MICHAEL K. KONTESSIS

MULTIOBJECTIVE TECHNIQUES FOR OPTIMUM PARALLEL REDUNDANCY WITH SEVERAL COST FUNCTIONS - A TRADE-OFF OF FUNCTION APPROACH
CHAPTER 1

RELIABILITY OF A SYSTEM

1.1 Introduction and System Reliability Definition

Modern life is relied on a large number of types of machines and instruments. The term "system" is introduced in order to describe more generally a set of components built in some configuration satisfying one or more tasks. However, a system designed to accept some inputs and to derive some desirable outputs in time is accompanied by the uncertainty of its adequate operation which means that a system failure at a moment in time must be considered quite probable. This is what the experience points out. Many real world examples could be arranged varying between the simple electrical bulb system and the mostly sophisticated spacecraft one.

Therefore, for any reason out of our control, a system inadequate operation may happen at any time and thus problems about our confidence in this system arise. Of course, we would like to measure this confidence. Statistics enables us to do so and it can be done using the concept of the system reliability, that is "the probability of a system performing its purposes adequately for the period of time intended under the operating conditions encountered" (Radio-Electronics Television Manufacturers Association, 1955).

Thus, the reliability of a system is a function of the time and of the conditions of its operation. For example, an old in use light bulb is not as reliable as one that has recently been put into service and, also, a system is more reliable functioning under normal conditions of its environment.

Barlow and Proschan (1965) give a functional formula describing the reliability of a system, as it has been defined above. For, let the period of time intended is \( t \) (from 0 to \( t \)). The random variable \( T \) will represents the life of the system. For \( T = t \) we define a state variable
X(t) taking the value 1 if the system functions adequately at the time t and taking the value 0 otherwise. Assuming that adequate operation at time t implies adequate performance during the period of time between 0 and t we can write

\[
\text{System Reliability in } [0,t] = \text{Prob}(X(t) = 1) \tag{1.1}
\]

Thus, we can summarize at this point that:

a) Reliability is a function of time. For example, an almost worn-out electrical bulb would be not as liable as a new in use one.

b) Reliability is a function of conditions of use or environment. For example, our electrical bulb is not going to function very reliably if we hit it with a hammer.

1.2 Failure Distributions Related to Reliability

Giving the definition of a system in the foregoing section 1.2 we have describe it as a set of components. For example, a usual computer system could be considered as the set of the components: Card Reader Unit, C.P.U, Line Printer Unit, Disk Drive Unit and Magnetic Tape Unit. Therefore, in calculating the reliability of a system the failure distributions of its particular components are of essential interest. Denoting the time to failure of a component with T (T is a random variable) the failure distributions listed below are very important giving information about the component life.

a) Probability Distribution of the Time to Failure.

\[
\text{Prob}(t \leq T \leq t + dt) = f(t)dt, \quad t \geq 0 \tag{1.2}
\]

The function f(t) derives at each time the probability of component failing between the time t and t + dt.

b) Cumulative Probability Distribution of Time to Failure.

\[
\text{Prob}(T \leq t) = F(t) = \int_0^t f(t)dt, \quad t \geq 0 \tag{1.3}
\]

The function F(t) gives the probability that a component has failed by the time t. The equation 1.3 gives the function F(t) in terms of the function f(t). Conversely, on differentiating 1.3 we find

\[
f(t) = F'(t) \tag{1.4}
\]
c) Survival function.

\[ \text{Prob}(T > t) = S(t) = 1 - F(t) = \int_t^\infty f(t) \, dt, \quad t \geq 0 \quad (1.5) \]

The function \( S(t) \) often arises very naturally, for example in considering the probability that a component will survive a guarantee period. One can easily verify that

\[ f(t) = -S'(t) \quad (1.6) \]

d) Failure Rate.

\[ \text{Prob}(t \leq T \leq t + dt / T > t) = r(t)dt = \frac{f(t)}{S(t)} \, dt \quad (1.7) \]

Roughly speaking \( r(t) \) gives the probability of almost immediate failure of a component known to be of age \( t \). The failure rate \( r(t) \) is related to the survival function via probability distribution \( f(t) \)—formula 1.7—but a direct relation between them can be derived and it is:

\[ S(t) = \exp \{- \int_0^t r(t) \, dt \}, \quad t \geq 0 \quad (1.8) \]

The concept of the failure rate is important, both theoretically and in practice and gives information about the life of a component according to the behaviour of the function \( r(t) \). Thus, constant failure rate means that the probability of component failing next moment is constant independently of the age of the component. Increasing \( r(t) \) means that the same probability increases as the component becomes older. Finally, decreasing failure rate means that this probability decreases as the component age increases. However, this last case is not often occurred in practice.

Figure 1.1 and figure 1.2 illustrate two examples each for constant failure rate and proportional to \( t \) one respectively related with their own survival functions.

1.3 Some of the most common in use Failure Distributions

There exists a variety of families of distributions that could be well
used in order to describe the life of a component. The listing below contains those theoretical distributions that are often in use in the practical applications.

\[ z = S(t) = e^{-\lambda t} \]

(Constant Failure Rate)

\[ z = r(t) = \lambda \]

\[ z = S(t) = e^{-\left(\frac{\lambda}{2}\right)t^2} \]

(Proportional to t Failure Rate)

\[ f(t) = \lambda e^{-\lambda t} \quad , \quad \lambda > 0 \quad , \quad t \geq 0 \]

(1.9)
with:

\[
\text{Failure rate } r(t) = \lambda \\
\text{mean } \mu = \frac{1}{\lambda} \\
\text{variance } \sigma^2 = \frac{1}{\lambda}
\]

Such a failure density appears when the component failures occur randomly and independently. For example, this is the case of the failure occurrence of certain types of electronic devices.

b) Gamma failure distribution.

\[
f(t) = \lambda (\alpha t)^{\alpha - 1} \Gamma^{-1}(\alpha) \exp(-\lambda t), \quad \alpha, \lambda > 0, \quad t \geq 0 \quad (1.10)
\]

with:

\[
\text{failure rate } r(t) = f(t) \cdot \left\{1 - \int_0^t f(t)dt\right\}^{-1} \quad (1.10')
\]

\[
\text{mean } \mu = \alpha \lambda^{-1} \\
\text{variance } \sigma^2 = \lambda^{-1} \sqrt{\alpha}
\]

When \( \alpha = 1 \) the Gamma failure density gives the exponential one.

c) Weibull failure distribution.

\[
f(t) = \lambda \alpha t^{\alpha - 1} \exp(-\lambda t^\alpha), \quad \lambda, \alpha > 0 \quad t \geq 0 \quad (1.11)
\]

with:

\[
\text{failure rate } r(t) = \lambda \alpha t^{\alpha - 1} \\
\text{mean } \mu = \lambda^{-1} \Gamma(1 + 1/\alpha)
\]

For \( \alpha = 1 \) we again find the exponential distribution from 1.11. The Weibull density will be a suitable model for a situation in which all components on test fail around the same time.

d) Modified extreme value failure distribution.

\[
f(t) = \lambda^{-1} \exp\left(-\frac{e^t - 1}{\lambda} + t\right), \quad \lambda > 0, \quad t \geq 0 \quad (1.12)
\]

with:

\[
\text{failure rate } r(t) = \lambda^{-1} e^t
\]
e) Truncated normal failure distribution.

\[
f(t) = (\alpha \sqrt{2\pi})^{-\frac{1}{2}} \exp \left\{ -\frac{(t - \mu)^2}{2\sigma^2} \right\}, \quad \sigma, \mu > 0, \quad t \geq 0 \quad (1.13)
\]

where \(\alpha\) is a normalizing constant and the failure rate is calculated using a formula similar to the formula 1.10'.

f) Log-normal failure distribution.

\[
f(t) = (t\sigma \sqrt{2\pi})^{-\frac{1}{2}} \exp \left\{ -\frac{(\log t - \mu)^2}{2\sigma^2} \right\}, \quad \sigma > 0, \quad \mu > 0, \quad t \geq 0 \quad (1.14)
\]

with failure rate being calculated by the formula 1.10'.

All densities a) to e) possess a monotonic failure rate (either increasing or decreasing) and therefore can be considered suitable for handling failure rate data due to wear and tear. Log-normal, in the other hand, has a non-monotonic failure rate which possesses a maximum value, decreasing then to zero and for this reason it is not a good distribution for representing failure data.

For the purposes of the present work the component reliabilities of a complex system are assumed known and it will be valid everywhere in the subsequent pages.

### 1.4 System Reliability Calculation

Before passing to the discussion of the main subject of this work, which is the reliability improvement, a review of the techniques for reliability calculation of various types of systems in terms of the given component reliabilities is of practical interest.

Whatever the patricular system under consideration it is assumed that its components fail independently subjected to their own reliability laws.

#### 1.4.1 Series and Parallel Systems

Consider a system consisting of \(k\) components and we assume that the failure of at least one component causes the failure of the entire system. Such a system is called «series - system».
Relating the electrical current flow through a series-electrical-circuit with the reliability flow through the components of a series-system, we can easily see that a similar to the first case network could be drawn in order to give a graphical representation of the second case. Thus, for every series-system a reliability network is associated describing the logic of failure of this system. The figure 1.3 illustrates a k-component series system.

![Figure 1.3](image)

Given the reliabilities \( p_i \), \( i = 1, 2, \ldots, k \) of the particular components (that is the probabilities of the component survival during a specified period of time, assuming that at \( t = 0 \) all components are new) we can calculate the over-all system reliability in terms of the component reliabilities using simply the probability multiplication law:

\[
R = \prod_{i=1}^{k} p_i = \prod_{i=1}^{k} (1 - q_i)
\]  \( (1.15) \)

where \( q_i = 1 - p_i \) are the component unreliabilities.

Let us now consider another situation in which the configuration of the system is such that it fails if and only if all the components fail. This is what we call "parallel-system" and in a similar to series systems way its reliability logic can be represented by a reliability network as it is illustrated in figure 1.4 corresponding to an n-component parallel-system.

Given the reliabilities \( p_j \), \( j = 1, 2, \ldots, n \) of the components of an n-component parallel-system we calculate the probabilities \( q_j = 1 - p_j \) of the component failures; consequently, by the probability multiplication law, the probability of the entire system failing is

\[
Q = \prod_{j=1}^{n} (1 - p_j) = \prod_{j=1}^{n} q_j
\]  \( (1.16) \)
and therefore the over-all system reliability is:

\[ R = 1 - \prod_{j=1}^{n} (1 - p_j) = 1 - \prod_{j=1}^{n} q_j \]  \hspace{1cm} (1.17)

![Diagram of system reliability](image)

*Figure 1.4*

Of particular interest is the case in which all components in parallel have equal reliabilities. Such a case arises when all components are identical. The formula 1.17 is reduced then to:

\[ R = 1 - (1 - p)^n = 1 - q^n \]  \hspace{1cm} (1.18)

where \( p \) is the common component reliability.

1.4.2 *Parallel-Series Systems*

A simple system, mostly studied in its reliability behaviour, is the "parallel-series" one which is a series combination of parallel subsystems.
Figure 1.5 illustrates a parallel-series system structure with \( k \) subsystems (stages) in series and each subsystem \( i \) consists of \( n_i \) components in parallel.

