SOME EFFICIENT ALGORITHMS TO DEAL WITH REDUNDANCY ALLOCATION PROBLEMS

Submitted: 26th June 2019; accepted: 25th March 2020

Mustapha Es-Sadqi, Abdellah Idrissi, Ahlem Benhassine

DOI: 10.14313/JAMRIS/4-2020/45

Abstract: *In this paper, we will discuss some algorithms in order to better optimize the problems of redundancy allocation in multi-state systems. The goal is to find the optimal configuration of the system that maximizes the availability and minimizes the investment cost. The availability will be evaluated using the universal generating function. In first step, our contribution consists in improving the genetic algorithm. In a second step, in the framework of the Constraint Programming, we propose a new method of optimization based on the Forward Checking as solver. Finally, we used the top-k method in our choice that helps us to get the best k elements from all possible values with highest availability. In comparison with the chosen study, our methods yield better results that satisfy the constraints of the problem in a shorter time.*

Keywords: *Redundancy Allocation Problem, Constraint Programming, Forward Checking, Optimization, Genetic Algorithm, Top_k*

1. Introduction

We will treat in our study the electrical network whose structure is multi states. In addition, this structure is serie-parallel. The primary function of a power grid is to provide electricity to its customers at optimal operating costs with the assurance of quality and reasonable continuity at all times [1]. To plan an electrical system, it is imperative to identify the variables and the constraints to model it [2]. Because of their sensitivity to defects, these systems have increasing complexity. So it is necessary to improve their reliability and install redundant components in parallel [3]. As we know, the energy supply process is a high-level complex installation (production, transport distribution and consumption). The process requires several interconnected subsystems to achieve the high level objectives expected.

The components used in each step often work in essentially different operating modes, characterized by varying loads and performance, demand management of a different nature, or different environmental conditions [4]. These modes result in different failure rates and life distributions. However, in terms of reliability analysis, this is a problem that is not officially solved without quantifying the effect of multiple loads on systemic reliability [5]. The architecture of these systems is a series-parallel structure consisting of several components whose performance states vary from nominal to full failure, so the state of the global system is described by the state of its components and such systems are called (multi-state systems MSS). The most efficient tool for this study is the universal generating function UGF [6].

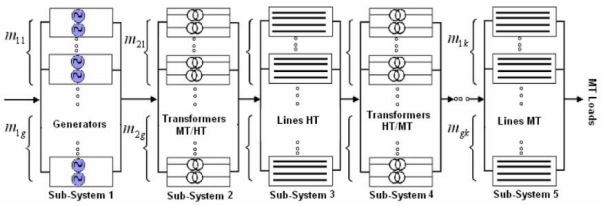

Fig. 1. Architecture of an electrical parallel-series system [6]

In the next section, we will describe the mathematical formulation of the problem; our new methods (and the various parameters associated) are reported in Section III, followed by the experimental results and their comparison with those of the literature. Discussion and a conclusion complete the text.

2. Mathematical Formulation of the Problem

In this paper, we will adopt the mathematical tool detailed by the authors in [7, 8, 9, 10]. A serial-parallel multi-state system is often made up of n subsystems in series. Each subsystem i ($1 \le i \le n$) contains ki parallel components with versions Vi. The version of each component is v such that ($1 \le v \le V_i$). All these components are characterized by their availability A_{iv}, their performance G_{iv}, their cost C_{iv} in the market. The structure of each subsystem i, is defined by the number K_{iv} of parallel components for each version [10].

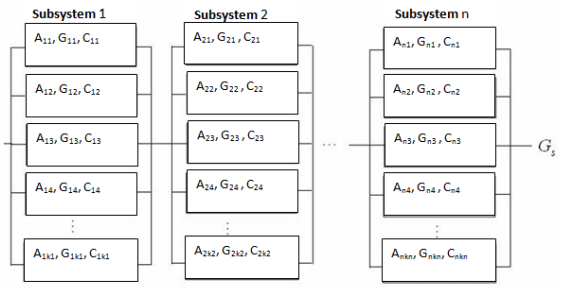

Fig. 2. Series-parallel system composed of n subsystems each with k_i components

2.1. Cost

Since the cost function is linear, the global cost of the whole system is defined by the vector:

$$k_i\left\{k_{iv_i}\right\} \text{ with } \begin{cases} 1 \le i \le n \\ 1 \le v \le V_i \end{cases}$$

Given the set of vectors $\{k_i = k_1, k_2, ..., k_n\}$, the cost of the entire system is given by the formula:

$$C = \sum_{i=1}^{n} \sum_{v=1}^{v_i} K_{iv} C_{iv} \tag{1}$$

2.2. Availability and Demand

We define the availability of a component as its ability to be operational at a given time t. It can be formulated mathematically by the following formula:

$$a\,(t) = p\,[\text{s functioning at time t}]. \tag{2}$$

The system must provide a g0 demand predicted by a cumulative curve distributed often over four periods. so to meet this demand the availability of the system must be greater than or equal to this demand [11]. If a(t) represents the instantaneous availability of the system, g(t) its performance and g_0 its required demand for a period t_m, we can write using (2):

$$a = \text{prob}\,[g(t) \ge g_0]. \tag{3}$$

The demand described above, is used to divide the operation period t in m time intervals t_m ($1 \le m \le m$), the availability of the mss can be written as follows:

$$A = \frac{1}{\sum_{m=1}^{M} T_m} \sum_{m=1}^{M} \text{Prob}\left(G(t) \ge G_m\right) T_m \tag{4}$$

