Budgetary and Redundancy Optimisation of Homogeneous Series-Parallel Systems Subject to Availability Constraints Using Matlab Implemented Genetic Computing

Walid Chaaban[1], Michael Schwarz[2], and Josef Börcsöck[3]

Department of Computer Architecture and System Programming

University of Kassel, Kassel, Germany

E-mail: [1]walid.chaaban@uni-kassel.de

Abstract – This paper deals with finding optimal structures and redundant safety designs with minimal costs for series-Parallel multi state system (MSS) configurations subject to availability constraints using genetic algorithms as optimisation mean, since these biologically inspired evolution concepts showed stability, powerfulness, and effectiveness in solving such complex combinatorial optimisation tasks. The routine has been written in Matlab and the tests have been performed using some test data belonging to already existing models (Levitin, Lisnianski, and Ouzineb).

Keywords – Genetic Algorithms, Series-Parallel Systems, Optimisation, Redundancy Allocation Problem (RAP), Multi State System (MSS), Universal Moment Generating Function (UMGF).

I INTRODUCTION

Solving redundancy problems and finding optimal reliability/availability designs with minimized budgetary of homogeneous series-parallel Multi States Systems (MSS) subject to some predefined constraints, like availability, weight, and volume represent one of the most challenging tasks in reliability and safety engineering. The main purpose is to achieve a specific level of reliability desired according to predefined requirement specifications set for the intended end product. This Problem is referred to in the literature as the Redundancy Allocation Problem (RAP) or Redundancy Optimisation Problem (ROP) [1] and can be often encountered in many applications areas of the safety engineering world like electrical power systems and in the consumer electronic industry where system designs are mostly assembled using standard certified component types with known characteristics, e.g., reliability, availability, nominal performance, cost, etc.. This matter has been intensively studied over the years and has been classified as a complex nonlinear integer programming combinatorial problem, where deterministic or conventional mathematical optimisation approaches become ineffective by means of computational effort and quality of solution [1]. Metaheuristic optimisation techniques, e.g., Genetic Algorithms (GAs), Tabu Search (TS), Simulated Annealing, etc., have shown instead how powerful and effective they are as means to find high qualitative solutions for such problems, especially when the corresponding search or solution space of the formulated problem becomes larger. The rest of the paper is organised as follows. Section II gives a short mathematical demonstration about the advantages of the series-parallel configuration over the parallel-series configuration. Section III discusses genetic algorithms and the different belonging operators. Section IV presents a detailed formulation of the optimization problem discussed in this paper and which is going to be solved using heuristic genetic techniques. Section V deals with the calculation of the total availability of the different redundancy structures using the UMGF (Universal Moment Generating Function). Section VI shows different results, evaluations, and graphical representations obtained by the algorithm for the different treated models which are compared with existing evaluations while concluding remarks are made in section VII.

II WHY SERIES-PARALLEL SYSTEMS

There have been various publications over the last decade dealing with the optimisation of redundant multi state systems. The big part of these publications is concerned with the optimisation of series-parallel systems represented Fig.1. In the following, it will be explained mathematically and physically why studying or handling especially this arrangement or structure while many other configurations exist.

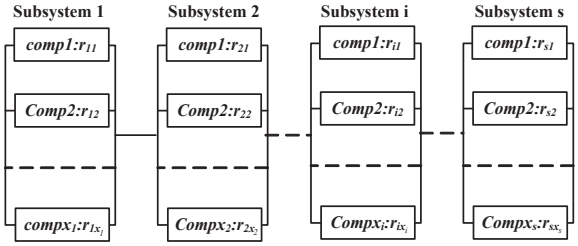

Fig. 1: series-parallel system consisting of s-subsystems

In Fig. 1 r_{ij} represents the reliability of a component of version j within the subsystem i. A short mathematical computation and explanation will show why series-parallel configurations are suitable and why they have been treated all the time. One Brief explanation is mentioned in [1]

Consider a series-parallel configuration consisting of 4 components that are connected according to Fig. 2.

