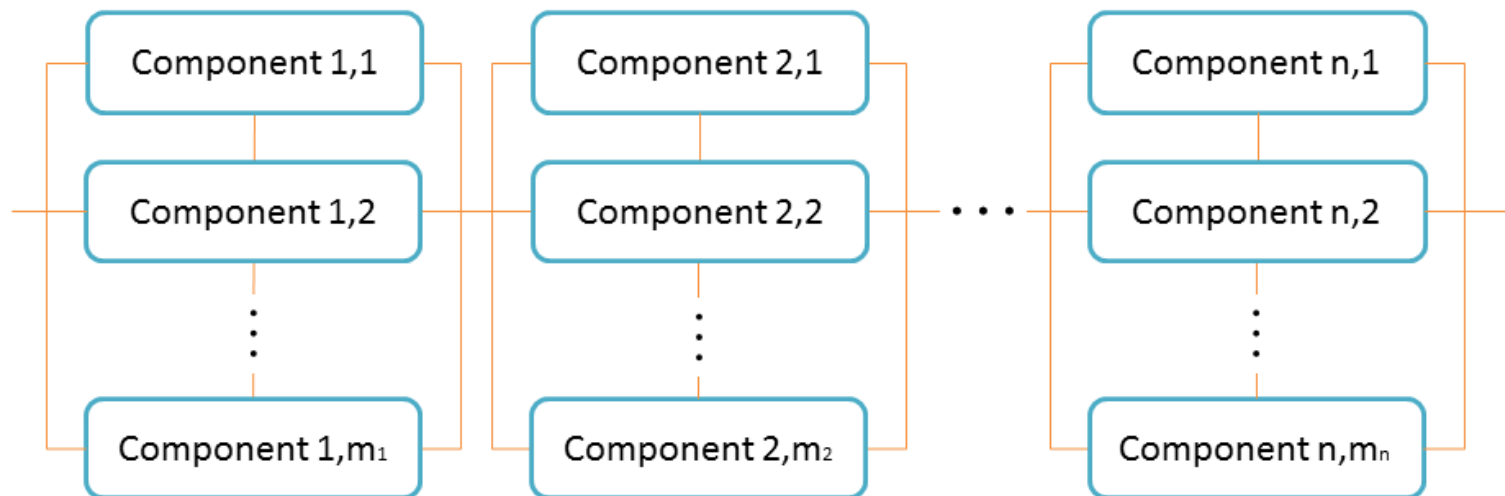
Kriangkrai Chaonithi

57070701702

King Mongkut's University of Technology Thonburi

# Project Description and Scope

- Implement genetic algorithm to optimize system reliability, weight, and cost of series-parallel systems using weighted sum.



# Programming Challenge

- The algorithm must be able to increase population diversity of to keep exploring the problem space

# Assumptions

- A component can only be functional or fail
- System reliability, cost, and weight of each component are fixed and provided
- 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

$Z$	Fitness of a system
$X_{ij}$	Architecture $X$ for component $j$ at subsystem $i$
$n$	Number of subsystem within the system ( $n = \infty$ )
$m_i$	Number of component choices available for subsystem $i$
$R_i$	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 $i$
$\alpha$	Fitness weight of system reliability
$\beta$	Fitness weight of system cost
$\gamma$	Fitness weight of system weight

# Objective Function

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

Where

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

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

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

# 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 ( $\alpha$ )
- Fitness weight of system cost ( $\beta$ )
- Fitness weight of system weight ( $\gamma$ )