In engineering and for a given system S, the availability A is related to the index of load loss probability (LOLP) and is defined by [12,13]: LOLP= Prob($G < G_0$)
This allows us to write:

$$\text{LOLP} = 1 - A. \tag{5}$$

LOLP represents the probability that the system cannot provide the given demand G_0. In the case of components with total failure that we consider in this study, each component j is characterized by its nominal performance G_j and availability A_j. Thus we can write:

$$\begin{cases} \Pr\left[G_j = W_0\right] = A_j \\ \Pr\left[G_j = 0\right] = 1 - A_j \end{cases} \tag{6}$$

2.3. Universal Generating Function UGF

The authors in [10] have largely represented the UGF technique that we will describe below:

UGF were introduced by I. Ushakov in 1986, and since then many scientists have proved the effectiveness of the method such as G. Levitin and A. Lisnianski [14–16].

Let n be the number of discrete and independent random variables $X_1, ..., X_n$ and let us assume that each variable X_i can be represented by vectors x_i and p_i, such as:

$$\begin{cases} x_i = \left(x_{i1}, \ldots, x_{ik_i}\right) \\ p_i = \left(p_{i1}, \ldots, p_{ik_i}\right), \text{s.t.} p_{ij} = \Pr\left\{X_i = x_{ij}\right\} \text{ with } j = 0, \ldots \end{cases} \tag{7}$$

The $\mathcal{Z}$ Transform of the variable X_i is defined by the distribution function of the polynomial form:

$$u_i(z) = \sum_{j_i}^{k_i} p_{ij_i} Z^{X_{ij_i}} \tag{8}$$

Let $\otimes_f$ be a multiplicative operator, this operator acts differently on the function $u_i(z)$, on the one hand:

$$\otimes_f \sum_{j_{i=1}}^{k_i} p_{ij_i} Z^{X_{ij_i}} = \sum_{j_1=1}^{k_1} \sum_{j_2=2}^{k_2} \cdots \sum_{j_n=1}^{k_n} \left(\prod_{i=1}^{n} p_{ij_i} Z^{f(x_{ij_i}, \ldots, x_{ij_n})} \right) \tag{9}$$

If the random variable is identical to a given performance as is the case in our study $X_i = G_i$, and according to (9), the resulting $\mathcal{U}$-function from the combination of a set of m components is:

$$\begin{aligned} &\otimes_f \left(u_1(z), u_2(z), \ldots, u_m(z)\right) = \\ &= \sum_{j_1=1}^{k_1} \sum_{j_2=2}^{k_2} \cdots \sum_{j_m=1}^{k_m} \left(\prod_{i=1}^{m} p_{ij_i} Z^{f(G_{ij_i}, \ldots, G_{ij_m})} \right) \end{aligned} \tag{10}$$

Note that $f(G_1, G_2, ..., G_m)$ represents the equivalent productivity of m components. When these components are connected in series, the function takes the following form:

$$f(G_1, G_2, ..., G_m) = \min(G_1, G_2, ..., G_m) \tag{11}$$

And in the case where the m components are connected in parallel, the function becomes:

$$f\left(G_1, G_2, \ldots, G_m\right) = \sum_{i=1}^{m} G_i \tag{12}$$

To satisfy a requested performance state, we introduce a satisfaction operator defined as the following:

$$\phi\left[\left(pZ^{G_i}\right)Z^{-G_0}\right] = \phi\left(pZ^{G_i - G_0}\right) = \begin{cases} p \text{ if } G_i \ge G_0 \\ 0 \text{ if } G_i < G_0 \end{cases} \tag{13}$$

This operator also checks the following property:

$$\phi\left(\sum_{i=1}^{I} p_i Z^{G_i - G_0}\right) = \sum_{i=1}^{I} \phi\left(p_i Z^{G_i - G_0}\right) \tag{14}$$

The objective of this operator is to eliminate all the terms not satisfying the given performance demand: $G_i < G_0$.

2.4. Application of the UGF on the MSS

To assess the availability of a MSS, the $\mathcal{S}$ operator is introduced for the series composition and the operator $\mathcal{P}$ for the parallel one. These operators determine the polynomial U(z) for a group of elements.