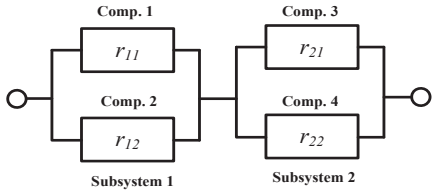

Fig. 2: Series-parallel configuration

The total reliability R_{sp} of the series-parallel configuration shown in Fig.2 is given by [1][2][3]

$$\begin{aligned} R_{sp} &= \prod_{i=1}^{s=2}[1-\prod_{j=1}^{x_i=2}(1-r_{ij})] \\ &= [1-(1-r_{11})(1-r_{12})][1-(1-r_{21})(1-r_{22})] \\ &= r_{11}r_{21}+r_{11}r_{22}-r_{11}r_{21}r_{22}+r_{21}r_{12}+r_{12}r_{22}-r_{21}r_{12}r_{22} \\ &\quad -r_{11}r_{21}r_{12}-r_{11}r_{12}r_{22}+r_{11}r_{12}r_{21}r_{22} \end{aligned} \quad (1)$$

The corresponding parallel-series configuration of the series-parallel configuration of Fig. 2 using the same set of components can be represented in two different ways depending on how the components will be connected together. The 2 different arrangements are depicted in Fig. 3(a) and 3(b).

The reliability of a parallel-series system is given by

$$R_{ps} = 1-\prod_{i=1}^{s=2}(1-\prod_{j=1}^{x_i=2}r_{ij}) \quad (2)$$

Based on equation (2) the corresponding reliabilities R_{ps_a} and R_{ps_b} of the parallel-series configurations depicted in Fig. 3(a) and Fig. 3 (b) are respectively given by [1][2][3]

$$\begin{aligned} R_{ps\_a} &= 1-(1-r_{11}r_{21})(1-r_{12}r_{22}) \\ &= r_{11}r_{21}+r_{12}r_{22}-r_{11}r_{12}r_{21}r_{22} \end{aligned} \quad (3)$$

and

$$\begin{aligned} R_{ps\_b} &= 1-(1-r_{11}r_{22})(1-r_{12}r_{21}) \\ &= r_{11}r_{22}+r_{12}r_{21}-r_{11}r_{12}r_{21}r_{22} \end{aligned} \quad (4)$$

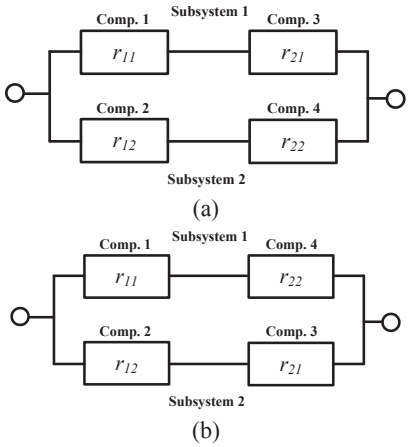

Fig.3: Parallel-series configurations

Calculating the difference of reliabilities obtained from equation (1) and equation (3) will give respectively for R_{sp} - R_{ps_a}

$$\begin{aligned} R_{sp}-R_{ps\_a} &= r_{11}r_{22}(1-r_{12}-r_{21}+r_{21}r_{12}) \\ &\quad +r_{21}r_{12}(1-r_{11}-r_{22}+r_{11}r_{22}) \\ &= r_{11}r_{22}(1-r_{21})(1-r_{12})+r_{21}r_{12}(1-r_{11})(1-r_{22}) \end{aligned} \quad (5)$$

and for R_{sp} - R_{ps_b}

$$\begin{aligned} R_{sp}-R_{ps\_b} &= r_{11}r_{21}(1-r_{12}-r_{22}+r_{12}r_{22}) \\ &\quad +r_{12}r_{22}(1-r_{11}-r_{21}+r_{11}r_{21}) \\ &= r_{11}r_{21}(1-r_{12})(1-r_{22})+r_{21}r_{22}(1-r_{11})(1-r_{21}) \end{aligned} \quad (6)$$

The results obtained from equations (5) and (6) are in both cases positive values since all reliabilities r_{ij} are decimal numbers between 0 and 1. In other words let us consider the systems depicted in Fig. 2 and Fig. 3(a). If the components 1 and 4 or 2 and 3 fail, the series-parallel system of Fig. 2 remains working since failing components are recovered by redundant components 2 and 3 or 1 and 4, whereas in the parallel-series configuration of Fig. 3(a) a total system failure occurs. Same for the parallel-series configuration of Fig. 3(b), if the components 1 and 3 or 2 and 4 fail, the series-parallel system of Fig. 2 remains operating since failing components are recovered by redundant components 2 and 4 or 1 and 3, whereas in the parallel-series configuration of Fig. 3(b) a total system failure occurs.