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	-	-	-



# Problem Size

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

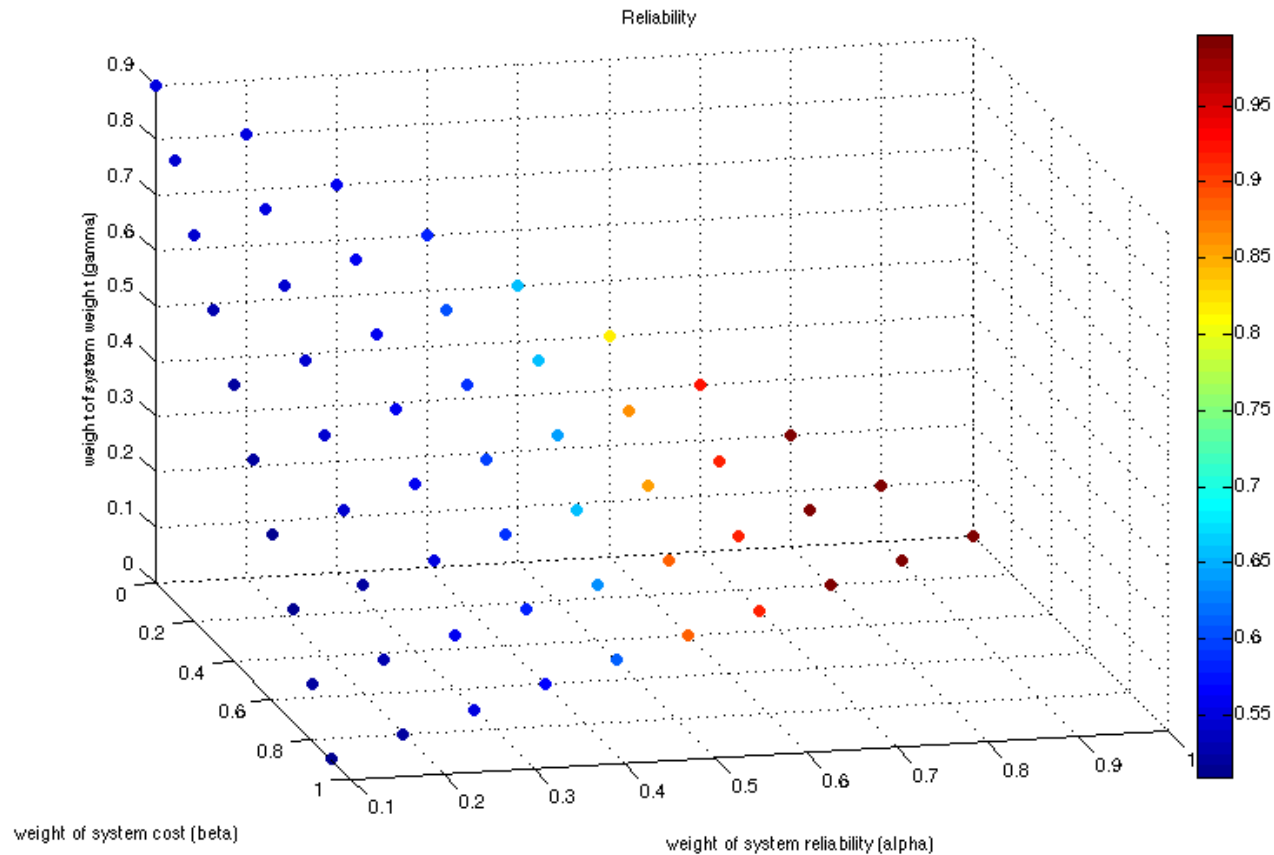
$$\textit{Problem size} = \left[ \binom{4}{1} + \binom{4}{2} + \binom{4}{3} + \binom{4}{4} \right]^4 \times \left[ \binom{3}{1} + \binom{3}{2} + \binom{3}{3} \right]^3$$

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

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

# Weighted Sum

Sum of the fitness weight must equal to 1:  $\alpha + \beta + \gamma = 1$



# Genetic Algorithm

- Max Generation Run: 5,000
- Population size: 50
- Chromosome size: 25 (1 gene for 1 component )
- Crossover Method: Tournament Selection
- Crossover Rate: 80%
- Mutation Rate: 20%

# Chromosome Repairing Algorithm

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

- A subsystem with no component will be randomly inserted a component
- The repairing function will execute after each evolution phase

# Population Elimination

- Prevent algorithm from local optimum
- Improve population diversity
- Select 10 top worst individuals, and regenerate them