![Diagram showing parallel-series system structure]

Given the reliabilities \( p_{ij} \) of the components \( j = 1, 2, ..., n_i \) in each subsystem \( i = 1, 2, ..., k \) we can calculate the overall reliability taking

\[
R_i = 1 - \prod_{j=1}^{n_i} (1 - p_{ij}) = 1 - \prod_{j=1}^{n_i} q_{ij} \tag{1.19}
\]

the reliability of each subsystem \( i \) and then

\[
R = \prod_{i=1}^{k} R_i = \prod_{i=1}^{k} (1 - \prod_{j=1}^{n_i} (1 - p_{ij})) = \prod_{i=1}^{k} (1 - \prod_{j=1}^{n_i} q_{ij}) \tag{1.20}
\]

the desirable overall reliability, where

\[
q_{ij} = 1 - p_{ij}
\]

is the unreliability of the \( j^{th} \) component in the \( i^{th} \) subsystem.

Of interest is also the special case where all the components in the subsystems are equally reliable that is when

\[
p_{ij} = p_i, \quad j = 1, 2, ..., n_i
\]

The formulas 1.19 and 1.20 become respectively
\[ R_i = 1 - (1 - p_i)^{n_i} = 1 - q_i \]  
\[(1.21)\]

and

\[ R = \prod_{i=1}^{k} \left( 1 - (1 - p_i)^{n_i} \right) = \prod_{i=1}^{k} (1 - q_i) \]  
\[(1.22)\]

The reliability improvement via parallel redundancy going to be discussed in next Chapter is directly related with this situation.

1.4.3. More Complicated Systems

When a system appears in a more complicated configuration a direct calculation of its reliability, based on the probability theory laws, becomes laborious.

In such a case an indirect method for calculating system reliability is discussed by Caufman (1972) which is considered more efficient than the direct one. This method is built on the concept of the "system structure function".

In order to define this function for a given system we introduce the "component state variables" setting

\[ x_i = 1 \quad \text{if the } i^{th} \text{ component functions} \]
\[ x_i = 0 \quad \text{otherwise} \]

The structure function depends on the state variables \(x_1, x_2, \ldots, x_k\) (if \(k\) components form the system) with the requirement to be equal to 1 when the entire system functions and equal to 0 otherwise, that is:

\[ f(x_1, x_2, \ldots, x_k) : \{0,1\}^k \rightarrow \{0,1\} \]  
\[(1.23)\]

The set \(\{0,1\}\) is assumed to be equipped with the usual real numbers algebra. The way in order to derive an analytical formula for the function \(f\) of a given system is explained below where some general examples are given.

**Series Systems**

In such a system it is not difficult to verify that its structure function \(f(x_1, x_2, \ldots, x_k)\) must be equal to 1 if the state vector \((x_1, x_2, \ldots, x_k)\)
equals to \((1,1,...,1)\) and 0 otherwise. One can immediately derive the formula of the function \(f\) to be

\[
f(x_1,x_2,...,x_k) = \prod_{i=1}^{k} x_i = x_1 \cdot x_2 \cdot ... \cdot x_k
\]

(1.24)

and to justify that this function equals to 1 when \((x_1,x_2,...,x_k) = (1,1,...,1)\) while it is zero otherwise.

**PARALLEL SYSTEMS**

In this situation we are looking for a function \(f\) taking the value 0 when the state vector equals to \((0,0,...,0)\) and taking the value 1 otherwise. A function verifying this requirement is

\[
f(x_1,x_2,...,x_k) = 1 - \prod_{i=1}^{k} (1 - x_i)
\]

(1.25)

**PARALLEL-SERIES SYSTEMS**

Another system configuration in which it is rather easy to calculate its structure function is that of a parallel-series one. Combining the formulas 1.24 and 1.25 we obtain the structure function of such a system (illustrated in figure 1.5) which is:

\[
f(x_{11},...,x_{1n_1},...,x_{kn_k}) = \prod_{i=1}^{k} \left\{ 1 - \prod_{j=1}^{n_i} (1 - x_{ij}) \right\}
\]

(1.26)

**SERIES-PARALLEL SYSTEMS**

This is the situation illustrated in figure 1.6 representing a system consisting of \(k\) series subsystems in parallel. Each subsystem \(i\) contains \(n_i\) components in series. For this system we can calculate its structure function and it is:

\[
f(x_{11},...,x_{1n_1},...,x_{kn_k}) = 1 - \prod_{i=1}^{k} \left( 1 - \prod_{j=1}^{n_i} x_{ij} \right)
\]

(1.27)

Note that the component state variables \(x_{ij}\) in the formulas 1.26 and
1.27 have been defined so that \( x_{ij} = 1 \) if the \( j \)th component of the \( i \)th subsystem functions and \( x_{ij} = 0 \) otherwise.

![Diagram](image)

**Figure 1.6**

Having on hand the structure function of a certain system we can achieve its reliability in terms of the component reliabilities following the steps of the procedure below:

**PROCEDURE 1.1**

a) Reduce the structure function in its polynomial form performing the operations denoted in it.

b) Set \( x_{a-i} = x_i \) (since \( x_i = 0 \) or 1) for every \( a \geq 1 \)

c) Replace the component state variables \( x_i \) by the component reliabilities \( p_i \). The resulted expression of \( p_i \) gives the desirable system reliability in terms of the component reliabilities.

Kaufman (1972) gives a number of numerical examples of this procedure. In the general nature examples presented in subsections 1.4.1 and 1.4.3 the system reliabilities have been derived on the basis of the probability laws. Alternately, we obtain the relevant formulas 1.15, 1.17 and 1.20 from the structure functions 1.24, 1.25 and 1.26 respectively setting directly \( p_i \) in the place of \( x_i \) since no power \( x_{a-i} \) is going to appear performing the operations. Similarly, we obtain the reliability of a parallel-series system from the formula 1.27 and it is:

\[
R = 1 - \Pi_{i=1}^{k} (1 - \Pi_{j=1}^{n_i} p_{ij}) = 1 - \Pi_{i=1}^{k} \left( 1 - \Pi_{j=1}^{n_i} \{1 - q_{ij}\} \right) \quad (1.28)
\]
CHAPTER 2

RELIABILITY OPTIMIZATION

2.1 Introduction to the Redundancy

A certain system consisting of a given number of components possesses a reliability level depending on the reliabilities of its particular components.

In Chapter 1 the general methodology of calculating the system reliability in terms of the component reliabilities has been discussed. In the present Chapter a theoretical review is going to be presented giving an answer to the following problem:

PROBLEM 2.1

"If the system reliability is regarded too low for our aims under consideration what can we do in order to improve it? What are the effects on the system reliability and on certain cost functions related with it if a change of the system configuration is made in order to obtain higher reliability level?".

An obvious answer to the first part of the problem is to use components of greater reliability. However, there are particular situations in which the effects of such a technique on the over-all reliability may be regarded very poor. In some other cases the component reliabilities are not under our control. Gordon (1956) gives an idea of the poorness of the effects on the system reliability when we increase the component reliability. He considers an original series system consisting of k components whose the reliabilities are taken to be equal to each other, say p. Thus the over-all reliability is given by the formula 1.15 which in this case is reduced to

\[ R = p^k \]  

(2.1)
The function $p^k$ (since $0 < p < 1$) tends to zero when $k$ becomes large. Thus taking $k$ to be large the component reliability must be very close to unity if the system reliability is not to be regarded so low that the system is worthless.

In addition to poor effects on the over-all reliability, higher component reliability could require a higher component production cost or increase the component weight, volume, e.t.c. and the balance between higher over-all reliability and higher cost functions may be regarded unsatisfactory.

According to what has been up to now discussed, the system reliability improvement via higher component reliability is not the best way in many cases. A more convenient method that is not related with the component reliabilities but with the number of components in the system is what we call "Redundancy". With this technique spare components are added to each basic component of the system in such a number that increases the over-all reliability to a desirable level.

Redundancy can be applied in two ways:

**Parallel Redundancy**

To each basic component a number of $n$ similar spare components is added and all $n + 1$ components of the system operate at the same time.

**Stand-by Redundancy**

To each basic component a number of an similar spare components is added. Only one component is in function at a time while a spare component is put into operation as soon as the functioning one fails.

This work on hand deals with the application of the multiobjective programming techniques on the problem of parallel redundancy optimization. Therefore, the parallel redundancy effects on the over-all reliability is the main subject of the subsequent sections. However, most of which is discussed in the next pages is applicable to stand-by redundancy as well, if some changes on the calculation of the over-all reliability are made.

2.2 Parallel Redundancy Effects on the Over-all Reliability

According to the definition of the parallel redundancy, given in the section 2.1, the set of one basic component $i$ and the spare ones $n_i$ could
be regarded as a parallel type system whose all \( n_i + 1 \) components are identical possessing equal to each other reliabilities, say \( p \). Therefore the reliability of this set is calculated by the formula 1.18, that is:

\[
R_i = 1 - \left(1 - p\right)^{n_i+1} = 1 - q
\]

(2.2)

Assuming \( n_i \) to be independent variable, the reliability function \( R_i(p,n_i) \) is represented by a single—parameter family of curves on the \((R_i,n_i)\)—plane and its behaviour is illustrated in the figure 2.1. Clearly, as the component reliability increases, the number \( n_i \) of the redundant components in parallel decreases when the over-all reliability remains fixed. Solving the equation 2.2 with respect to \( n_i \) we express the number of redundant components as a function of the given component reliability and a specified over-all reliability \( R_i \), that is:

\[
n_i = \left[1 - \frac{\log(1 - R_i)}{\log(1 - p)}\right]
\]

(2.3)

where \([x]\) denotes the biggest integer less than \( x \).

Consider now the entire system consisting of \( k \) basic components in series. Let \( p_1,p_2,...,p_k \) be the reliabilities of these basic components. Thus the system reliability is:

\[
R' = \prod_{i=1}^{k} p_i
\]

(2.4)

Parallel redundancy can be applied to this system in two different ways. Firstly, we can duplicate the entire system in a sense illustrated by the figure 2.2. A new system is then obtained and its overall reliability is calculated using the formula 1.28, that is:

\[
R'' = 1 - (1 - R')^2 = 1 - \left(1 - \prod_{i=1}^{k} p_i\right)^2
\]

(2.5)

It is clear that

\[
R'' > R'
\]

(2.6)
and in this way we reach a higher over-all reliability duplicating the entire system.

Figure 2.1

Secondly, we can duplicate the particular system components obtaining heretofore a parallel-series new system of which the configuration is illustrated in figure 1.5. Thus the over-all reliability of this system is given if we set in the formula 1.22 \( n_i = 2 \), that is;
\[ R'''' = \prod_{i=1}^{k} \{1 - (1 - p_i)^2\} \]  

Comparing now \( R''' \) and \( R' \), as it has been done for the first case, we see that:

\[ p_i > p_i^2 \quad \text{or} \quad p_i < 2p_i - p_i^2 \quad \text{or} \quad p_i < 1 - (1 - p_i)^2 \quad \text{or} \]

\[ \prod_{i=1}^{k} p_i < \prod_{i=1}^{k} \{1 - (1 - p_i)^2\} \]

which means that:

\[ R'''' > R' \]  

that is, the original system reliability has been again improved. Finally, between the two methods of applying the parallel redundancy, being described above, the second one is the most effective since from 2.4 and 2.6 we can find:

\[ R'''' > R'' \]  

while all cost functions (i.e., system cost, weight, etc.) related to the system remain unchanged.