According to the upper mathematical calculation and physical interpretation, it can be concluded that the total reliability/availability of a series-parallel configuration is greater than the total

reliability/availability of a parallel-series system constructed or built using the same set of components and therefore longer available for the same investment costs. This is the main reason why series-parallel configurations are more suitable in designing reliable systems.

III Genetic Algorithms and Genetic Operators

The revolution of genetic algorithms (shortened GAs) started in the sixties of the 20th century with John Holland, an American computer scientist and psychologist at the University of Michigan [4]. Holland was impressed by the way how biological organisms manage and perform difficult tasks in easy manner [5]. Outgoing from his observations and deep studies Holland developed the concept with the aspect of optimisation and studying self-adaptiveness (self-guidance and self-repair) in biological processes.

GAs are very effective search metaheuristics that mimic the process of natural evolution of species. They represent iterative self-adaptive stochastic techniques based on the idea of randomness. GAs became very popular and widely used over the last decade and are very well suited as universal or common techniques for solving combinatorial optimisation problems, e.g., the very well-known TSP (Travelling Salesman Problem) and redundancy allocation problems like the one discussed in this paper and many other matters [6].

Genetic algorithms require as start point a randomly generated start (initial) population of different chromosomes, also called solution candidates that are encoded according to the handled problem (binary, integer, decimal, etc.) conducting herewith a search in many areas of the solution space at once. The encoding of solutions constitutes the most difficult and challenging task of GAs. The evolving procedure from one population to the next is called generation.

After each generation the new generated solutions are decoded and evaluated with the help of the fitness function. The fitness value of each chromosome represents a measure for the quality (the fitness) of the found solution. A general overview of the genetic cycle is given in Fig. 4.

The genetic run process terminates when at least one of its predefined termination criterions is met, e.g., maximum number of generations or repetitions N_{rep} or a specific number of successive runs without any solution's improvement is reached.

In genetic programming, during each generation or genetic cycle 3 main operators will be executed, hence the selection, crossover or recombination, and the mutation operator.

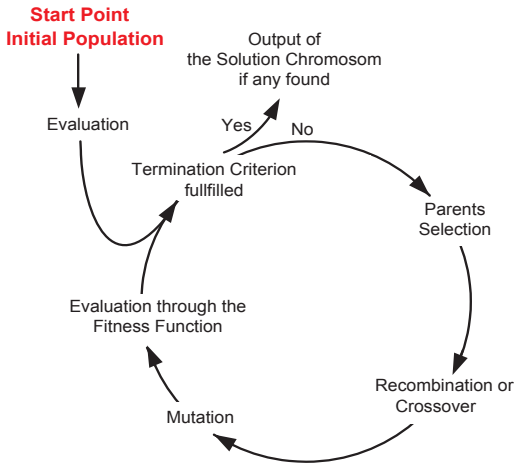

Fig. 4: General overview of the genetic process

These operators will be discussed in short in the following subsections.

a) Selection Operator

Outgoing from a start or initial population of different solution candidates the selection operator will be used in order to randomly select or choose pair of individuals or chromosomes which will reproduce and help building the next population during the genetic cycle. The chromosomes will be selected randomly according to the principle of survival of the fittest (selection probability proportional to relative fitness), which drives the evolution towards optimisation. There are many selection methods, some of them are listed in the following [4]

- Roulette Wheel selection,
- Tournament selection,
- Rank selection,
- Boltzmann selection,
- etc.

b) Recombination or Crossover

Whereas the selection operator determines which chromosomes of the recent population are going to reproduce, the crossover operator performs jumps between the different solution subspaces enabling the exploration of the whole solution space and avoiding herewith premature convergence in addition of exchanging some basic characteristics and inheriting these properties to the offsprings which will join next populations.

There are many crossover techniques used in genetic algorithms [4], like the one-point crossover, two-point crossover, uniform and half uniform crossover and many other crossover techniques.

In the following the one-point crossover operator is shortly discussed. A crossover point depending on the crossover rate and the length of the encoded chromosome is randomly selected on both selected parent chromosomes. All data beyond that point in either chromosome is swapped between the two

parent organisms so that two new individuals also called offspring or children chromosomes result. The one-point crossover procedure is depicted in Fig. 5.