Parallel components. When the performance G is linked to productivity or the system capacity, the overall performance of the parallel components of the system is the sum of performances according to equation (12). Therefore the $\mathcal{U}$-function $u_{par}(z)$ of a component i containing X_i elements in parallel can be obtained by using the operator $\mathcal{P}$:

$$u_{par}(z) = \mathcal{P}\left\{u_1(z), u_2(z) \ldots u_{X_i}(z)\right\}$$

Where:

$$\mathcal{P}(G_1, G_2, \ldots G_n) = \sum_{i=1}^{X_i} G_i$$

Thus, by using property's $\mathcal{U}$-functions of equations (8), (10) and (14) we find two parallel components:

$$\mathcal{P}(u_1(z), u_2(z)) = \mathcal{P}\{\sum_{i=1}^{n} p_i Z^{a_i}, \sum_{j=1}^{m} q_j Z^{b_j}\} =$$
$$= \sum_{i=1}^{n}\sum_{j=1}^{m} p_i q_j Z^{(a_i + b_j)}$$

a_i are b_j physically interpreted as the successive performances of two elements and n and m are the numbers of levels of these performances. p_i et q_j represent the probabilities equilibrium of each level. We can see that the operator $\mathcal{P}$ performs a simple multiplication of individual

$\mathcal{U}$-functions of each component:

$$U_{par}(z) = \prod_{i=1}^{X_i} u_i(z) \tag{15}$$

Series components. In the case of s elements in series, the functionality of such system is provided by the element with the lowest performance, it acts as a bottleneck for the system. In this case, the $\mathcal{U}$-function $u_{ser}(z)$ is given using the operator $\mathcal{S}$ which also performs simple multiplications of the individual $\mathcal{U}$-functions of each component. Considering n cardinal levels, equations (9) and (10) allow to write the following equation:

$$u_{ser}(z) = \mathcal{S}\{u_1(z), u_2(z)\}$$

with:

$$\mathcal{S}(G_1, G_2, \ldots G_n) = \min\{G_1, G_2, \ldots G_n\}$$

So for two elements in series and using equations (8), (9) and (14) we obtain the following:

$$\mathcal{S}(u_1(z), u_2(z)) = \mathcal{S}\{\sum_{i=1}^{n} p_i Z^{a_i}, \sum_{j=1}^{m} q_j Z^{b_j}\} =$$
$$= \sum_{i=1}^{n}\sum_{j=1}^{m} p_i q_j Z^{\min\{a_i, b_j\}} \tag{16}$$

Series-parallel systems. The $\mathcal{U}$-function of the entire parallel-series system is given, using consecutively, operators $\mathcal{P}$ and $\mathcal{S}$. In addition, in the case of systems with total failure as cited in section 3(ii), for each i component containing k_i version of components v_i, having the nominal performance G_{iv} and availability A_{iv} according to equation (8) we can write:

$$\begin{cases} \Pr(G = G_{iv}) = A_{iv} \\ \Pr(G = 0) = 1 - A_{iv} \end{cases}$$

The $\mathcal{U}$-function of each component will contain only two terms, which gives:

$$u_i^*(z) = (1 - A_{iv}) Z^0 + A_{iv} Z^{G_{iv}}$$
$$u_i^*(z) = 1 - A_{iv} + A_{iv} Z^{G_{iv}}$$

So the $\mathcal{U}$-function of a component i is:

$$u_i(z) = \left[u_i^*(z)\right]^{k_i} = \left[1 - A_{iv} + A_{iv} Z^{G_{iv}}\right]^{k_i}$$

Therefore, for a system consisting of n subsystems in series, for each subsystem i and for each version of components v, the subsystem is modeled as the following:

$$u_{par}^i(z) = \mathcal{P}\begin{pmatrix} \left(A_{i1} Z^{G_{i1}} + (1 - A_{i1})\right)^{K_{i1}} \\ \left(A_{i2} Z^{G_{i2}} + (1 - A_{i2})\right)^{K_{i2}} \\ \ldots \\ \left(A_{iV_i} Z^{G_{iV_i}} + (1 - A_{iV_i})\right)^{K_{iV_i}} \end{pmatrix}$$

The whole system contains n subsystems, thus, introducing the operator $\mathcal{S}$ we obtain:

$$\left[u_{par}^1(z), u_{par}^2(z), \ldots u_{par}^n(z)\right] \tag{17}$$

Once $u_{ser}(z)$ is determined, we will need to assess the probability to meet the given demand required for w_0, for this, using equation (15) we have:

$$\Pr(G \geq W) = \phi\left(U_{ser}(z) Z^{-W_0}\right) \tag{18}$$

2.5. Formulation of the Problem

The problem can be summarized as the following:

- Maximize the availability resulting from:

$$\begin{cases} \otimes \langle \sum_{i=1}^{n} \sum_{j=1}^{m} P_i Q_j z^{a_i + b_j} \rangle z^{-w} \\ \langle \sum_{i=1}^{n} \sum_{j=1}^{m} P_i Q_j z^{\min\{a_i, b_j\}} \rangle z^{-w} \end{cases}$$

While Respecting the given demand $G^{\min\{a_i, b_j\}} \geq W_0$

- Minimize the Cost: $\min \sum_{i=1}^{n} \sum_{V=1}^{V_i} K_{iv} C_{iv}$

3. Effective Genetic Algorithm EGA

Genetic algorithms attempt to simulate the process of natural evolution following the Darwinian model in

a given environment. The individual is represented by a chromosome consisting of genes that contain the hereditary characteristics of the individual. The principles of selection, crossing, mutation are inspired by natural processes of the same name. It is associated with the value of the criterion to be optimized, its adaptation. We then generate iteratively populations of individuals on which we apply processes of selection, crossing and mutation. We start by generating a random population of individuals. To go from generation k to generation k + 1, the following operations are performed. At first, the population is reproduced by selection where the good individuals reproduce better than the bad ones. Then, a cross is applied to the pairs of individuals (the parents) of a certain proportion of the population to produce new ones (the children). A mutation operator is also applied to a certain proportion of the population. Finally, new individuals are evaluated and integrated into the population of the next generation2.