The result from the previous comparison, that is duplicating each system component is more effective than duplicating the entire system, is of a more general application valid for any type of original system even though it is not a series one, as it was before. This is what Kaufmann, Cryon and Grouchko (1969) and Kaufmann (1972) point out.

2.3 The Problem of Parallel Redundancy Optimum Allocation

Let the original system be one consisting of \( k \) basic components. Let

\[ \mathbf{n} = (n_1, n_2, \ldots, n_k) \]  

be the vector whose the entries \( n_i = 0,1,2,\ldots \) represent the number of
the redundant components added to each basic component \( i = 1, 2, \ldots, k \). Thus the over-all reliability is a function

\[
z = R(n) = R(n_1, n_2, \ldots, n_k)
\]  

(2.11)

which is increasing with respect each \( n_i \) assuming the basic component reliabilities \( p_i \) to be fixed. This means it is possible to make the system reliability as high as it is desirable adding redundant components.

It is clear that any redundant component added to the basic components of the system increases the system cost and/or weight and/or volume etc. together with the over-all reliability. Generally speaking, a system reliability increasing causes changes to some cost functions related to. Let these cost function be:

\[
h_j(n) = h_j(n_1, n_2, \ldots, n_k)
\]  

(2.12)

for \( j = 1, 2, \ldots, r \). Since these resources are normally limited by upper bounds the over-all reliability cannot be as high as we would like. An of the earliest and mostly important problem in the reliability theory is to find the optimum redundancy allocation, that is the allocation giving the maximum value of the over-all reliability subject to limitations on the cost functions.

In terms of mathematical programming this problem is stated as follows:

\[
\text{maximize: } z = R(n) \\
\text{subject to: } h_j(n) \leq b_j, \quad j = 1, 2, \ldots, r \\
n_i = 0, 1, 2, \ldots
\]  

(2.13)

A wide number of various observations of this problem are made and a similar number of solving algorithms has been developed according to particular assumptions, definitions and the methodology which is followed.

The crucial points making differences between the particular cases of treating the problem 2.13 could be listed as follows:

i) The type of the original system which may be a series one or of some other configuration.

ii) The number of cost functions being 1 or 2,... or \( r \).
iii) The type of the cost functions $h_j$ which could be linear or not.
iv) The particular area of mathematical programming in which the problem of redundancy optimization is embedded, say Dynamic Programming, Integer Programming, Geometric Programming, e.t.c.

In the subsequent sections of this Chapter a theoretical review of the methodology followed in solving the problem 2.13 is discussed. This discussion will be restricted in the presentation of the various solving algorithms where they are not of particular interest through the pages of this dissertation. However, a detailed description will be neccessary in presenting the algorithms that are going to be applied in Chapter 3.

2.4 Reliability Maximization under Constraints

2.4.1 Series Systems; Single Constraint

A first treatment of the problem 2.13 with a single constraint, $r = 1$, for series systems has been developed by Moskovitz and McLean (1956) and Mine (1956) using Lagrange multiplier method. In applying this technique the variables $n_i$ are assumed to be continuous, but they actually take only integer values. The obtained solutions are therefore approximated.

Bellman and Dreyfus (1962) give a Dynamic Programming approach to the same problem where the single constraint represents a linear cost function. In mathematical terms the particular problem solved by the authors is the following:

$$\text{maximize: } R(n) = \prod_{i=1}^{k} (1 - q_i^{n_i-1})$$

$$\text{Subject to: } \sum_{i=1}^{k} c_i n_i \leq b$$

$$n_i = 0,1,2,... \text{ for all } i = 1,2,...,k$$

where $c_i$ is the cost of one component of type $i$, $n_i$ is the number of redundant components added to the basic component $i$. The set of all $n_i + 1$ identical components is called "stage $i". Finally, $b$ is the budget that the total redundancy cost cannot exceed. The relevant recurrence equation is:
\[ f_t(b) = R([b/c_1]) \] (2.15)

\[ f_N(b) = \max \{ R_N(n_N) \cdot f(b - n_N c_N) \} \quad , \quad N = 2,3,\ldots,k \] (2.16)

where \( f_N(b) \) represents the value of the objective function \( R(n) \) obtained using an optimal policy while

\[ n_N c_N \leq b \] (2.17)

\( R_N(n_N) \) is the reliability of the stage \( N \) if \( n_N \) redundant components are added and \([x]\) denotes the biggest integer less than \( x \).

Barlow and Proschan (1965) describe an algorithm solving the same problem based on the concept of the undominated redundancy allocation. This procedure (refered as "procedure I" by the authors) generates a family of "undominated allocations", that is each member has the property that any other allocation giving higher reliability than a member under consideration must need higher cost. In the strict mathematical notation an undominated redundancy allocation is defined as follows:

\[ \mathbf{n} \text{ is undominated} \quad \text{for every } \mathbf{n}' \neq \mathbf{n} \quad \text{then,} \quad \text{def} \]

\[ R(\mathbf{n}') > R(\mathbf{n}) \quad \text{for} \quad \sum_{i=1}^{k} c_i n_i' > \sum_{i=1}^{k} c_i n_i \quad \text{and,} \]

\[ R(\mathbf{n}') = R(\mathbf{n}) \quad \text{for} \quad \sum_{i=1}^{k} c_i n_i' \geq \sum_{i=1}^{k} c_i n_i \]

The relevant algorithm generating undominated allocations is the following:

**Algorithm 2.1 (Procedure I)**

a) Start with the cheapest cost allocation \( \mathbf{0} = (0,0,\ldots,0) \) which is always undominated.

b) If the present allocation is \( \mathbf{n} \) obtain a next allocation \( \mathbf{n}' \) by adding one component to the stage \( i \) for which the quantity:
\[
q_i = \frac{1}{c_i} \cdot \{\log R_i(n_i + 1) - \log R_i(n_i)\}
\] (2.19)

becomes maximal. This means that we add one component to that stage \(i\) for which the over-all reliability increasing per unit of money spent is maximum.

c) Repeat the step b) until a specified cost bound is exceeded.

The main disadvantage of this procedure is that it derives an incomplete family of redundancy allocations, as it is emphasized by the authors, and this family contains undominated members under the additional condition that \(\log R(n)\) must be concave. In the particular problem 2.14 in which we are interested in the present subsection this requirement is valid since

\[
\log R(n) = \sum_{i=1}^{k} \log R_i(n_i)
\]

and since the considered system is a series one we have

\[
\log R_i(n_i) = \log(1 - q_i^{n_i+1})
\]

for all \(i = 1,2,...,k\), which means that \(\log R_i(n_i)\) are concave and, therefore, \(\log R(n)\) is also concave.

In order to obtain a complete family of undominated redundancy allocations for the same problem 2.14, Kettle (1962) has developed an algorithm based on the Dynamic Programming formulation. This procedure derives undominated redundancy allocations for larger subsystems from undominated allocations for smaller ones. A step-by-step description of this algorithm could be stated as follows:

**Algorithm 2.2**

a) Consider the subsystem consisting of the stages 1 and 2 only.

b) Start with the first undominated allocation (0,0) for this subsystem (1,2).

c) Obtain the next undominated redundancy allocation choosing the cheapest cost allocation with reliability no lower than the one just achieved. If there exist two allocations of identical cost choose the one
with the highest reliability. If there exist two allocations with identical cost and highest reliability choose the one with the lowest component index. If the present allocation is at the intersection of the row $i$ and column $j$ search for the next allocation may be confined to the union of rows $1, 2, ..., i$ and columns $1, 2, ..., j$, where $j$ is the number of redundant components in stage 1 and $i$ the number in stage 2.

d) Repeat the steps a) to c) for the next subsystem $(2,3)$ consisting of the stages 3 and 4 only.

e) Repeat the steps a) to c) combining the two subsystems $(1,2)$ and $(3,4)$. Thus, a complete family of undominated redundancy allocations for the subsystem $(1,2,3,4)$ has been achieved.

f) In this way, obtaining a complete family of undominated redundancy allocations for the next two-stage subsystem and combining it with the subsystem achieved just before, we can derive a complete family of allocations for the entire subsystem.

A schematical representation of such a subsystem combination for a six-stage original system is illustrated in figure 2.3.

![Figure 2.3](image-url)
2.4.2 Series Systems; Multiple Constraints

In this case, with the additional assumption that there are only two constraints, say linear cost and linear weight, Bellman and Dreyfus (1958) have developed a Dynamic Programming procedure combined with Lagrange multiplier method in order to solve the following problem in mathematical terms:

\[
\text{maximize: } R(n) = \prod_{i=1}^{k} \left( 1 - q_i \right)^{n_i+1}
\]

Subject to:
\[
\sum_{i=1}^{k} c_{ij} n_i = c_j(n) \leq b_j, \quad j = 1, 2, \ldots, r
\]
\[
n_i = 0, 1, 2, \ldots \quad , \quad i = 1, 2, \ldots, k
\]

for \( r = 2 \), where \( c_{i1} \) and \( c_{i2} \) are positive numbers representing the cost and the weight of the \( i^{th} \) component in the stage 1 and 2 respectively. Setting now:

\[
R_i(n_i) = 1 - q_i
\]

\[
f_N(b_j) = \max_{S_N^N} \prod_{i=1}^{N} \{R_i(n_i) \cdot \exp(-\lambda c_{i2} n_i)\}
\]

where \( S_N \) is the set of solutions defined by the constraints in 2.20 and \( \lambda \) is a Lagrange multiplier such that the weight constraint is satisfied, an easy in calculations forward recurrence equation is obtained:

\[
f_1(b_1) = \max_{n_1} R_1(n_1) \cdot \exp(-\lambda c_{12} n_1)
\]

\[
f_N(b_1) = \max_{n_N} \{R_N(n_N) \cdot \exp(-\lambda c_{N2} n_N) \cdot f_{N-1}(b_1 - c_{N1} n_N)\} , \quad N \geq 2
\]

The maximum in the later formula is taken over all \( n_N \) such that:

\[0 \leq n_N \leq \lfloor b_1 / c_{N1} \rfloor\]

For various values of \( \lambda \) we calculate solutions to the problem 2.20 choosing that gives the highest weight less than \( b_2 \).
Another approach to the problem 2.20, given in its general form with \( r \) linear constraints, is referred by Barlow and Proschan (1965). Two different procedures (called "procedure 1 for multiple cost factors" and "procedure 2" respectively) are discussed both generating undominated allocations for the problem 2.20. In a similar way to the section 2.4.1 the definition of an undominated redundancy allocation when multiple linear cost factors are involved is stated as follows:

\[
\mathbf{n} \text{ is undominated } \quad \text{for every } \mathbf{n}' \neq \mathbf{n} \quad \text{then}, \quad \text{def}
\]

\[
\begin{align*}
R(\mathbf{n}') > R(\mathbf{n}) & \quad \Rightarrow \quad c_j(\mathbf{n}') > c_j(\mathbf{n}) \quad \text{for at least one } j \\
R(\mathbf{n}') = R(\mathbf{n}) & \quad \Rightarrow \quad c_j(\mathbf{n}') > c_j(\mathbf{n}) \quad \text{for at least one } j \quad \text{or} \\
& \quad c_j(\mathbf{n}') = c_j(\mathbf{n}) \quad \text{for all } j = 1, 2, \ldots, r
\end{align*}
\]

where the expression of the cost factors \( c_j(\mathbf{n}) \) is given in the problem 2.20. On the basis of this definition the two algorithms described below derive families (generally incomplete) of undominated redundancy allocations for the problem 2.20.