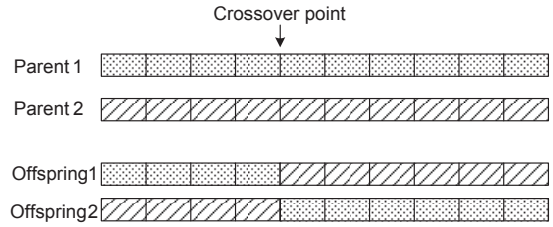

Fig. 5: One-point Crossover

Note, that this simple crossover technique has been used in the genetic algorithm implemented in this paper.

c) Mutation

After the offsprings have been born through crossover, they undergo mutation with a low mutation rate. The mutation operator introduces diversity into the GA algorithm and inserts small disturbance into the properties (genes) of the proposed solutions avoiding herewith convergence into local maxima. After the mutation process has been completed, the new resulting mutated chromosomes can join the next population.

IV Cost- Redundancy Optimisation Problem – Formulation

Two main steps that help improving the total reliability of an engineering system are (1) increasing the reliabilities of the individual components, hence using high reliable components, and (2) using redundancy at various stages [1]. However, total system reliability improvement is accompanied with an increase in total system costs since an increment in component reliability is related to an increase in component cost, and using redundant structures at different stages is accompanied by additional costs incurred by an increase in equipment units required to improve the stages reliabilities.

The cost- redundancy allocation problem can be mathematically formulated as to minimize the cost function $C_{sys}(X,J)$ (objective function) of the whole system given by [1] [7]:

$$C_{sys}(X,J)=\sum_{i=1}^{s} c_{iJ_i} x_i \tag{7}$$

Subject to the total system availability $A_{sys}(X,J)$ which must match or surpass a minimum level of availability required A_0 (inequality constraints)

$$A_{sys}(X,J)=\prod_{i=1}^{s} A_i(x_i) \geq A_0 \tag{8}$$

$$l_i \leq x_i \leq u_i \tag{9}$$

with x_i being a positive integer corresponding to the number of components used in subsystem i. l_i and u_i represent respectively the lower and upper bound of allowed components on stage i.

Finally the optimal structure (X, J) of the series-parallel reliability system according to the solution found with respect to the given constraints will be determined. The cost function C_{sys} (also called the objective or fitness function in the upper optimisation task) is given in equation 6 and is represented as the sum of the costs of the individual components C_{iJi} used in each stage of the subsystems, whereas J_i is the version of the component available on the market used in the found configuration. $A_i(x_i)$ represents the availability of each stage or subsystem which depends on the number of parallel connected components and the availability A_{ij} of the used components.

V Universal Moment Generating Function (UMGF) - Fundamentals

The best way to evaluate the availability of series-parallel MSS systems where the total system capacity must not underpass a specific instantaneous demand level according to the cumulative load curve and which depends on the nominal performance of the individually used components is to use the so called Ushakov- or u-transform [3][8][9]. This mathematical technique occurs also in the literature under universal z-transform or Universal Moment Generating Function (shortened UMGF).

The u-function $u(z)$ of a component with M different possible states is defined as a polynomial in the form

$$u(z)=\sum_{m=1}^{M} p_m z^{W_m} \tag{10}$$

where p_m being the probability, that the nominal performance of the component at state m is equal to W_m. Consider for example a binary state component j (only two states for perfect operating with nominal performance W_j and probability $Pr[W_m=W_j] =A_j$ or totally failing with the probability of failure $Pr[W_m=0] =1-A_j$). According to equation (10) the UMGF of a binary state component is given by

$$\begin{aligned} u_j(z)&=\sum_{m=1}^{2} p_m z^{W_m}=(1-A_j)z^0+A_j z^{W_j} \\ &=(1-A_j)+A_j z^{W_j} \end{aligned} \tag{11}$$

Since the nominal performance of the component in case of total failure is 0.

For the evaluation of the total availability of series-parallel systems with respect to different demand levels required over a partitioned interval of time T using the u-transform, two basic operators are required. The first one is the Γ- Operator and is used for the evaluation of the total u-function of parallel structures, and the other one is the η- Operator which is used to determine the u- function of series structures.