Several criteria for stopping the algorithm are possible: the number of generations can be fixed a priori (constant time) or the algorithm can be stopped when the population does not evolve sufficiently quickly.

Genetic Algorithm (EGA), on a MSS with the objectives of minimizing the cost and maximizing the availability of the given system. The algorithm for EGA is the following:

EGA (imputs: file)
Number_generation←0;
Population←Init_population(P[NUMINDIVIDUAL]);
Evalpopulation(P[Number_generation]);
While(Number_generation<MAXNUMBER);
P_S ← selection(Population);
P_C ← crossover(P_S);
mutation(P_C);

Population←addnewindividualsIFDosentExist (Population,P_C);
Population ← RemoveWeakestIndividuals (Population);
Number_generation++;
End While
Return best;
End

3.1. Encoding of Solutions

Our solutions are in the form of a set of strings that represents the set of the sub-systems of our system, and each string is a set of integers where the length of the set is equal to the number of devices of the subsystem, and each integer reflects the id of the device.

3.2. Evaluation of Each Solution

Each time an individual is created, a fitness value is associated to it. This value is used by the selection process to favor the most suitable solutions, as it reflects the performances of such individual towards our problem.

We proposed two fitness functions to evaluate the solutions, the first one included in the EGA1, and the second one in the EGA2.

The evaluation function of the EGA1 to calculate the fitness, in order to satisfy the objectives of, maximizing availability and minimizing the cost is the following:

$$\text{Fitness\_function1(System)} = \max\left(\frac{A_S \times C_0}{C_S}\right)$$

where:
C_s : is the cost of the individual component.
A_s: is the availability of the individual component.
C_0: is the initial cost which is used as an upper bound as shown in the constraint

The evaluation function of the EGA2 to calculate the fitness, in order to satisfy the objectives of, maximizing availability and minimizing the cost where is the following:

$$\text{Fitness\_function1 (System)} = \max(\alpha \times A_s + \beta \times C_s)$$

where:
α and β are respectively the given weights for availability, cost of the system and that verified : α + β=1.

3.3. Initial Population

The choice of the initial population is based on a totally random solutions design, meaning that each subsystem of the given system has initially a random number of devices where these random solutions respect the constraints: $A \geq A_0$; $G \geq G_0$ et $C \geq C_0$. G is the system's performance, and G_0 the energy demand.

3.4. Selection

The selection process defines how many times an individual is involved in the re-production process, the individuals with the best performances (best fitness values) are selected more often than the others and are used in the following step which is the reproduction. Selection has done using roulette wheel scheme.

3.5. Reproduction

The reproduction or variation helps to find better individuals, by producing individuals based on the best solutions of the current population. It is carried out using two main operators: Crossover and mutation. We first create the set A of the common genes in the both selected somes P_1 and P_2, then we create the set B and C which contains respectively P_1 – A and P_2 – A. Then we add to Z_1 and Z_2 all elements of A, and we choose randomly in which set we will take the next element, if it's the set B, we add the element to Z_1 and we add the element to Z_2, else we do the same with choosing the unchosen set.

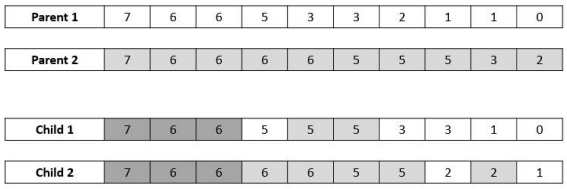

Fig. 3. An example of the crossover operator

After the crossover, the mutations must occur with a low probability Pm. For EGA, mutations occur with a probability is the length of the individual encoding. And it uses the Swap mutation, since, as mentioned before, changing the order of integers for a system does not change anything. Evaluation and selection for replacement.

3.6. Termination

The termination criterion we chose for EGA is the number of generations. Once this number is reached EGA return the best solution found.

4. Constraint Satisfaction Problem in the Four Algorithms: EFC1, EFC2, TopK-FC and GenGA-FC

4.1. Constraint Satisfaction Problem

We consider here a CSP defined by a triplet (X, D, C) where X is a set of n variables (X_1, X_2, ..., X_n) their respective finite domains (D(X_1), D(X_2), ..., D(X_n)) and C a set of relations or constraints between these variables (a constraint on X_{i_1}, X_{i_2}, ..., X_{i_k} is a subset of the Cartesian product $D(X_1) \times D(X_2) \times ... \times D(X_k)$). For an optimization problem (here of maximizing a function), we also consider a cost function *f* and a constraint on this cost $f(X_1, X_2, ..., X_n) \leq C$ where *C* is a constant that the optimization strategy makes evolve. We express our problem in constraint logic programming (CLP) [17] and use the ECL^iPS^e system [18] which implements all the constraints classical, linear (# =, # ≤, ...) and others (alldistinct, element, ...), and also allows to simply define new ones (direct operations on domains, precise control of coroutining, ...). The min_max predicate (minimize, maximize, ...) optimizes a linear expression by integrating the resolution goal of the problem (usually variable instantiation, labeling) within a Branch & Bound (i.e. search tree path with pruning by limitation of the cost function).