**Algorithm 2.3 (Procedure 1 for multiple cost factors)**

a) Start with the cheapest allocation \((0, 0, \ldots, 0)\) which is always undominated.

b) Obtain a next allocation by adding one component to the stage \( i \) for which the quantity:

\[
q_i = \left\{ \sum_{j=1}^{r} \alpha_j c_{ij} \right\}^{-1} \cdot \left\{ \log R(n_i + 1) - \log R(n_i) \right\} \quad (2.22)
\]

is maximum, where \( \alpha_j \) are non-negative real numbers summing to unity.

c) Repeat the step b).

The meaning of the step b) is that we add one component to that stage \( i \) for which the rate of the over-all reliability increasing and a linear convex combination of the corresponding cost factors to be maximum. The quantities \( \alpha_j \) are chosen in the interval \((0, 1)\) and to each vector \( \alpha = (\alpha_1, \alpha_2, \ldots, \alpha_r) \) one family of undominated allocations is corresponded. If we repeat the algorithm 2.3 for various vectors \( \alpha \), say starting from the vector \((1, 0, \ldots, 0)\) and proceeding until the final vector \((0, 0, \ldots, 0, 1)\) varying the \( \alpha_j \) by some fixed increment of \( \Delta \), we obtain a wider family.
Even if all the possible vectors $\alpha$ have been taken the family resulted in this way is still incomplete.

**Algorithm 2.4**

Let $\lambda = (\lambda_1, \ldots, \lambda_r)$ be a vector of which each $\lambda_j \geq 0$ and at least one $\lambda_j > 0$. For $i = 1, 2, \ldots, k$ obtain $n_i(\lambda)$ as the smallest integer $m$ satisfying the inequality:

$$\log R_i(m + 1) - \log R_i(m) < \sum_{j=1}^{r} \lambda_j c_{ij}$$

(2.23)

This procedure, and the foregoing one, generates an incomplete family of undominated allocations (since $\log R(n)$ is concave in the problem 2.20).

Proshcan and Bray (1965) have developed a generalized Kettelle algorithm in order to treat the problem 2.20 in its general case, i.e., when $r$ cost factors are involved.

Fedorowicz and Mazumdar (1968) use Geometric Programming Techniques in order to solve the problem 2.20. The corresponding Geometric Program is obtained setting:

$$x_i = n_i$$

and

$$\ln q_i = 1 - x_i \geq z_i$$

for $i = 1, 2, \ldots, k$, and it is:

$$\text{minimize } f(x, z) = \prod_{i=1}^{k} z_i^{-1}$$

(2.24)

subject to:

$$\ln q_i = x_i + z_i \leq 1, \quad i = 1, 2, \ldots, k$$

$$-b_j \cdot \prod_{i=1}^{k} x_i \leq 1, \quad j = 1, 2, \ldots, r$$

$$x_i, z_i \geq 0 \text{ for all } i$$
Mizukami (1968) considers the same problem setting:

\[ z(n) = \log R(n) = \sum_{i=1}^{k} \log R_i(n) = \sum_{i=1}^{k} \log(1 - q_i) \]

Since the functions \( R_i(n) \) are concave the function \( z(n) \) is concave separable. Maximizing \( R(n) \), as the problem 2.20 requires, is equivalent to maximize \( z(n) \). Therefore, the following concave programming formulation is equivalent to the original one:

\[
\text{maximize : } z(n) = \sum_{i=1}^{k} (1 - q_i) \tag{2.25}
\]

subject to:

\[
\sum_{i=1}^{k} c_i n_j \leq b_j, \quad j = 1, 2, ..., r
\]

\[ n_i = 0, 1, 2, ... \text{ for all } i \]

This concave program is equivalent now to the following Linear Program:

\[
\text{maximize : } z = \sum_{i=1}^{k} y_i \tag{2.26}
\]

subject to:

\[ y_i \leq \lambda_i n_i + \mu_i, \quad i = 1, ..., k \]

\[ s = 1, ..., l \]

\[
\sum_{i=1}^{k} c_{ij} n_i \leq b_j, \quad j = 1, 2, ..., r
\]

\[ n_i = 0, 1, 2, ... \text{ for all } i \]

where,

\[ \lambda_i = \{ R(n_{is}) - R_i(n_{it_{s-1}}) \} / (n_{is} - n_{it_{s-1}}) \]

\[ \mu_i = \{ n_{is} R_i(n_{it_{s-1}}) - n_{it_{s-1}} R(n_{is}) \} / (n_{is} - n_{it_{s-1}}) \]

and

\[ n_{i1}, n_{i2}, ..., n_{il} \]

are given values of \( n_i \).

Ghare and Taylor (1969) convert the problem 2.20 into a multidi-
mentional Knapsack problem. The authors prove that the problem 2.20 is equivalent to the program:

\[
\text{maximize: } z = \sum_{i=1}^{k} \sum_{l=1}^{\infty} \alpha_{il}x_{il} \tag{2.27}
\]

subject to: \( \sum_{i=1}^{k} \sum_{l=1}^{\infty} c_{il}x_{il} \leq d_{j}, \quad j = 1, 2, ..., r \)

\( x_{il} = 0 \) or 1

where \( x_{il} = 0 \) implies \( x_{im} = 0 \) for all \( m > 1 \), \( \alpha_{il} = \ln(1 - q_{l}^{1+l}) - \ln(1 - q_{l}) \)

and

\( d_{j} = b_{j} - \sum_{l=1}^{k} c_{lj} \)

A modified branch and bound algorithm developed by Ghare and Walters is used in order to solve the program 2.27.

2.4.3 More complicated Situations

Jensen (1970) considers the problem of optimum redundancy design for a series original system duplicating not each system component but duplicating the series subsystems in which the system can be subdivided. If this is made a "series-parallel-series" redundant reliability network is resulted, an example of which is illustrated in figure 2.4. In this example the original system \((1,2,3,4,5,6)\) is divided into three subsystems \((1,2), 3, (4,5,6)\) and the first and third ones have been duplicated while the second one has been tripled. In general, a redundant design is defined by a set of triplets:

\[ D = \{(i_1,j_1;n_1), (i_2,j_2;n_2), ..., (i_k,j_k;n_k)\} \]

In the example of the figure 2.4 we have:

\[ D = \{(1,2;2), (3,3;3), (4,6;2)\} \]

A dynamic programming formulation is developed in order to solve the problem which is in mathematical terms;
maximize \( R(D) = \prod_{(i,j,n)} R(i,j;n) \)  \hspace{1cm} (2.28)

\[
R = \sum_{j=1}^{r} r_j Q_j 
\]

where \( R \) is the expectation of total reward, \( r_j \) is the reward associated with the goal \( G_j \) and \( Q_j \) is the probability that the goal \( G_j \) will be met, subject to

\[
\sum_{i=1}^{k} c_i n_i \leq w 
\]

where \( c_i \) is the cost of one component of type \( i \).

2.5 Cost Functions minimization under a Reliability Constraint

So far, in the section 2.4 we were occupied with the problem of achieving maximum reliability via parallel redundancy given cost-function constraints associated with the number of redundant components in the
system. This problem means that some upper bounds for the system cost, weight, volume, e.t.c., has been already specified and we are seeking for the redundancy vector $n$ maximizing the over-all reliability. But, in applying this methodology to particular real life problems, the resulted maximum over-all reliability could be regarded very poor for our aims. An obvious solution to this later problem is to increase the bounds in the cost-function constraints and then to solve the maximum reliability problem from the beginning or to find a new solution of the basis of the past one by sensitivity analysis.

An alternative method is to consider the “inverse of the maximum reliability problem”, that is the problem of simultaneously minimizing the cost functions related with the redundant vectors while maintaining the over-all reliability at some desirable level.

In the case of one cost function only Kettelle’s algorithm as well as Barlow and Proschan’s one, both referred in section 2.4.1 as algorithm 2.1 and 2.2 respectively, could be applied for solving the inverse of the maximum reliability problem in this case, which is in mathematical terms:

$$\begin{align*} 
\text{Minimize} & : c(n) = \sum_{i=1}^{k} c_i n_i \\
\text{subject to} & : R(n) = \prod_{i=1}^{k} (1 - q_i)^{n_i+1} \geq L \\
n_i & = 0,1,2,... \text{ for all } i
\end{align*}$$

(2.29)

It is sufficient to seek for that member of the redundancy allocation family, generated by these algorithms, corresponding to a reliability level higher than $L$ but, it cannot be said for the case of multiple cost functions.

Tillman and Liitswager observe a certain objective function of redundancy allocations to be optimized while a set of constraints (linear or not) must be satisfied and the over-all reliability must exceed a desirable level. Speaking in mathematical terms the solved problem is stated as follows:

$$\begin{align*} 
\text{Optimize} & : z(n) = \sum_{i=1}^{k} f_i(n_i) \\
\end{align*}$$

(2.30)
Subject to: \[ \sum_{i=1}^{k} \sum_{j=1}^{n_i+1} g_{ij}(n_i) \leq b_j, \quad j = 1, 2, \ldots, r \]
\[ \prod_{i=1}^{k} (1 - q_i^{n_i}) \geq L \]
\[ n_i = 0, 1, 2, \ldots \text{ for all } i \]

Where \( z(n) \) is the objective function of the system to be optimized, \( f_i(n_i) \) is the objective function at stage \( i \), \( g_{ij}(n_i) \) is the amount of the \( j^{th} \) resource consumed at stage \( i \). An integer programming formulation of the problem 2.30 is made giving the optimum redundancy allocation:

\[ \text{Optimize: } z = \sum_{i=1}^{k} \sum_{m=0}^{n_i'} \Delta f_{im} n_{im} \] (2.31)

Subject to: \[ \sum_{i=1}^{k} \sum_{m=0}^{n_i'} \Delta g_{ijm} n_{im} \leq b_j, \quad j = 1, 2, \ldots, r \]
\[ \sum_{i=1}^{k} \sum_{m=0}^{n_i'} \Delta \ln(1 - q_{im}^{n_{im}+1}) \geq \ln L \]
\[ n_{i0} = 1 \]
\[ n_{im} - n_{i,m-1} \leq 0 \]
\[ n_{im} \geq 0 \text{ for all } i \text{ and } m \]

where \( m \) is an index used to denote a particular redundant component at the stage \( i \), \( n_{im} \) is the \( m^{th} \) redundancy at stage \( i \) with \( n_{im} = 1 \) for \( m \leq n_i \) and \( n_{im} = 0 \) for \( n_i < m \leq n_i' \)(\( n_i' \) is the maximum number of redundant units used at stage \( i \)),

\[ n_i = \sum_{m=1}^{n_i'} n_{im} \]

is the number of redundant components at stage \( i \), \( \Delta f_{im} = f_{im} - f_{i,m-1} \) with \( \Delta f_{i0} = f_{i0} \) and \( \Delta g_{ijm} \) and
\[
\Delta \ln(1 - q_{im}^{n_{im}+1})
\]

are similarly defined. The particular problems solved by this method are 
a) Maximize Reliability subject to multiple linear cost function constraints, b) Minimize cost subject to multiple non-linear and separable restraint functions while maintaining an acceptable reliability level and 
c) Optimal choice of design for parallel redundancy system.