a) Γ- Operator – u(z) of Parallel Configuration

The *u*-function of a parallel system consisting of x_i components is given by

$$u_{parallel}(z) = \Gamma(u_1(z), u_2(z), ..., u_{x_i}(z)) \quad (12)$$

and the total performance function $f(W_1, W_2, ..., W_{xi})$ of the system is the sum of performances of all its components, i.e.,

$$f(W_1, W_2, ..., W_{x_i}) = \sum_{i=1}^{xi} W_i \quad (13)$$

That means for a pair of parallel connected components with the *u*-functions $u_1(z)$ and $u_2(z)$ given according to equation (10) the total resulting *u*-function $u_{parallel}(z)$ of the system is given by

$$u_{parallel}(z) = \Gamma(U_1(z), U_2(z)) = \sum_{i=1}^{n}\sum_{j=1}^{m} P_{1i}P_{2j}z^{W_{1i}+W_{2j}} \quad (14)$$

where *n* and *m* respectively represent the maximal number of states that the components 1 and 2 can take. W_{1i} *and* W_{2j} are respectively the performances of the components *1* and *2* at respective states *i* and *j* which occur with the respective probabilities P_{1i} and P_{2j}. From equation (14) it can be concluded that the *Γ*- operator is nothing else than the polynomial product of the individual *u*-functions of all parallel connected components and therefore equation (12) can be represented as

$$u_{parallel}(z) = \prod_{e=1}^{x_i} u_e(z) \quad (15)$$

For a Subsystem *i* within a series-parallel configuration consisting of x_i different parallel connected binary state components, which UMGF representation is given by equation (11), the *u*-function according to equation (15) is written in the form

$$u_{parallel}(z) = \prod_{e=1}^{x_i} [(1 - A_{ij}) + A_{ij}z^{W_{ij}}] \quad (16)$$

where *j* represents the index corresponding to the version of the component used in case of non-homogeneity. In the case of homogeneity, i.e., all x_i components are identical, equation (16) will be reduced to

$$u_{parallel}(z) = [(1 - A_{ij}) + A_{ij}z^{W_{ij}}]^{x_i} \quad (17)$$

According to the binomial theorem the power representation of equation (17) can be expanded and rewritten as a sum of the form

$$u_{parallel}(z) = \sum_{k=0}^{x_i}\binom{x_i}{k}(A_{ij}z^{W_{ij}})^k(1 - A_{ij})^{x_i-k} = \sum_{k=0}^{x_i}\binom{x_i}{k}A_{ij}{}^k(1 - A_{ij})^{x_i-k}z^{kW_{ij}} = \sum_{k=0}^{x_i}\alpha_{ik}z^{kW_{ij}} \quad (18)$$

with

$$\alpha_{ik} = binm(k, A_{ij}, x_i) = \binom{x_i}{k}A_{ij}{}^k(1 - A_{ij})^{x_i-k} \quad (19)$$

and the binomial coefficients are given by

$$\binom{x_i}{k} = \frac{x_i!}{k!(x_i - k)!} \quad (20)$$

That means, that the u-function or the polynomial z-representation of each parallel subsystem consisting of xi -identical component can be computed according to equation (18).

b) η - Operator – u(z) of Series Configuration

In the case of series connected components, the element with the minimal or least performance becomes the bottleneck of the system. This element makes the last decision about the total system productivity. To calculate the U-function of a system containing *s* elements connected in series, the *η*-operator should be used:

$$u_{series}(z) = \eta(u_1(z), u_2(z), ..., u_s(z)) \quad (21)$$

where the total performance function *f* is given by

$$f(W_1, W_2, ..., W_s) = \min(W_1, W_2, ..., W_s) \quad (22)$$

That means for a pair of components $u_1(z)$ and $u_2(z)$ connected in series the resulting *u*- function of the system $u_{series}(z)$ is given by

$$u_{series}(z) = \eta(u_1(z), u_2(z)) = \eta(\sum_{i=1}^{n}P_{1i}z^{W_{1i}}, \sum_{j=1}^{m}P_{2j}z^{W_{2j}}) = \sum_{i=1}^{n}\sum_{j=1}^{m}P_{1i}P_{2j}z^{f(W_{1i},W_{2j})} \quad (23)$$

Using equation (22), equation (23) will be reduced to

$$u_{series}(z) = \sum_{i=1}^{n}\sum_{j=1}^{m}P_{1i}P_{2j}z^{\min(W_{1i},W_{2j})} \quad (24)$$

c) Series-Parallel Systems - (Γ, η)

Using both composition operators *Γ* and *η* respectively, the UMGF of a series-parallel system can be determined. Since the problem discussed in this paper deals with homogeneous redundancy the *Γ*-operator represented in equation (18) can be used for determining the specific UMGF of each subsystem consisting of identical parallel connected components. Afterward the UMGF $u_{sys}(z)$ of the entire system consisting of *s* series connected subsystems will be determined using the *η*- Operator and is given by

$$u_{sys}(z) = \eta(u_1(z), u_2(z), ..., u_s(z)) = \eta(\sum_{k=0}^{x_1} \alpha_{1k} z^{kW_{1j}}, \sum_{k=0}^{x_2} \alpha_{2k} z^{kW_{2j}}, ..., \sum_{k=0}^{x_s} \alpha_{sk} z^{kW_{sj}}) = \sum_{m=0}^{M} \delta_m z^{W_m} \quad (25)$$

where δ_m and W_m are real numbers determined according to equation (24).