# Execution Result

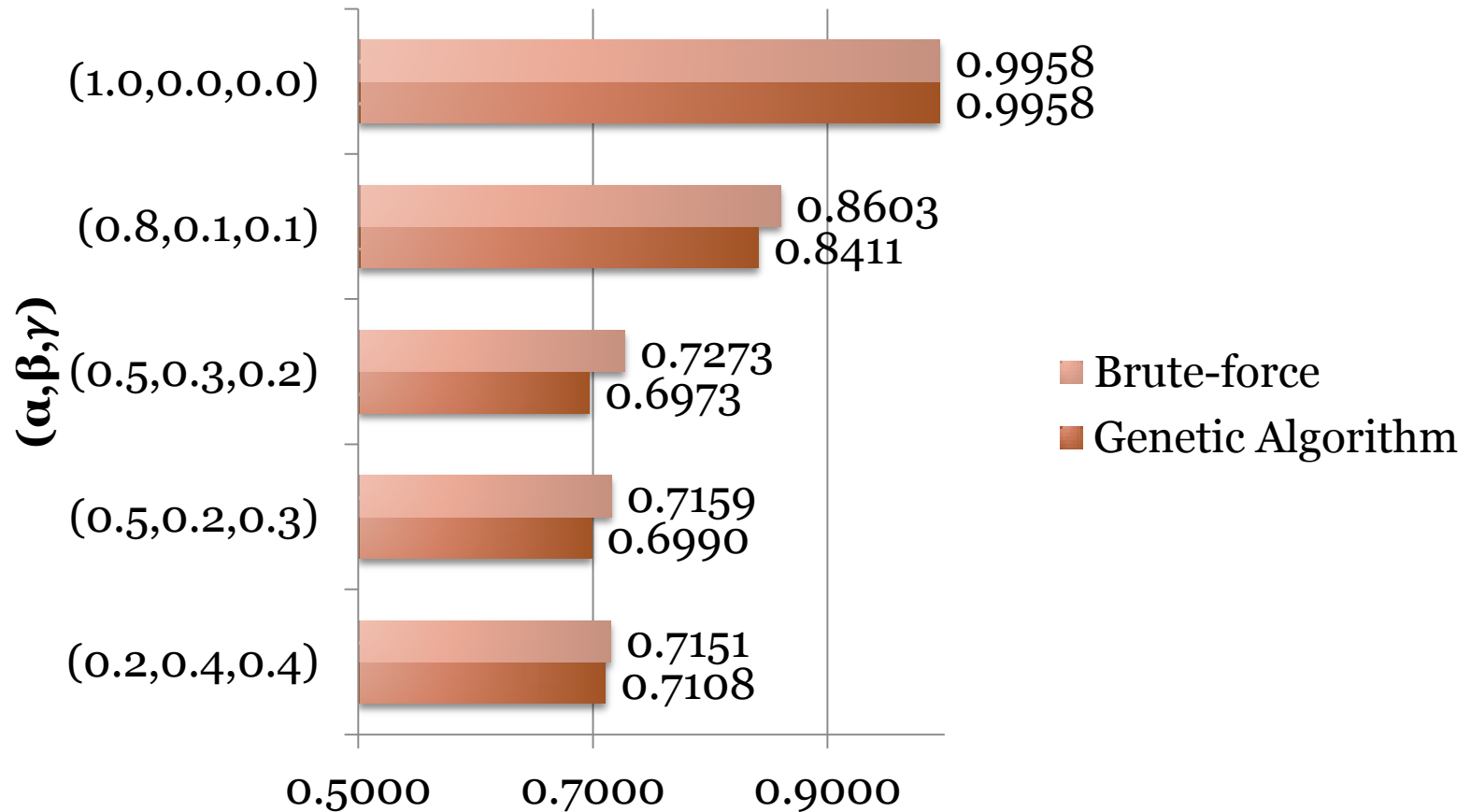
- Genetic Algorithm

$\alpha$	$\beta$	$\gamma$	Fitness	Reliability	Cost	Weight	CPU Time (sec)
<b>0.2</b>	0.4	0.4	0.7108	0.5249	16	37	0.5555
<b>0.5</b>	0.3	0.2	0.6990	0.8359	32	63	0.8449
<b>0.5</b>	0.2	0.3	0.6973	0.7821	29	52	0.4965
<b>0.8</b>	0.1	0.1	0.8411	0.9868	55	102	0.1185
<b>1.0</b>	0.0	0.0	0.9958	0.9958	75	136	0.0457