The general situation of minimizing several cost function simultaneously while maintaining the over-all reliability over a desirable level is of our interest in the next Chapter 3. It seems that Multiobjective Programming Techniques could be applied in order to give an answer to this problem.
CHAPTER 3

MULTIOBJECTIVE TECHNIQUES IN PARALLEL REDUNDANCY ALLOCATION

3.1 Multiobjective Programming Methods

Practitioners and Theorists engaged on real world problems for decision making feel that in a large number of situations several objectives should be simultaneously considered.

In such problems what it is desirable is a vector

\[ \mathbf{x} = (x_1, x_2, \ldots, x_k) \]  

(3.1)

where \( x_i \) are non-negative real numbers and the vector \( \mathbf{x} \) must simultaneously optimize (maximize or minimize) \( r \) real-valued functions:

\[ f_j(\mathbf{x}) = f_j(x_1, \ldots, x_k) \]  

(3.2)

Without loss the generality we assume the \( r \) functions \( f_j \) to be minimized.

A set of restraints:

\[ h_m(\mathbf{x}) = h_m(x_1, x_2, \ldots, x_k) \leq b_m, \quad m = 1, 2, \ldots, l \]  

(3.3)

define a set

\[ C^k \subseteq \mathbb{R}^k \]  

(3.4)

of all feasible vectors \( \mathbf{x} \) that are regarded to be candidate solutions.

We note that the requirement of this problem, namely the simultaneous minimizing of all objective functions is hardly met in practice. For example in a transportation problem we would like maximize the transportation velocity and minimize the transportation cost at the same
time. Clearly, it is very difficult to obtain both, since any increasing in the velocity implies higher cost. How can we overcome this difficulty?

Eilon (1972) suggests a variety of methods that could be applied giving efficient solutions that balance the optimum values of the objectives around a tolerable level.

To do this it is necessary to compare the values of the objective functions not separately but simultaneously. In other words, we need compare the various vectors

\[(f_1(x), f_2(x), \ldots, f(x))\]  \hspace{1cm} (3.5)

with each other. This is essentially the basic idea of the following mathematical formulation of the multiobjective problem stated in the beginning of this section \{Bod (1963) and Rudeanu (1969)\}.

Minimize: \[F(x) = (f_1(x), f_2(x), \ldots, f(x))\]  \hspace{1cm} (3.6)
Subject to: \[h_m(x) \leq b_m, \quad m = 1, 2, \ldots, l\]
\[x_1, x_2, \ldots, x_k \geq 0\]

where \(F(x)\) is a vector-valued function taking values in \(\mathbb{R}^r\), that is \(F(x) \leq \mathbb{R}^r\) and its entries \(f_j(x)\) are \(r\) real-valued functions.

Saying “minimum of \(F(x)\)” means the existence of some relation defined on the field for values \(\mathbb{R}^r\) of the function \(F(x)\), denoted by the symbol \(\leq\), on the basis of which two any values \(F\) and \(F'\) in the set \(\mathbb{R}^r\) will be comparable to each other.

One may define in \(\mathbb{R}^r\) any relation in order to compare its elements with each other, but the most common in use ones, related to the real-life situations, are the following:

**RELATION VIA A UTILITY FUNCTION**

It is defined as follows:

\[(f_1, \ldots, f_r) \leq (f'_1, \ldots, f'_r) \quad \text{iff} \quad U(f_1, \ldots, f_r) \leq U(f'_1, \ldots, f'_r)\]  \hspace{1cm} (3.7)

where the function

\[U : \mathbb{R}^r \rightarrow \mathbb{R}\]  \hspace{1cm} (3.8)
is a real-valued "utility" function, non-decreasing with respect to each function \( f_j, j = 1, 2, \ldots, r \). For example, in the case of two objectives only, the function \( U \) could be chosen as the fractional form:

\[
U(f_1, f_2) = \frac{f_1(x)}{f_2(x)}, \quad f_1(x) < 0, \quad f_2(x) > 0
\]

or it could be chosen as the additive utility function:

\[
U(f_1, f_2) = u_1\{f_1(x)\} + u_2\{f_2(x)\}
\]

commonly used in the theory of consumer's choice, where \( u_1, u_2, b_1 \) and \( b_2 \) are positive and \( f_1 \) and \( f_2 > 0 \) for all \( x \).

Thus, considering \( U \) as a function of the vector \( x \) in \( C^k \), we may write:

\[
U(f_1(x), \ldots, f_r(x)) = u(x) \in R
\]  
(3.9)

Therefore, any solution of the single-objective program:

**Minimize:** \( u(x) \)

**Subject to:** \( h_m(x) \leq b_m, \quad m = 1, 2, \ldots, l \)

\( x_1, x_2, \ldots, x_k > 0 \)

is also a solution of the problem 3.6, the set \( R^k \) being equipped with the relation 3.7. To prove it, let \( x^* \) be an optimal solution of the problem 3.10. Therefore,

\[ u(x^*) \leq u(x) \text{ for every } x \in C^k \]

and so,

\[ U(f_1(x^*), \ldots, f_r(x^*)) \leq U(f_1(x), \ldots, f_r(x)), \text{ for every } x \in C^k. \]

Finally, by 3.7

\[ (f_1(x^*), \ldots, f_r(x^*)) \leq (f_1(x), \ldots, f_r(x)), \text{ for every } x \in C^k \]

which means that \( x^* \) is also an optimal solution of the problem 3.6.

The "trade-off" method referred by Eilon (1972) can be considered particular case of the relation 3.7 if we adopt the \( U \) to be the weighted average of the objective functions \( f_j \), that is:

\[
U(f_1(x), \ldots, f_r(x)) = \sum_{j=1}^{r} w_j f_j(x) = u(x)
\]  
(3.11)
where the weights are chosen between 0 and 1 and they sum to unity. The meaning of the weights is that they measure the contribution of each objective function \( f_j \) to the "trade-off" function \( U \) according to its importance. The choice therefore of the numbers \( w_j \) is a particular problem depending on the real-life problem goals.

**Lexicographic Relation**

It is defined as follows:

\[
(f_1, \ldots, f_r) \leq (f'_1, \ldots, f'_r) \quad \text{iff} \\
\quad f_1 < f'_1 \quad \text{or} \\
\quad (f_1 = f'_1 \text{ and } f_2 < f'_2) \quad \text{or} \\
\vdots \quad \vdots \quad \vdots \\
\quad (f_j = f'_j, \quad j = 1, 2, \ldots, r - 1 \text{ and } f_r < f'_r) \quad \text{or} \\
\quad (f_j = f'_j, \quad \text{for all } j)
\]  

(3.12)

One can easily verify that the set \( R^r \) equipped with the relation 3.12 is a totally ordered set and so, if there exists a minimal element \( F^* \) of the set \( F(C^k) \) it is unique.

"Optimization in tandem" referred by Eilon (1972) is embeded into this case. The relevant procedure in order to obtain a vector \( x \) in \( C^k \) minimizing the function \( F(x) \) could be described as follows:

*Adopt some preference ordering between the objectives \( f_j \) according of their importance in the particular problem under consideration. Let a decreasing priority be defined by the sequence \( f_1, f_2, \ldots, f_r \).

*Minimize \( f_1 \) subject to the original restraints. Let the set of its solutions be \( C^{k_1}_1 \); minimize \( f_2 \) over this set. Let \( C^{k_2}_2 \) be the set of its solutions; minimize \( f_3 \) over the new set, e.t.c. In general, minimize the next objective function \( f_j \) over the set \( C^{k_{j-1}}_{j-1} \) of the optimal solutions of the immediately previous function \( f_{j-1} \) minimized over the set \( C^{k_{j-2}}_{j-2} \).

*The procedure will be terminated either when a set of optimal solutions is met containing only one element, or after having minimized the last objective function \( f_r \). In the primer case the solution of the problem 3.6 is unique while in the later the solutions are all the elements contained in the last set.

More precisely, the described procedure leads to a sequence of single-objective function programs as follows:

\[
\text{for } j = 1: \quad \text{Minimize: } f_1(x) \\
\text{Subject to: } h_m(x) \leq b_m, \quad m = 1, 2, \ldots, l \\
x_1, \ldots, x_k \geq 0
\]  

(3.13)
for \( j > 1 \): Minimize \( f_j(x) \)  
Subject to: \( h_m(x) \leq b_m, \ m = 1, 2, \ldots, l \) 
\( f_{\mu}(x) = f_{\mu}^*, \ \mu = 1, 2, \ldots, j - 1 \) 
\( x_1, \ldots, x_k \geq 0 \)  
\( (3.14) \)

where \( f_1^*, f_2^*, \ldots, f_{j - 1}^* \) denote the minimum values of the functions \( f_1(x), f_2(x), \ldots, f_{j - 1}(x) \) obtained by the solution of the previous \( j - 1 \) programs 3.14.

One easily verify that any solution \( x^* \) achieved from the sequence 3.14 is also a solution of the problem 3.6, the set \( R^r \) being provided with the lexicographic relation.

Finally, we note that this method is better than the trade-off one in that no measure of the objective importance is needed; it is sufficient simply to say which objective is the most important between the remainder ones. In the other hand, the disadvantage of a larger volume of calculations should be emphasized.

**Component-to-Component Relation**

It is defined as follows:

\[(f_1, \ldots, f_r) \leq (f'_1, \ldots, f'_r) \iff f_j \leq f'_j, \text{ for all } j\]  
\( (3.15) \)

The set \( R^r \) equipped with the relation 3.15 is a partially ordered set and therefore the set \( F(C^*) \), which is a subset of \( R^r \), adopts several minimal elements (if they exist) being incomparable with each other with respect the component-to-component relation.

It is no difficult one to see that any optimal element in \( F(C^*) \) according to lexicographic relation in \( R^r \) is also an optimal element of the same set if we consider the set \( R^r \) to be provided with the component to-component relation, but the inverse is not valid. Therefore, applying successively the procedure described for the lexicographic relation and cancelling all the vectors in \( F(C^*) \) that are comparable to the optimal vector obtained each time we can find all the incomparable minimal vectors in \( F(C^*) \).

Surely, this last relation and its associated procedure is the most realistic of the three. But the difficulty in the practical applications becomes clear looking at the very large volume of calculations.

Rudeanu (1969) has applied the lexicographic and component-to
component relations in order to obtain the solution of the problem 3.6 in the case of "pseudoboolean" objective functions, that is when the variables $x_i$ are $0 - 1$ valued.

The main purpose of the present work is to point out the possibility of applying the multiobjective methodology in the problem of simultaneously minimizing several cost functions while maintaining the overall reliability over a desirable level and, even though we introduce several objectives, any method generating a family of undominated redundancy allocations may be applied at least in the case of adopting the first of the three relations described before.