The evaluation of the probability that the whole system overpasses or achieves a specific level of performance W_0^k is given by

$$A_{ms}(W_0^k) = \Pr(W_m \geq W_0^k) = \sum_{W_m \geq W_0^k} \delta_m \quad (26)$$

Hence the total availability $A(X,J)$ of the system for all demand level within a period of time T is obtained by

$$A(X,J) = \frac{1}{T} \sum_{k=1}^{K} \sum_{W_m \geq W_0^k} \delta_m T_k = \frac{1}{\sum_{k=1}^{K} T_k} \sum_{k=1}^{K} \sum_{W_m \geq W_0^k} \delta_m T_k \quad (27)$$

where K and T_k represent respectively the number of demands levels and the operation time corresponding to each demand level.

VI Numerical Results – Matlab Evaluation

For mathematical evaluation and graphical representation a Matlab routine has been written which imports the data belonging to the different test that has been resumed in excel sheets and executes the genetic algorithm over this data, in order to determine the corresponding series-parallel structure belonging to the optimal (minimal) found cost value with respect to the predefined availability constraint A_0. 3 different models have been tested, the *Levitin*-model with 4 subsystems and 3 demand levels (lev4-(4/6)-3), the *Levitin*- model with 5 subsystems and 4 demand levels (lev5-(4/9)-4), and finally the *Ouzineb*- model with 6 subsystems and 4 demand levels (ouz6-(4/11)-4). The data has been taken from [9][10]. Note that the representation -(jmin/jmax)- stands respectively for the minimum and maximum number of versions available on the market and which can be taken from the data. The imported data contains the number of subsystems of each model and the number of versions available on market for each component that may be used on each stage, in addition to the cost, availability and nominal performance of the individual components.

The algorithm starts by generating a random population of size Pop_{size}, where the chromosomes are integer encoded according to read data (integer based coding). Each chromosome within the population is represented as a vector Y with a length l_{chrom} equal to twice the number of subsystems. Y results by horizontally concatenating 2 vectors X and J each of length $l_{chrom}/2$. Both subvectors X and J represent respectively the number and the version of components used in each subsystem (homogeneous redundancy, i.e., only one version is used in each subsystem). Consider as an example the chromosome Y=[1 2 3 2 4 3 1 5]. This belongs to a model containing 4 subsystems, where the number of components used on each stage is given by the first 4 elements of the vector, i.e., subvector X=[1 2 3 2] and the version number used on each stage is given by the next 4 elements, i.e., the subvector J=[4 3 1 5].

The algorithm parameters have been set in almost all cases as in the following:

- Population size Pop_{size}=100
- Total number of repetitions N_{rep}=200 to 500
- Crossover rate $P_{crossover}$= 0.8
- Mutation rate $P_{mutation}$= 0.2
- Min. number of elements pro. Stage l_i= 1
- Max. number of elements pro. Stage u_i= 10

After ranking and evaluating each population the best 2 chromosomes are selected explicitly to mate, recombine, and finally mutate in order to build new offsprings and complete a next population of size Pop_{size}. This genetic procedure repeats until the predefined maximal number of generations N_{rep} is reached.

Fig.6 shows the results of one run of the GA over the Lev4-(4/6)-3 model data by an availability constraint of A_0=0.900. The different plots show the evolution process outgoing from the random initial population until the predefined maximum number of generations is reached. The best result got after each genetic cycle is depicted.

The time needed to find the best solution depends on the quality of solutions generated in the start population and on how the selected fittest chromosomes evolve throughout crossover and mutation. On the left hand side of Fig. 6 the best solution found (Top: cost value, bottom: Availability value for found cost) during each generation is plotted against generation number whereas the same plots are represented on the right hand side against processing time. In the plot, the best chromosome corresponding to the optimal (minimal) found cost subject to the given availability constraint is represented in addition to the generation and convergence time for which the best result has been identified.