4.2. Modeling the Problem of RAP Applied to a Series-Parallel System as a VCSP

As explained above, we will propose a modelling for the redundancy allocation problem as a VCSP. For that, we have to define variables, domains of variables, constraints and objective functions.

i. Variables:
$X = \{X_1, ..., X_n\} =$
$\{subsystem_1, ..., subsystem_n, j_i, Dev\};$

ii. Domains:

$D = \{D_{x_1}, ..., D_{x_n}\}; = D_{subsystem_i} = \{(C_{i_{ji}}, A_{i_{ji}}, G_{i_{ji}})\},$

$D_{Dev} = \{Dev_{i_{ji}}\} = \{1, ..., Dev_{i_{max}}\}$; *et* $D_{ji} = \{1, ..., k_i\};$

where:
- *n is the number of subsystems, i is a number of a subsystem*
- j_i *is the number of devices for each subsystem i*
- $\{C_{i_{ji}}\}$ *is the set of costs that are floats, where* $C_{i_{ji}}$ *is the cost of a device* j_i *of a subsystem i*
- $A_{i_{ji}}$ *is the set of availability values that are floats, where* $A_{i_{ji}}$ *is the availability of a device* j_i *of a subsystem i*
- $\{G_{i_{ji}}\}$ *is the set of performance values that are floats, where* $G_{i_{ji}}$ *is the performance of a device* j_i *of a subsystem i*
- $\{Dev_{i_{ji}}\}$ *is the number of devices that can be chosen, where* $Dev_{i_{ji}}$ *is the number of devices* j_i*, we can choose for a subsystem i*

iii. Constraints:

- $\sum_{i=1}^{k_i} Dev_{i_{ji}} \leq Dev_{i_{max}}$

  this constraint assures that the choice of devices respects the given required number of devices for each subsystem of the system
- $UFG(S_{G \geq G_0}) \geq A_0$*, this constraint assures that the solution S must be available for a performance that respects a given demand. This is calculated using the UGF.*

iv. Objective functions:
- *Maximize the availability A, st* $A \geq A_0$ *under the constraint:* $G \geq G_0$
- *Minimize the cost C, st.* $C \leq C_0$

The problem here is to conceive the configuration of a system by making a choice of components and by allocating an appropriate level of redundancy.

4.3. Solver Based on Forward Checking Algorithm

To solve our problem using the VCSP model and the UGF introduced above and compare it with the previous approaches, we adapted and extended the forward checking algorithm [19], which consists of constructing a solution, by considering assignments to variables in a particular order; an order where the constraints are satisfied. The vector in our case is a set of assignments of devices to subsystems. A solution vector is a set where the devices choice satisfied the constraints (Cost, Availability, and Performance). The forward checking examines partial solutions, which

are assignments to a sub-set of the variables, and try to extend those partial solutions until all variables are assigned; it prevents assignments that guarantee later failure. When we are considering a possible value v_{il} for the current variable V_i it is sufficient to look for a zero in $Domain_i$. Hence, we do not need to do the backwards consistency checks that are characteristic of backtracking. The price, of course, is that when we make a successful assignment to the current variable, we must check it against all outstanding values of the future variables, updating Domain as necessary.

After initialization, the call of Forward Checking (i) will print all solutions. An assignment to fails if there is a domain wipe-out|| (DWO), which means that we have discovered that every value of some future variable is inconsistent with our choices so far.

In our case, the set $s = s_1, \ldots, S_n$ is the set of devices that leads to a solution that respects the objectives of performance and cost. The availability and cost of that solution will help us to compare our method with some of the most relieving works of the literature.

In the EFC1, EFC2, topK-FC, we added a third constraint which is:

$$\sum_{l=1}^{n}\sum_{j_i}^{k_i} r_i Dev_{i_{,ji}} C_{i_{,ji}} \le C_0$$

This constraint allows optimizing each subsystem depending on its number of devices that constitute the system. It also allows respecting a given maximum value of cost for a system.

Note that:

$$r_i = \frac{Dev_{i_{ji}}}{\sum_{j_i}^{k_i} Dev_{i_{ji}}}$$

The forward checking method, which is in common between all our algorithms, can be resumed in the following algorithm:

```
Forward-Checking (i)
  % loop on each value of the domain of and checks the
  constraints in order to find a solution
For each (C_ji,A_ji,G_ji) ∈ D_subsystem_i
      s_i ← (C_ji,A_ji,G_ji)[1];
  If ( C_ji < C_0 / (r_i Dev_i,ji) ) then
    If I=n then
      If (UGF(S1, ..., Sn)_G > G_0 ≥ A_0 then
            Print S1, ..., Sn;
        Else
      If Check-Forward(i) then
        Forward-checking(i+1);
        Restor(i);
      End If
    End If
  End If
End If
End for each
  End
```