In the subsequent sections of this Chapter we are going to prove it and to show the way of applying, in the trade-off treatment, the Kettelle's and Barlow-Proshchan's algorithms described in the previous Chapter 2.

3.2 Formulation of the Parallel Redundancy Allocation Problem

The Parallel Redundancy Allocation Problem which is going to be treated is the following:

"Obtain the optimum number of redundant components in parallel at each stage $i$ of a system consisting of $k$ stages in series so that $r$ objective functions — say system cost, weight, e.t.c. — $f_j(n_1,n_2,...,n_k)$ to be minimized while the over-all reliability $R(n_1,n_2,...,n_k)$ must be maintained over a given level".

According to what has been discussed, the following explicit formulas are available:

$$f_j(n) = \sum_{i=1}^{k} c_{ij}(n_i + 1), \quad j = 1,2,...,r$$

(3.16)

and

$$R(n) = \prod_{i=1}^{k} R_i(n_i) = \prod_{i=1}^{k} \left(1 - q_i \right)^{n_i+1}$$

(3.17)

where $c_{ij} > 0$ is the linear cost coefficient of the $i^{th}$ component in the $j^{th}$ linear cost function $f_j$, $q_i$ is the unreliability of a component of type $i$, $n_i$ is the number of redundant components in parallel at the stage $i$ and $R_i(n_i)$ is the reliability of the stage $i$. 

17
The foregoing sections of this Chapter lead to the following multiobjective programming formulation:

\[
\text{minimize: } F(n) = \left( \sum_{i=1}^{k} c_{i1}(n_1 + 1), \ldots, \sum_{i=1}^{k} c_{in}(n_i + 1) \right) \quad (3.18)
\]

Subject to: \( R(n) = \prod_{i=1}^{k} (1 - q_i) \geq L \)

\( n_i = 0, 1, 2, \ldots \) for all \( i \)

This last program can be simplified reformating the entries \( f_i(n) \) of the vector-valued objective function \( F(n) \) to the form:

\[
f_i(n) = \prod_{i=1}^{k} c_{ij} n_j + \sum_{i=1}^{k} c_{ij}
\]

Since the last term is independent of the \( n_i \) can be ignored. Thus, any optimal solution of the problem 3.18 is also an optimal solution of the program 3.19 described bellow and visa versa.

\[
\text{minimize: } F(n) = \left( \sum_{i=1}^{k} c_{i1} n_1, \ldots, \sum_{i=1}^{k} c_{in} n_i \right) \quad (3.19)
\]

Subject to: \( R(n) = \prod_{i=1}^{k} (1 - q_i) \geq L \)

\( n_i = 0, 1, 2, \ldots \) for all \( i \)

In this program only one constraint is involved, the reliability one.

3.3 Trade-off Application

Choosing some proper weight-vector \( w = (w_1, w_2, \ldots, w_r) \) with entries between 0 and 1 and summing to unity, we may calculate the trade-off function for the program 3.19 as follows:

\[
U(f_1(x), \ldots, f_r(x)) = \sum_{i=1}^{k} \left( \sum_{j=1}^{r} w_j c_{ij} \right) n_i \quad (3.20)
\]
In applying the trade-off technique to real world multiobjective problems we have to find the values of the weights $w_j$. Unfortunately, there is no some general rule to do this since the choice of these values depends on the particular situation. For example, in a transportation problem with two goals, namely the transportation cost and the overall transportation time, the relative importance of the cost and the time will depend on the conditions of the particular realistic situation. Since the choice of $w_j$ is strongly related to the priorities of the objectives this choice depends on the particular conditions of the problem.

B. Roy (1969) in his article “Classemment et choix en presence de points de vue multiples” (that is “Classification and choice with multiple objectives”) gives some examples of the choice of the weights $w_j$. If, for example, all the objectives are characterized with an equivalent priority we may choose $w_j = 1/r$. In the other hand, if the objectives must be strictly ordered we may set $w_j = 2^{r-1}/(2^{r+1} - 1)$. An intermediate situation is the choice of the weights for the examination papers where some papers could be of equal priorities and some other of greater or less importance.

The function 3.20 is non-decreasing with respect to each $f_j$ and being considered as a function of the redundancy vector $n$, ie $u(n)$, is a non-decreasing function of each $n_i$. Therefore, the problem 3.19 is reduced to the following single-objective function program:

$$
\text{Minimize: } u(n) = \sum_{i=1}^{k} \left( \sum_{j=1}^{r} w_j c_{ij} \right) \cdot n_i
$$

$$
\text{Subject to: } R(n) = \prod_{i=1}^{k} \left(1 - q_i \right)^{n_i + 1} \geq L
$$

$$
n_i = 0,1,2,... \text{ for all } i
$$

In this way, the problem of finding the optimum parallel redundancy allocation simultaneously minimizing several cost functions while keeping the over-all reliability over some level has been reduced to a problem of finding the optimum allocation minimizing a single trade-off function under the same reliability restriction.

Any method, included the ones discussed in Chapter 2, solving the problem of achieving the optimum redundancy allocation that minimize
a single cost function while maintaining the over-all reliability at some
level (specified or not) can be applied to the program 3.21. Barlow and
Proschan's procedure for generating an incomplete family of undomina-
ted allocations, as well as Kettelle's algorithm deriving a complete family
of undominated allocations can be applied to the problem 3.21, and both
generate a solution to the original problem 3.19.

A question arises at this point: "Given some undominated redundancy allocation in the problem 3.21 is it also an undominated one in
the problem 3.19? Is the inverse valid?".

In order to have an answer to the first part of the question, let \( \mathbf{n} \)
be some undominated allocation in the problem 3.21. According to the
definition of the undominated allocation (see formula 2.21), for every allo-
cation \( \mathbf{n}' \neq \mathbf{n} \) the following relations are valid.

\[
\begin{align*}
R(\mathbf{n}') &> R(\mathbf{n}) \rightarrow u(\mathbf{n}') > u(\mathbf{n}) \\
R(\mathbf{n}') &> R(\mathbf{n}) \rightarrow u(\mathbf{n}') \geq u(\mathbf{n})
\end{align*}
\]

Since the function \( U \) is non-decreasing with respect each \( f_j \) the statement

\[
\begin{align*}
u(\mathbf{n}') &> u(\mathbf{n}) \rightarrow U(f_1(\mathbf{n}'),...,f_r(\mathbf{n}')) > U(f_1(\mathbf{n}),...,f_r(\mathbf{n}))
\end{align*}
\]

implies that there exists at least one \( j \) for which \( f_j(\mathbf{n}') > f_j(\mathbf{n}) \). In the
other hand the statement:

\[
\begin{align*}
u(\mathbf{n}) = u(\mathbf{n}') \rightarrow U(f_1(\mathbf{n}),...,f_r(\mathbf{n})) = U(f_1(\mathbf{n}'),...,f_r(\mathbf{n}'))
\end{align*}
\]

implies either \( f_j(\mathbf{n}') = f_j(\mathbf{n}) \) for all \( j \), or there exists at least one \( j \) for
which \( f_j(\mathbf{n}') > f_j(\mathbf{n}) \) since the relation \( f_j(\mathbf{n}') < f_j(\mathbf{n}) \) for all \( j \) implies:

\[
\begin{align*}
U(f_1(\mathbf{n}'),...,f_r(\mathbf{n}')) < U(f_1(\mathbf{n}),...,f_r(\mathbf{n}))
\end{align*}
\]

which is not true on the basis of our assumption. Summarizing the results
we have find that:

\[
\begin{align*}
R(\mathbf{n}') &> R(\mathbf{n}) \rightarrow f_j(\mathbf{n}') > f_j(\mathbf{n}) \text{ for at least one } j, \quad (3.22) \\
R(\mathbf{n}') &= R(\mathbf{n}) \rightarrow f_j(\mathbf{n}') = f_j(\mathbf{n}) \text{ for all } \tilde{j} \text{ or } \\
f_j(\mathbf{n}) &> f_j(\mathbf{n}) \text{ for at least one } j
\end{align*}
\]

which means that the allocation \( \mathbf{n} \) is also an undominated allocation in
the problem 3.19.

For every \( \mathbf{n}' \neq \mathbf{n} \) the definition 3.22 is valid but it is easy to see
that:

\[
\begin{align*}
f_j(\mathbf{n}') &> f_j(\mathbf{n}) \text{ for at least one } j \rightleftharpoons \\
U(f_1(\mathbf{n}'),...,f_r(\mathbf{n}')) &> U(f_1(\mathbf{n}),...,f_r(\mathbf{n}))
\end{align*}
\]
and therefore \( R(n') > R(n) \) does not imply \( u(n') > u(n) \) which means that \( n \) is not an undominated allocation for the problem 3.21 even though it is an undominated allocation in the problem 3.19.

Thus, we have proved the following property:

**PROPERTY 3.4**

Every undominated parallel redundancy allocation in the problem 3.21 is also an undominated one in the problem 3.19. The inverse is not valid.

### 3.4 Barlow and Proshan’s Procedure Application

In applying the algorithm 2.1 (Barlow and Proshan’s procedure 1 for a single cost factor) to the problem 3.21 we do not expect a complete family of undominated allocations for the problem 3.19, since this algorithm derives an incomplete family for the problem 3.21.

Thus we create a new undominated redundancy allocation for the problem 3.19 by adding one redundant component at the stage \( i \) for which the increasing of the over-all reliability per unit of average “cost” spent is the maximum. Mathematically speaking, the stage \( i \) is that maximizing the quantity:

\[
Q_i = \left\{ \sum_{j=1}^{r} w_{ij}c_{ij} \right\}^{-1} \cdot \left\{ \log(1 - q_i)^{n_i+2} - \log(1 - q_i)^{n_i+1} \right\} 
\]

(3.23)

It is worth to note that, from a computational point of view, we need the same calculations for solving the problem of obtaining the optimum allocation with minimum several cost functions and reliability over a given level with those in applying the algorithm 2.3 for solving the problem of achieving the optimum allocation maximizing the reliability under several cost constraints.

As a numerical example let the the table 3.1 contains the data of a redundancy allocation problem with two cost functions, say the system cost and the system weight, for a four-stages original system and a reliability level \( L = .90 \). We want to find that parallel redundancy allocation with minimum system cost and system weight keeping the system reliability greater than .90.

Desiring to treat this problem by the trade-off method we have to
choose the weights \( w_1 \) for the system cost and \( w_2 \) for the system weight. Let:

\[
\mathbf{w} = (w_1, w_2) = (.25, .75)
\]

The numbers .25 and .75 are hypothetical being arbitrarily chosen for numerical purposes only. If a real world problem were to be solved we would remind the remark on the weight choice of the section 3.3.

<table>
<thead>
<tr>
<th>i</th>
<th>( c_{i1} )</th>
<th>( c_{i2} )</th>
<th>( q_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.20</td>
<td>5.00</td>
<td>.20</td>
</tr>
<tr>
<td>2</td>
<td>2.30</td>
<td>4.00</td>
<td>.30</td>
</tr>
<tr>
<td>3</td>
<td>3.40</td>
<td>8.00</td>
<td>.25</td>
</tr>
<tr>
<td>4</td>
<td>4.50</td>
<td>7.00</td>
<td>.15</td>
</tr>
</tbody>
</table>

*Table 3.1*

Appendix-I contains a Computer Program creating the family of undominated parallel redundancy allocations for this problem. The input of the program is a table of the form 3.1. The output includesthe family of allocations extending up to the given reliability level (in the present example up to .99955) and the corresponding trade-off, cost and weight function values.