The best test results got within 30 successive runs of the genetic algorithm over the different models mentioned previously are represented in table 1. The last column of table 1 represents the time results computed by Ouzineb for the 3 considered models using hybridised genetic approaches (Tabu Search) [9][10]. It is important to mention at this point, that all results (availability and cost) got over the handled

models match the results obtained by Ouzineb [9][10]. Note that the Matlab algorithm implemented in this work showed very high effectiveness and robustness concerning the obtained results and rapid convergence on qualitative and mostly exact solutions, as it can be recognised from table 1 when comparing convergence time corresponding to both algorithms.

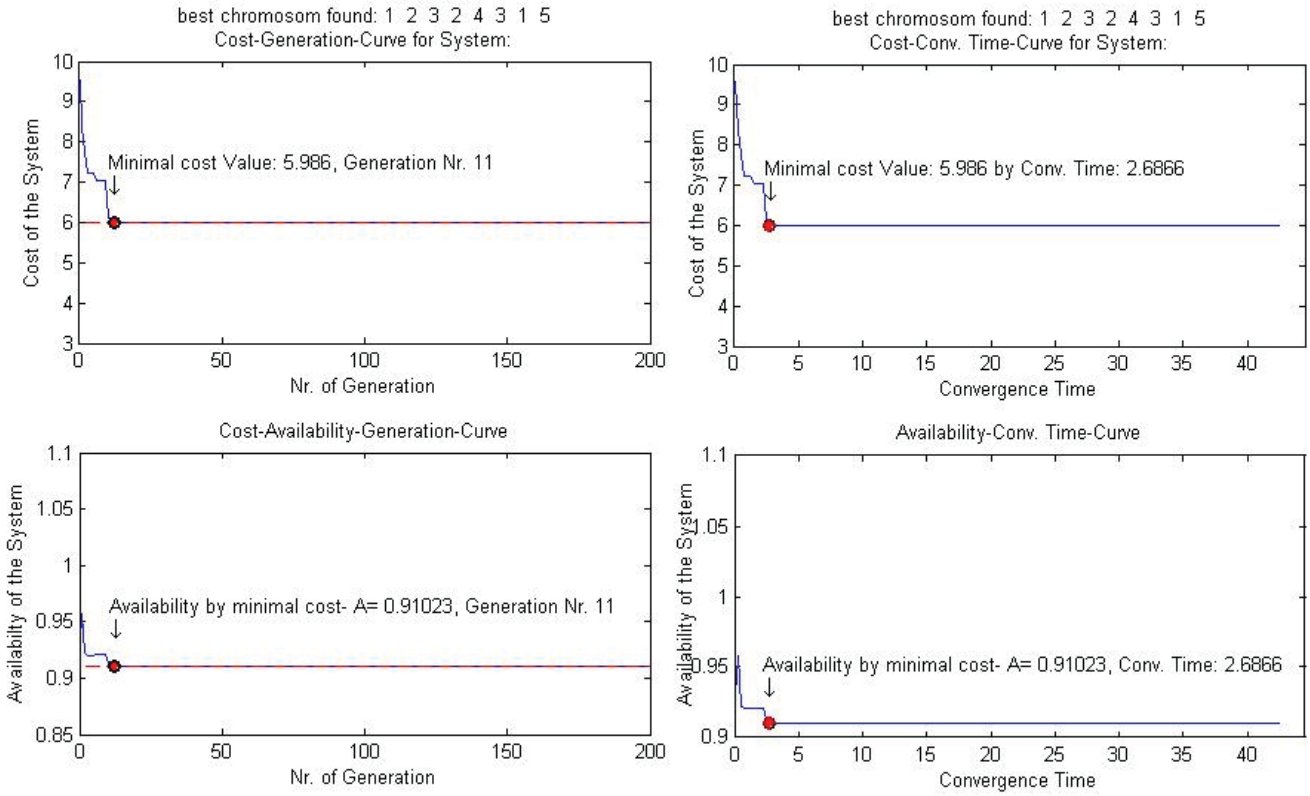

Fig. 6: Example of the Results found by the genetic algorithm run (Levitin- model containing 4 subsystems, availability constraint A_0=0.900). The cost - generation and availability - generation curves dependencies are depicted on the left side, while the cost-time and availability-time dependencies are depicted on the right side. On the top of each plot u can see the chromosome or system structure corresponding to the best found or optimal values.