4.4. Extended Forword Checking 1: EFC1

As we added in the constraints, the partial cost which has to be taken in consideration, the number of possible values is still high, so we developed a specific method to respond to that need. The domain initialization method is detailed in the following algorithm:

```
Domain-Initialization (Dev_i,ji, k_i)
Result : list;
      i, i_1, j, k: intenewger;
      temp, temp_1: string ;
        For i_1 from 0 to Dev_i,j_i – 1 do
          temp ← i_1 ; // empty string
          k ← 0;
          While (k < k_i) do
            temp ← i_1;
            k++;
          end while
          result.add(temp);
          j ← temp.length();
            for i from i_1 + 1 to k_i – 1 do
          temp_1 ← temp.substring(0, j/i)
            for z from j/i to temp.length() do
              if z mod Dev_i,j_i = 0 then
              temp_1 ← temp_1 + 1
              else
temp_1 ← temp_1 + (z mod (Dev_i,j_i –1))
          End if
        End for
        Result.add(temp_1);
        temp ← temp_1
      End for
    End for
  Return result;
End
```

4.5. Extended Forword Checking 2: EFC2

The domain initialization method in this algorithm was implemented by getting all the possible values respecting the participative cost and returning domains chosen randomly from the previous list not exceeding a given limit. The domain initialization method is detailed in the following algorithm:

```
Domain-Initialization (i, Dev_i,ji, k_i, C_i, limit)
% is the number of the subsystem
% is the partial cost
% is the limit for the returned list where the number of
  possible value ≤ limit * Dev_i,ji
  result : list;
  result Possible-Values(0, result, k_i, C_i, i);
    if result.size() > limit * Dev_i,ji then
      result ← make-to-limit(result, limit * Dev_i,ji);
    end if
    return result;
End
      Possible-Values (j, lst, k_i, C_i, i)
      newlist : list;
      e, temp : string;
```

```
        k : integer;
        cost : float;
          if j = k_i then
            return lst;
          end if
            for each e ∈ lst do
              for k from 0 to k_i – 1
                temp ← e + k
                Cost ← cost(temp);
                  If
!newlist.contains(temp)&&cost ≤ C_i then
                    newlist.add(temp);
                  End if
                End for
              End for
            Return Possible-values(j+1, newlist, k_i, C_i, i);
          End
        Make-to-limit(list, limit)
          %choose randomly limit number of the list
          and return the new list
      End
```

4.6. Top-K Forword Checking 1: TopK-FC

The result of the ranking queries is therefore a set of objects (n-tuples in the relational databases) sorted by score. Each object is represented by an identifier and a score to measure its relevance and similarity to the request. The result of a ranking query is usually all the top k objects, most often those with the highest scores, this set is called top-k, and the query is simply called top-k query. We find a detailed study on this algorithm in [20].

In this method, we used the top-K method in our choice which helps us to get the best K elements from all possible values with highest availability. The domain initialization method is detailed in the following algorithm:

```
  Domain-Initialization (i, Dev_{i,j_i}, k_i, C_i, limit)
  % is the number of the subsystem
  % is the partial cost
    % is the limit for the returned list where the num-
    ber of possible value
  result : list;
  result ← Possible-Values (0, result, k_i, C_i, i);
    if result.size() > limit_i * Dev_{i,j_i} then
      result ← make-to-limit
(result, limit_i, Dev_{i,j_i});
    end if
  return result;
  end
  Possible-values(j, lst, k_i, C_i, i)
    newlist :string ;
    e,temp :string ;
    k :integer;
    cost :float;
    if j=k_i then
      return lst;
    end if
    for each e ∈ lst
      for k from 0 to k_i – 1 do
        temp ← e+ k;
        cost ← cost(temp);
        if !newlist.contain(temp&&cost ≤ C_i then
            newlist.add(temp);
        end if
      end for
    end for
  return Possible-values(j+1, newlist, k_i, C_i, i);
End
Make-to-limit(list, limit_i)
  % choose limit_i number of the list with the highest
  availability and return the new list
End
```

4.7. Generating GA Forword Checking 1: GenGA-FC

In this case, we observed that subsystems can compensate each other for the cost, so it can be that there are subsystems which exceed their partial cost, and others with too low cost, and the sum coincides with the perfect constrained cost, this helps to get solutions with the greatest availability. So we used an algorithm inspired from the step of generation of the initial population in the genetic algorithm. We decomposed the method into two procedures, the first one, gets all possible values for each subsystem under the constraint of the cost C. And the second one, takes the result list of the previous method and for a certain number of iteration, we choose randomly in the domains generated, check if they respect the constraint, if it's true, we add them to the result list.

5. Experimental Results

In this section, we will present and discuss the results found using each method described above.

5.1. Experimentation

We implemented the algorithms above using Java 8 and we run them on an i7 laptop. As inputs, we choose two tables given in the literature in order to be able to compare our results and to prove the performance of our proposed methods.

5.2. Comparisons

To demonstrate this efficiency, we will conduct a comparative study with the best results obtained in the literature [21–22].