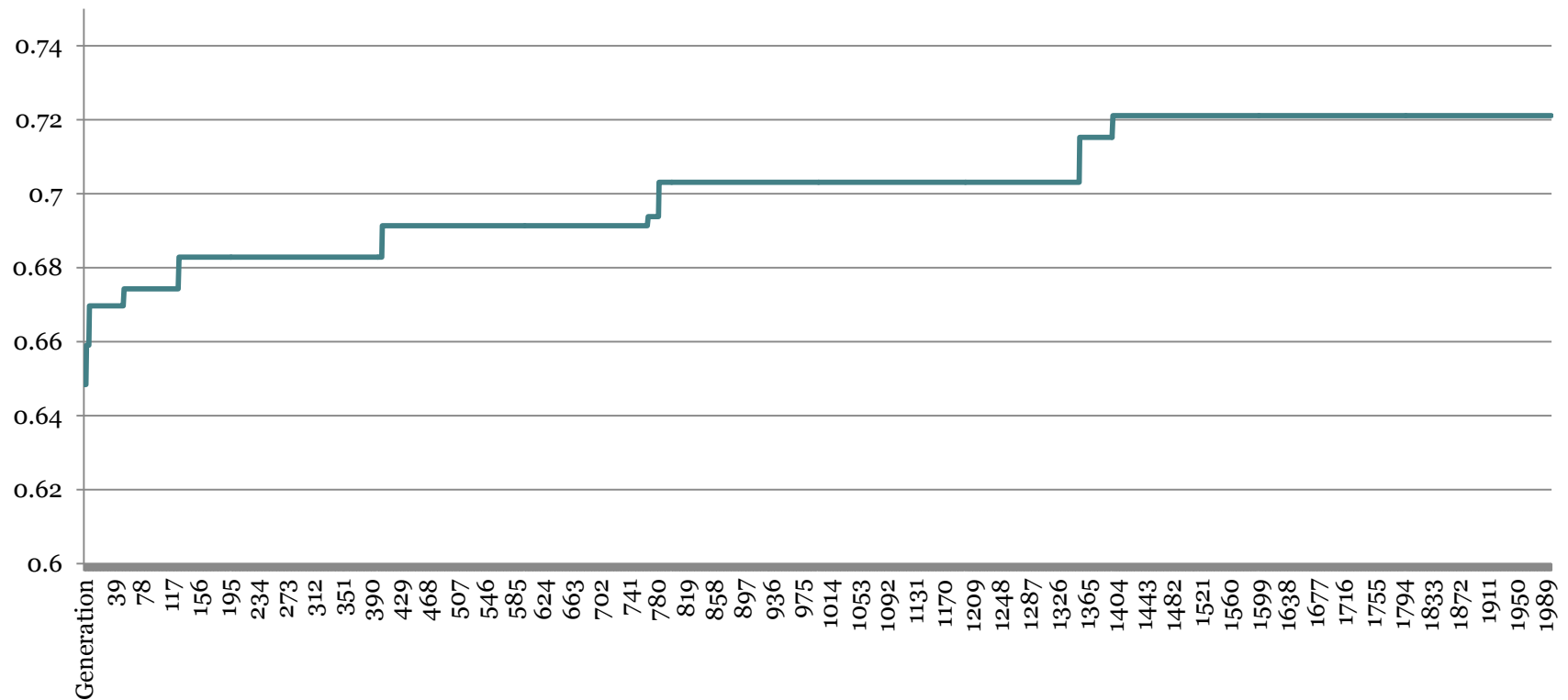
- Brute Force

$\alpha$	$\beta$	$\gamma$	Fitness	Reliability	Cost	Weight	CPU Time (sec)
<b>0.2</b>	0.4	0.4	0.7151	0.507	14	38	89.1709
<b>0.5</b>	0.3	0.2	0.7159	0.9152	30	75	87.8117
<b>0.5</b>	0.2	0.3	0.7273	0.8796	36	58	86.5529
<b>0.8</b>	0.1	0.1	0.8603	0.9692	40	84	84.3733
<b>1.0</b>	0.0	0.0	0.9958	0.9958	75	136	83.2641

# Result Comparison



# GA Convergence Graph





# Conclusion

- GA can return acceptable results
- GA uses very small amount of CPU time
- Due to the enormous size of problem space, GA stuck in local optimum very often

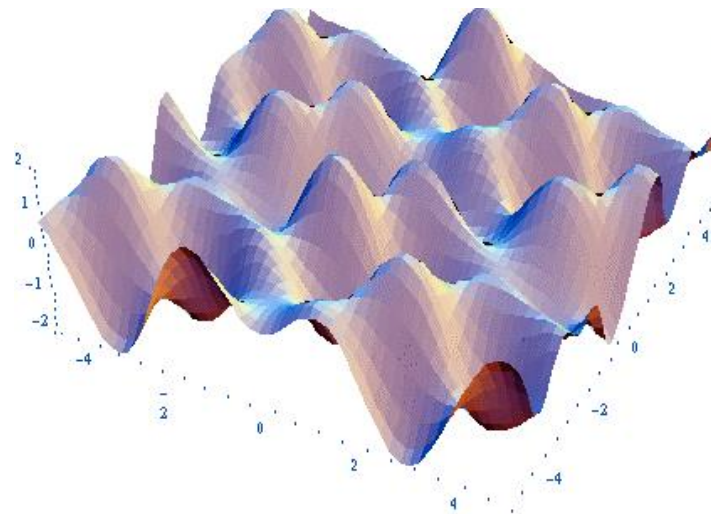


Image from: <http://plato.asu.edu/gom.html>

# Further Improvement

- Advance GA mechanism to increase exploration



Image from: <https://secure.flickr.com/photos/lakelandlocal/404688085/>

# Reference

- Chaonithi, K. (2014, November 24). CPE614ComponentAllocation-GA. Retrieved (cfc9fffb64) from <https://github.com/spicydog/CPE614-Component-Allocation-GA/tree/series-parallel>
- 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>
- Input data from Dr Naruemon Wattanapongsakorn

# Thank you

Q&A