Figure 3.1 represents graphically the results with the over-all reliability against the trade-off function.

3.5 *Kettelle's Algorithm Application*

The Algorithm 2.2 (Kettelle's algorithm) generates a complete family of undominated parallel redundancy allocations for the problem 3.21 but not for the original multiobjective program 3.19, as it has been justified by the property 3.1.

Solving the example of the section 3.4 by this method the calculations are made by two computer programs developed in Appendix-II. The first program generates a complete family of undominated allocations for the subsystem consisting of the stages 1 and 2 and, by repetition, for the subsystem consisting of the stages 3 and 4. The second program derives a complete family of undominated parallel redundancy allocations for the problem 3.21 (being incomplete for the original 3.19)
combining the two subsystems in order to build the entire system consisting of the stages 1, 2, 3, 4.

Figure 3.2 is a diagrammatical representation of the final family with the trade-off function against the over-all reliability.

![Figure 3.1]

### 3.6 Discussion of the Results and Conclusions

Before proceeding to a comparison of the results we note that in applying Algorithm 2.1 to the example of the table 3.1 the cost and weight values as well as the trade-off function values have been calculated in
Figure 3.2
order to include their constant initial values, while the corresponding values in applying Algorithm 2.2 to the same problem do not. Therefore, in order to compare the results obtained by these algorithms we have to subtract the initial values from the cost functions and trade-off ones achieved by Algorithm 2.1.

The figures 3.1 and 3.2 may be used in order to compare the families of undominated parallel redundancy allocations derived by both procedures. Inspecting these figures we see that Algorithm 2.2 (kettelle) generates a wider family than Algorithm 2.1 (Barlow and Proschan) containing a larger number of allocations.

Taking the reliability lower level to be .90, the Algorithm 2.1 provides the optimal redundancy allocation (2,2,2,1) with over-all reliability equal to .92876, trade-off value 34.67, redundancy cost 18.30 and redundancy weight 41.0. Algorithm 2.2 derives the optimal solution (2,3,11), over-all reliability .901711, trade-off value 32.050, redundancy cost 17.20 and redundancy weight 37.00. Thus the later algorithm generates a cheaper trade-off solution.

Through the pages of this work, the problem of obtaining the optimal parallel redundancy allocation with minimum several cost functions under a reliability constraint has been treated using multiobjective techniques.

Considering the set of vectors with entries the values of the cost functions provided with the relation via some utility function, we have applied the trade-off technique replacing the objectives by their weighted average. Surely, we may use some other utility function of non-linear form. In this case Algorithms 2.1 and 2.2 cannot be applied and so we have to seeking for some other method depending on the utility function form.

In the other hand, in the “optimization in tandem”, that is if we adopt the lexicographic relation, a sequence of single-objective function programs should be solved with the additional restraint to maintain the optimal solutions of the predecessor program. Tillman and Liittschwager (1967) method seems to be convenient for solving these programs.

Finally, in the most realistic situation of adopting the component to component relation, the same methodology may be followed but the difficulty of a large volume of calculations as well as some technical problems in cancelling the path leading to a particular minimal vector into the set of the vectors with entries the values of the objective functions arise.
C THIS PROGRAM GENERATES AN UNCOMPLETE FAMILY OF UNDOMINATED ALLOCATIONS FOR THE PROBLEM OF ACHIEVING A GIVEN RELIABILITY LEVEL WITH MINIMUM SEVERAL LINEAR COST FACTORS USING BARLOW-PROSCHAN METHOD

PROGRAMERI Y. K. KONTISSIS
DATE: 05/08/1975
COMPUTER: ICL
INSTALLATION: UNIVERSITY OF SOUTHAMPTON COMPUTER CENTER

MASTER KONT
DIMENSION C(15,5),W(5),Q(15),N(15),COST(5),REL(15),S(15),QUANT(15)
INTEGER R

READ(5,104) K,R,BR
100 FORMAT(215,F7,5)
READ(5,103) (Q(I),I=1,K)
105 FORMAT(4F7,5)
READ(5,110) (W(J),J=1,R)
110 FORMAT(2F7,5)
READ(5,115) (C(I,J),I=1,K,J=1,R)
115 FORMAT(4F9,2)
WRITE(6,116)
116 FORMAT(1H1)

DO 5 I=1,K
5 N(I)=0
10 DO 15 J=1,R
COST(J) = 0,
DO 15 I = 1, K
15 COST(J) = COST(J) + C(I, J) * (N(I) + 1)
DO 20 I = 1, K
20 REL(I) = REL(I) * (N(I) + 1)
SYSREL = 1,
DO 25 I = 1, K
25 SYSREL = SYSREL * REL(I)
TOTCOS = 0,
DO 30 J = 1, R
30 TOTCOS = TOTCOS + W(J) * COST(J)
WRITE(6, 120) (N(I), I = 1, K)
120 FORMAT('ALLOCATION: ', A(128), ',')
WRITE(6, 130) SYSREL, TOTCOS, (J, COST(J), J = 1, R)
130 FORMAT('RELIABILITY: ', F7.5/14, ', ' TRADE-OFF FUNCTION: ', F8.2
+ /7H ', 'COST-FACTORS: ', F2(', C(', 128, ') = ', F8.2))
IF(SYSREL .GE. SR) GO TO 50
DO 40 I = 1, K
40 S(I) = 0,
DO 35 J = 1, R
35 S(I) = S(I) + W(J) * C(I, J)
QUANT(I) = (1 / S(I)) * (LOG(1 - Q(I)) * (N(I) + 2)) + LOG(1 - Q(I)) * (N(I) + 1))
XMAX = QUANT(I)
IMAX = 1
DO 45 I = 1, K
45 IF(QUANT(I), LE. XMAX) GO TO 45
XMAX = QUANT(I)
IMAX = I
CONTINUE
N(IMAX) = N(IMAX) + 1
GO TO 10
STOP
END
ALLOCATIONS: 0, 0, 0, 0
RELIABILITY: 0.35700
TRADE-OFF FUNCTION: 20.85
COST-FACTORS: c(1) = 11.40 c(2) = 24.00

ALLOCATIONS: 0, 1, 0, 0
RELIABILITY: 0.45410
TRADE-OFF FUNCTION: 24.43
COST-FACTORS: c(1) = 13.70 c(2) = 28.00

ALLOCATIONS: 1, 1, 0, 0
RELIABILITY: 0.55692
TRADE-OFF FUNCTION: 28.40
COST-FACTORS: c(1) = 14.90 c(2) = 33.00

ALLOCATIONS: 1, 1, 1, 0
RELIABILITY: 0.69515
TRADE-OFF FUNCTION: 35.32
COST-FACTORS: c(1) = 18.50 c(2) = 41.00

ALLOCATIONS: 1, 1, 1, 1
RELIABILITY: 0.80057
TRADE-OFF FUNCTION: 41.70
COST-FACTORS: c(1) = 22.80 c(2) = 48.00

ALLOCATIONS: 1, 2, 1, 1
RELIABILITY: 0.85690
TRADE-OFF FUNCTION: 45.27
COST-FACTORS: c(1) = 25.70 c(2) = 52.00

ALLOCATIONS: 2, 2, 1, 1
RELIABILITY: 0.88643
TRADE-OFF FUNCTION: 49.33
COST-FACTORS: c(1) = 26.30 c(2) = 57.00

ALLOCATIONS: 2, 2, 2, 1
RELIABILITY: 0.92876
TRADE-OFF FUNCTION: 56.16
COST-FACTORS: c(1) = 29.70 c(2) = 65.00

ALLOCATIONS: 2, 3, 2, 1
RELIABILITY: 0.94880
TRADE-OFF FUNCTION: 59.75
COST-FACTORS: c(1) = 32.00 c(2) = 69.00
ALLOCATION: 2, 3, 2, 2,
RELIABILITY: 0.95532
TRADE-OFF FUNCTION: 66.12
COST-FACTORS: C(1) = 36.50 C(2) = 76.00

ALLOCATION: 2, 3, 3, 2,
RELIABILITY: 0.97681
TRADE-OFF FUNCTION: 72.97
COST-FACTORS: C(1) = 39.90 C(2) = 84.00

ALLOCATION: 2, 4, 3, 2,
RELIABILITY: 0.98240
TRADE-OFF FUNCTION: 76.55
COST-FACTORS: C(1) = 42.20 C(2) = 88.00

ALLOCATION: 3, 4, 3, 2,
RELIABILITY: 0.98874
TRADE-OFF FUNCTION: 80.60
COST-FACTORS: C(1) = 43.40 C(2) = 93.00

ALLOCATION: 3, 5, 3, 2,
RELIABILITY: 0.99042
TRADE-OFF FUNCTION: 84.17
COST-FACTORS: C(1) = 45.70 C(2) = 97.00

ALLOCATION: 3, 5, 3, 3,
RELIABILITY: 0.99327
TRADE-OFF FUNCTION: 90.53
COST-FACTORS: C(1) = 50.20 C(2) = 104.00

ALLOCATION: 3, 5, 4, 3,
RELIABILITY: 0.99819
TRADE-OFF FUNCTION: 97.40
COST-FACTORS: C(1) = 53.60 C(2) = 112.00

ALLOCATION: 4, 5, 4, 3,
RELIABILITY: 0.99747
TRADE-OFF FUNCTION: 101.45
COST-FACTORS: C(1) = 54.80 C(2) = 117.00

ALLOCATION: 4, 5, 4, 3,
RELIABILITY: 0.99778
TRADE-OFF FUNCTION: 105.03
COST-FACTORS: C(1) = 57.10 C(2) = 121.00

ALLOCATION: 6, 6, 5, 3,
RELIABILITY: 0.99871
TRADE-OFF FUNCTION: 111.87
COST-FACTORS: C(1) = 50.50 C(2) = 129.00
allocation: 4, 5, 5, 4,
reliability: 0.99914
trade-off function: 118.25
cost-factors: c(1) = 65.00 c(2) = 156.00

allocation: 5, 6, 5, 4,
reliability: 0.99940
trade-off function: 122.30
cost-factors: c(1) = 66.20 c(2) = 141.00

allocation: 5, 7, 5, 4,
reliability: 0.99995
trade-off function: 125.87
cost-factors: c(1) = 68.50 c(2) = 145.00
THIS PROGRAM GENERATES TWO FAMILIES OF UNDOMINATED ALLOCATIONS;
ONE FOR EACH TWO-STAGE SUBSYSTEM 1-2 AND 3-4, USING KETELLE &
ALGORITHM BEING APPLIED TO A MULTIOBJECTIVE REDUNDANCY ALLOCATION;
PROBLEM WITH A FOUR-STAGE SYSTEM:

PROGRAMERI K4, KONTESSIS

DATE 18/8/75

COMPUTER: ICL

INSTALLATION: SOUTHAMPTON UNIVERSITY COMPUTING CENTER

MASTER KONT1

DIMENSION Q(I,4), W(I,2), C(I,2), CNEW(I,4), C1(8,8), C2(8,8), F(8,8), RS(I,8);,
+ N41(8,8)