Problem Name	Av. Constraint A_0	Av. Value $A(X,J)$	Cost $C(X,J)$(mln $)	Best found Chromosome [X,J]	Time t (s)	Time(Ouz) t_{Ouz}(s)
lev4-(4/6)-3	0.900	0.9102	5.986	[1 2 3 2 4 3 1 5]	00.40	05.06
	0.960	0.9609	7.303	[2 3 3 2 2 3 1 5]	01.15	05.18
	0.990	0.9917	8.328	[3 3 3 5 1 3 1 2]	00.26	05.03
lev5-(4/9)-4	0.975	0.9774	16.450	[2 2 3 3 1 2 3 2 7 2]	06.93	52.91
	0.980	0.9808	16.520	[2 6 3 3 1 2 5 2 7 2]	23.36	102.36
	0.990	0.9937	17.050	[2 2 3 3 3 2 3 2 7 4]	33.92	86.57
ouz6-(4/11)-4	0.975	0.9790	11.241	[4 4 5 7 2 1 3 1 2 2 3 4]	05.80	112.71
	0.980	0.9802	11.369	[4 5 5 8 2 1 3 1 2 2 3 4]	07.22	126.49
	0.990	0.9902	12.764	[4 4 4 8 2 2 3 1 2 2 3 4]	17.71	124.76

Table 1. Comparison of computation time and results using the GA with computation results of Ouzineb using Hybridised GA+TS (Tabu Search)

VII Conclusion and future works

Based on the results represented in table 1 the Matlab algorithm implemented in this paper was well working and very efficient since the convergence into optimal results takes shorter time in comparison to the computation times implemented by Ouzineb [9][10]. Furthermore the algorithm was converging into the best optimal solution in almost all runs. One small disadvantage is the one known by genetic algorithms and which is resumed in the fact that the best optimal solution is not guaranteed or ensured in each run, due to the limitation of the maximum number of iterations that may result, that some regions of the search or solution space remains unexplored.

One of our future intentions would be to test genetic algorithms on non-homogeneous redundant series-parallel structures, since those structures allow though diversity system modernisation and achieving desired availability at lower cost than homogeneous systems. In addition that this kind of systems is less susceptible against common cause failures which represent one of the biggest disadvantages of homogeneous structures that may lead to the failure of the entire System. A further step will be to explore and investigate new methaheuristics or combination of algorithms which can be applied on such optimisation task discussed in this paper in order to get better performance and promising results.

References

[1] Way Kuo,V. Rajendra Prasad, Frank A.Tillman, and Ching-Lai Hwang. \Optimal Reliability Design, Fundamentals and Applications", *Cambridge University Press 2001*, ISBN 0521 78127 2.

[2] Josef Börcsöck, \Functional Safety- Basic Principles of Safety –related Systems", *Hüthig Verlag Heidelberg*, ISBN 978-3-7785-2986-7, , 2007.

[3] Gregory Levitin, Anatoly Lisnianski, Hanoch Ben Haim, David Elmakis. \Genetic Algorithm and Universal Generating Function Technique for Solving Problems of Power System Reliability Optimization". *The Israel Electric Corporation Ltd., Planning Development & Technology Division*, April 2000.

[4] M. Affenzeller, S. Winkler, S. Wagner, and A. Beham, Genetic Algorithms and Genetic Programming, Modern Concepts and Applications". *CRC Press 2009*.

[5] Holland J., \ Adaptation in Natural and Artificial Systems". The University of Michigan Press, Ann Arbor, Michigan,1975.

[6] Zhigang Tian, Ming J. Zuo, and Hongzhong Huang. \Reliability-Redundancy Allocation for Multi-State Series-Parallel Systems". *IEEE Transactions on Reliability*, Vol. 57, No. 2, June 2008

[7] Frank A. Tillman, Ching-Lai Hwang, Way Kuo. \Optimization Techniques for System Reliability with Redundancy- A Review". *IEEE Transactions on Reliability*, August 1977

[8] Gregory Levitin, \The Universal Generating Function in Reliability Analysis and Optimization". Springer ISBN-13: 978-1-85233-927-2, 2005.

[9] Mohamed Ouzineb, \Heuristiques efficaces pour l'optimisation de la performance des systèmes séries-parallèles". Département d'informatique et de recherche opérationnelle Faculté des arts et des sciences, Université de Montréal 2009.

[10] Mohamed Ouzineb, Mustapha Nourelfath, Michel Gendreau \Tabu search for the redundancy allocation problem of homogeneous series-parallel multi-state systems". *Reliability Engineering and System Safety 93 (2008)*, 1257–1272, 2008.