Tab. 1. Data of available different power components technologies [21–22]

Subsys-tems	Device Num-ber	Avail-ability	Cost(Mln$)	Perfor-mance (MW) ()
Power Units	1	0.980	0,590	120
	2	0.977	0.535	100
	3	0.977	0.535	100
	4	0.977	0.535	100
	5	0.977	0.535	100
	6	0.977	0.535	100
	7	0.977	0.535	100
HT Transformer	1	0.995	0.205	100
	2	0.996	0.189	92
	3	0.997	0.091	53
	4	0.997	0,056	28
	5	0,998	0,042	21
HT Line	1	0,971	7,525	100
	2	0.973	4,720	60
	3	0,971	3,590	40
	4	0.976	2.42	20
MT Transformer	1	0.977	0,180	115
	2	0.978	0,160	100
	3	0,978	0,150	91
	4	0.983	0.121	72
	5	0.981	0,102	72
	6	0,971	0.096	72
	7	0.983	0,071	55
	8	0.982	0.049	25
	9	0.977	0.044	25
MT Lines	1	0.984	0,986	128
	2	0.983	8,25	100
	3	0.987	0.490	60
	4	0.981	0,475	51

Tab. 2. Parameters of power demand curve [21–22]

Power Demand Level (%)	100	80	50	20
Duration (Hour)	4203	788	1228	2536
Probability	0.480	0.09	0.14	0,290

We will present in table 3 the results of our experimentation.

Tab. 3. Optimal solutions obtained by HS, AC, GA and our approaches EGA1, EGA2, EFC1, EFC2, topK-FC and GenGA-FC

Demand (%)	TopoLogy	Optimale Topology	Method	Cost C (m$)	A
99%	Sub1	4,4,6,7	Har-mony Search HS [21]	13,75	0,992
	Sub2	4,4,4,4, 4,4,4			
	Sub3	1,4			
	Sub4	7,7,7,9			
	Sub5	4,4,4			
99%	Sub1	3,4,4,6,7	Ant colony [22]	14,302	0,9906
	Sub2	5,5,5,5,5,5,4			
	Sub3	1,4			
	Sub4	7,7,7,8,8,9			
	Sub5	3,4,4,4			
99%	Sub1	4,4,6	Geneti Algo-rithm [21,22]	15,87	0,992
	Sub2	3,3			
	Sub3	2,2,3			
	Sub4	7,7,7			
	Sub5	4,4,4			
99%	Sub1	7,7,6,6	EGA1	10,165	0,999154
	Sub2	5,5,5,5,5,3,3			
	Sub3	3,3			
	Sub4	9,6,6,5			
	Sub5	4,3,3			
99%	Sub1	6,6,6,6	EGA2	10,322	0,999116
	Sub2	5,5,4,4,3,3,1			
	Sub3	3,3			
	Sub4	8,8,4,3			
	Sub5	3,3,3			
99%	Sub1	6,6,6,6	EFC1	9,795	0,999111
	Sub2	5,5,5,5,5,5,5			
	Sub3	3,3			
	Sub4	9,9,9,9			
	Sub5	4,4,4			
99%	Sub1	7,7,6,6	EFC2	10,009	0,99915
	Sub2	5,5,5,5,5,4,4			
	Sub3	3,3			
	Sub4	9,9,9,3			
	Sub5	4,4,4			
99%	Sub1	7,7,5,1	TOP-K -FC	11,148	0,999154
	Sub2	5,5,4,3,2,1,1			
	Sub3	3,3			
	Sub4	8,8,8, 4			
	Sub5	4,4,3			

Demand (%)	TopoLogy	Optimale Topology	Method	Cost C (m$)	A
99%	Sub1	6,6,6,6	GA-FC	9,819	0,999111
	Sub2	5,5,5,5,5,5,4			
	Sub3	3,3			
	Sub4	9,9,8,8			
	Sub5	4,4,4			

5.3. Discussion

As shown in Table 3, our methods are better regarding the availability and cost. The topologies obtained through the methods offer more flexibility to the system. In addition, these configurations have an identical choice of components in each subsystem which is a great advantage for designers of components.

6. Conclusion

We presented in this paper different methods for solving optimization problems of redundancy in multi-state systems, those methods based on genetic algorithm and constraints satisfaction including the extensions of the Forward checking algorithm gave the best results in the comparative case we studied, it also allows the verification of the results of the cost and availability obtained. The configurations obtained are simple and homogeneous and rarely varied, the orientation of the selection according to the application is a strong point of those methods. Our work has also helped proving that constraints oriented research improves complexity on one side and allows finding various and high quality solutions.

AUTHORS

Mustapha Es-Sadqi – Intelligent Processing Systems Team, Computer Science Laboratory (LRI), Faculty of Science, Mohammed V University in Rabat, Morocco.

Abdellah Idrissi * – Intelligent Processing Systems Team, Computer Science Laboratory (LRI), Faculty of Science, Mohammed V University in Rabat, Morocco, e-mail: idrissi@fsr.ac.ma.

Ahlem Benhassine – College of Computer Science and Engineering, University of Jeddah, Saudi Arabia.

*Corresponding author

REFERENCES

[1] D. K. Sambariya and R. Prasad, "Design and performance analysis of robust conventional power system stabiliser using cuckoo search algorithm", *International Journal of Power and Energy Conversion*, vol. 8, no. 3, 2017, DOI: 10.1504/IJPEC.2017.084914.

[2] A. H. Bhat, V. Muneer and A. Firdous, "Performance investigation of nine-level cascaded H-bridge inverter-based STATCOM for mitigation of various power quality problems", *International Journal of Industrial Electronics and Drives*, vol. 3, no. 3, 2017, DOI: 10.1504/IJIED.2017.084101.

[3] K.-H. Chang and P.-Y. Kuo, "An efficient simulation optimization method for the generalized redundancy allocation problem", *European Journal of Operational Research*, vol. 265, no. 3, 2018, 1094–1101, DOI: 10.1016/j.ejor.2017.08.049.

[4] M. Heydari and K. M. Sullivan, "An integrated approach to redundancy allocation and test planning for reliability growth", *Computers & Operations Research*, vol. 92, 2018, 182–193, DOI: 10.1016/j.cor.2017.12.013.

[5] A. Shahid, "Power management, intelligent control and protection in micro-grids - a review". In: *2017 International Smart Cities Conference (ISC2)*, 2017, 10.1109/ISC2.2017.8090807.

[6] I. A. Ushakov, "A universal generating function", *Soviet Journal of Computer and Systems Sciences*, vol. 24, no. 5, 1986, 118–129.

[7] M. Es-Sadqi, A. Laghrissi and A. Idrissi, "Reducing carbon footprint in redundancy allocation problem applied to multi-state systems". In: *2016 International Renewable and Sustainable Energy Conference (IRSEC)*, 2016, 1125–1129, DOI: 10.1109/IRSEC.2016.7984038.

[8] A. Amarir, M. Es-Sadqi and A. Idrissi, "An Effective Genetic Algorithm for Solving Series-Parallel Power System Problem". In: *Proceedings of the 2nd international Conference on Big Data, Cloud and Applications*, 2017, 1–6, DOI: 10.1145/3090354.3090377.

[9] A. Laghrissi, M. Es-Sadqi and A. Idrissi, "Solving redundancy allocation problem applied to electrical systems using CSP and forward checking". In: *2016 International Renewable and Sustainable Energy Conference (IRSEC)*, 2016, 1120–1124, DOI: 10.1109/IRSEC.2016.7984036.

[10] M. Es-Sadqi, A. Idrissi and A. Amarir, "An Effective Oriented Genetic Algorithm for solving redundancy allocation problem in multi-state power systems", *Procedia Computer Science*, vol. 127, 2018, 170–179, DOI: 10.1016/j.procs.2018.01.112.

[11] R. K. Saket, B. B. Sagar and G. Singh, "ATM reliability and risk assessment issues based on fraud, security and safety", *International Journal of Computer Aided Engineering and Technology*, vol. 4, no. 3, 2012, DOI: 10.1504/IJCAET.2012.046637.

[12] R. Kumar, "Redundancy effect on coal-fired power plant availability", *International Journal of Intelligent Enterprise*, vol. 3, no. 1, 2015, DOI: 10.1504/IJIE.2015.073458.

[13] R. N. Allan and R. Billinton, *Reliability Evaluation of Power Systems*, Springer, 1996, DOI: 10.1007/978-1-4899-1860-4.

[14] S. Nikolovski, G. Slipac and E. Alibašić, "Generator Assessment of Hydro Power Station Adequacy after Reconstruction from Classical to HIS SF6 Substation", *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 7, no. 2, 2017, 729–740, DOI: 10.11591/ijece.v7i2.pp729-740.

[15] A. Lisnianski and G. Levitin, *Multi-State System Reliability: Assessment, Optimization and Applications*, World Scientific, 2003, 10.1142/5221.

[16] G. Levitin, *The Universal Generating Function in Reliability Analysis and Optimization*, Springer-Verlag, 2005, DOI: 10.1007/1-84628-245-4.

[17] P. van Hentenryck, *Constraint satisfaction in logic programming*, MIT Press, 1989.

[18] M. Meier and J. Schimpf, "ECL^{i}PS^e, ECRC common logic programming system, User manual". *Technical Report TTI/3/93*, 1993.

[19] A. Darehshoor, L. Zadeh, L. Cerdà-Alabern and V. Pla, "Candidate selection algorithms in opportunistic routing based on distance progress", *International Journal of Ad Hoc and Ubiquitous Computing*, vol. 20, no. 3, 2015, DOI: 10.1504/IJAHUC.2015.073168.

[20] G. Li, X. Gao, M. Liao and B. Han, "An iterative algorithm to process the top-k query for the wireless sensor networks", *International Journal of Embedded Systems*, vol. 7, no. 1, 2015, DOI: 10.1504/IJES.2015.066139.

[21] A. Zeblah, E. Chatelet, M. El Samrout, F. Yalaoui and Y. M., "Series-parallel power system optimisation using a harmony search algorithm", *International Journal of Power and Energy Conversion*, vol. 1, no. 1, 2009, 15–30, DOI: 10.1504/IJPEC.2009.023474.

[22] M. Nourelfath and D. Ait-Kadi, "Optimization of series–parallel multi-state systems under maintenance policies", *Reliability Engineering & System Safety*, vol. 92, no. 12, 2007, 1620–1626, DOI: 10.1016/j.ress.2006.09.016.