INTEGER R

READ(5,10) K, R

100 FORMAT(2I5)

READ(5,10) (Q(I), I=1,K)

105 FORMAT(4F5.6)

READ(5,11) (W(J), J=1,K)

110 FORMAT(2F5.6)

READ(5,11) (C(I,J), I=1,K, J=1,R)

115 FORMAT(4F5.3)

INDEX=0

I1=1

I2=2

DO 5 I=1,K

CNEW(I)=0;

5 DO 5 J=1,R

CNEW(I)=CNEW(I)+C(I,J)*W(J)
10 WRITE(6,140) I1, I2
120 FORMAT(/,'REDUNDANCY ALLOCATIONS FOR SUBSYSTEM: 1, 11, 1 = 1, 11')
    DO 15 N1=1, 8
    DO 15 N2=1, 8
0041 C1(N1,N2) = C1(N1,1) + C1(1,N2) + C2(1,1) + C2(1,N2)
0042 C2(N1,N2) = C1(N1,1) + C1(1,N2) + C2(1,1) + C2(1,N2)
0043 C1(N1,N2) = CNEW(1) + CNEW(4) + CNEW(2)
0044 R8(N1,N2) = (1-Q(1)*R1) + (1-Q(2)*R2)
0045 NN1(N1,N2) = N1 = 1
0046 DO 20 N1 = 1, 8
0047 WRITE(6,145) (NN1(N1,N2),N2=1,8)
145 FORMAT(1H*,B(I1,8X))
0049 WRITE(6,150) (NN1(N2,N1),N2=1,8)
150 FORMAT(1H*,B(cX,I1,8X))
0051 WRITE(6,155) (C1(N1,N2),N2=1,8)
0052 WRITE(6,155) (C2(N1,N2),N2=1,8)
0053 WRITE(6,155) (F(N1,N2),N2=1,8)
155 FORMAT(1H*,B(FB,3,1X))
0059 WRITE(6,140) (R8(N1,N2),N2=1,8)
140 FORMAT(1H*,B(FB,6,1X))
0057 CONTINUE
20 CONTINUE
    IF(INDEX.EQ.1) GO TO 30
0059 INDEX = 1
0060 Q(1) = Q(3)
0061 Q(2) = Q(4)
0062 C1(1,1) = C1(3,1)
0063 C2(1,1) = C2(3,1)
0064 C1(1,2) = C1(3,2)
0065 C2(1,2) = C2(3,2)
0066 CNEW(1) = CNEW(3)
0067 CNEW(2) = CNEW(4)
0068 I1 = 3
0069 I2 = 4
0070 GO TO 10
30 WRITE(6,145)
0072 145 FORMAT(1H1)
0073 STOP
0074 END
## Redundancy Allocations for Subsystem 1 to 7

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.000</td>
<td>2.300</td>
<td>4.600</td>
<td>6.900</td>
<td>9.200</td>
<td>11.500</td>
<td>13.800</td>
<td>16.100</td>
</tr>
<tr>
<td>0</td>
<td>0.000</td>
<td>4.000</td>
<td>8.000</td>
<td>12.000</td>
<td>16.000</td>
<td>20.000</td>
<td>24.000</td>
<td>28.000</td>
</tr>
<tr>
<td>0</td>
<td>0.000</td>
<td>3.575</td>
<td>7.150</td>
<td>10.725</td>
<td>14.300</td>
<td>17.875</td>
<td>21.450</td>
<td>25.025</td>
</tr>
<tr>
<td>0.55400</td>
<td>0.72600</td>
<td>0.77840</td>
<td>0.795920</td>
<td>0.798360</td>
<td>0.799417</td>
<td>0.799895</td>
<td>0.799948</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.672000</td>
<td>0.976000</td>
<td>0.934000</td>
<td>0.956240</td>
<td>0.957667</td>
<td>0.959300</td>
<td>0.959790</td>
<td>0.959937</td>
</tr>
<tr>
<td>2</td>
<td>0.672000</td>
<td>0.976000</td>
<td>0.934000</td>
<td>0.956240</td>
<td>0.957667</td>
<td>0.959300</td>
<td>0.959790</td>
<td>0.959937</td>
</tr>
<tr>
<td>3</td>
<td>0.672000</td>
<td>0.976000</td>
<td>0.934000</td>
<td>0.956240</td>
<td>0.957667</td>
<td>0.959300</td>
<td>0.959790</td>
<td>0.959937</td>
</tr>
<tr>
<td>4</td>
<td>0.672000</td>
<td>0.976000</td>
<td>0.934000</td>
<td>0.956240</td>
<td>0.957667</td>
<td>0.959300</td>
<td>0.959790</td>
<td>0.959937</td>
</tr>
<tr>
<td>5</td>
<td>0.672000</td>
<td>0.976000</td>
<td>0.934000</td>
<td>0.956240</td>
<td>0.957667</td>
<td>0.959300</td>
<td>0.959790</td>
<td>0.959937</td>
</tr>
<tr>
<td>6</td>
<td>0.672000</td>
<td>0.976000</td>
<td>0.934000</td>
<td>0.956240</td>
<td>0.957667</td>
<td>0.959300</td>
<td>0.959790</td>
<td>0.959937</td>
</tr>
<tr>
<td>7</td>
<td>0.672000</td>
<td>0.976000</td>
<td>0.934000</td>
<td>0.956240</td>
<td>0.957667</td>
<td>0.959300</td>
<td>0.959790</td>
<td>0.959937</td>
</tr>
</tbody>
</table>

<p>| 8 | 8.400 | 10.700 | 13.000 | 15.300 | 17.600 | 17.600 | 17.600 | 17.600 |
| 9 | 35.000 | 39.000 | 43.000 | 47.000 | 51.000 | 55.000 | 59.000 | 63.000 |
| 11 | 0.672000 | 0.976000 | 0.934000 | 0.956240 | 0.957667 | 0.959300 | 0.959790 | 0.959937 |</p>
<table>
<thead>
<tr>
<th>REDUNDANCY ALLOCATIONS FOR SUBSYSTEM 3-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000 1 4.500 9.000 13,500 18,000 22,500 27,000 31,500</td>
</tr>
<tr>
<td>0.000 1 7.000 14.000 21.000 28.000 35.000 42.000 49.000</td>
</tr>
<tr>
<td>0.000 1 5.375 12.750 19.125 25.500 31.875 38.250 44.625</td>
</tr>
<tr>
<td>0.637500 0.753125 0.747265 0.747260 0.749943 0.749991 0.749999 0.750000</td>
</tr>
<tr>
<td>7 0.000 1 4.900 12.400 19.600 26.800 34.000 41.200 48.400</td>
</tr>
<tr>
<td>7 0.000 1 7.300 14.600 21.900 29.200 36.500 43.800 51.100</td>
</tr>
<tr>
<td>8 0.000 1 15.000 22.000 29.000 36.000 43.000 50.000 57.000</td>
</tr>
<tr>
<td>6.250 1 13.250 19.500 25.750 32.000 38.250 45.000 51.750</td>
</tr>
<tr>
<td>0.749975 0.749980 0.749984 0.749989 0.749999 0.750000</td>
</tr>
<tr>
<td>2 0.000 1 12.000 18.000 24.000 30.000 36.000 42.000 48.000</td>
</tr>
<tr>
<td>5.625 1 11.375 17.750 24.125 30.500 36.875 43.250 49.625</td>
</tr>
<tr>
<td>13.750 1 20.000 26.250 32.500 38.750 45.000 51.250 57.500</td>
</tr>
<tr>
<td>0.836919 0.837355 0.837792 0.838229 0.838666 0.839103 0.839539</td>
</tr>
<tr>
<td>3 0.000 1 14.700 21.400 28.100 34.800 41.500 48.200 54.900</td>
</tr>
<tr>
<td>24.000 1 31.000 38.000 45.000 52.000 59.000 66.000 73.000</td>
</tr>
<tr>
<td>20.3125 0.975362 0.992732 0.999959 0.999999 0.999999 0.999999</td>
</tr>
<tr>
<td>4 0.000 1 18.100 24.600 31.100 37.600 44.100 50.600 57.100</td>
</tr>
<tr>
<td>32.000 1 39.000 45.900 52.800 59.700 66.600 73.500 80.400</td>
</tr>
<tr>
<td>27.400 0.995652 0.999981 0.999998 0.999998 0.999998 0.999998</td>
</tr>
<tr>
<td>5 0.000 1 21.500 28.000 34.500 41.000 47.500 54.000 60.500</td>
</tr>
<tr>
<td>40.000 1 47.000 53.500 60.000 66.500 73.000 79.500 86.000</td>
</tr>
<tr>
<td>34.375 0.999924 0.999964 0.999994 0.999994 0.999994 0.999994</td>
</tr>
<tr>
<td>6 0.000 1 24.700 31.200 37.700 44.200 50.700 57.200 63.700</td>
</tr>
<tr>
<td>48.000 1 55.000 61.500 68.000 74.500 81.000 87.500 94.000</td>
</tr>
<tr>
<td>41.100 0.999974 0.999974 0.999974 0.999974 0.999974 0.999974</td>
</tr>
<tr>
<td>0.849948 0.849990 0.849990 0.849990 0.849990 0.849990</td>
</tr>
<tr>
<td>7 0.000 1 28.300 34.800 41.300 47.800 54.300 60.800 67.300</td>
</tr>
<tr>
<td>56.000 1 63.000 69.500 76.000 82.500 89.000 95.500 102.000</td>
</tr>
<tr>
<td>47.9992 0.999933 0.999983 0.999983 0.999983 0.999983 0.999983</td>
</tr>
<tr>
<td>0.849997 0.849997 0.849997 0.849997 0.849997 0.849997</td>
</tr>
</tbody>
</table>
This program combines the two two-stage subsystems and provides the final family of undominated allocations produced by Kettelle's algorithm being applied to a multiobjective redundancy allocation problem with a four-stage system.

Program: RRED
Date: 15/9/75
Computer: ICL
Installation: Southampton University Computing Center

MASTER KONTZ
DIMENSION M(12), N(12), N3(12), N4(12), C12(12), C122(12), C341(12),
+ C342(12), F12(12), F34(12), R12(12), R34(12), C12341(12),
+ C12342(12, 14), F1234(12, 14), R1234(12, 14),
+ N1(12, 14), N2(12, 14), N3(12, 14), N4(12, 14)
READ(5, 105) (M(I), N(I), N3(I), C12(I), C122(I), C341(I), C342(I), F12(I),
+ F34(I), R12(I), R34(I), C12341(I), C12342(I), F1234(I), R1234(I))
100 FORMAT(12(F11.1, 14(F9.0, 0)))
WRITE(0, 103)
105 FORMAT(/***/1H, 'REduNDaNCY AlLOCAtIOnS fOr tHe systEm: 1-2-3-41')
DO 5 L1=1, 12
DO 5 L2=1, 12
C12341(L1, L2)=C121(L1)+C341(L2)
C12342(L1, L2)=C122(L1)+C342(L2)
F1234(L1, L2)=F12(L1)+F34(L2)
R1234(L1, L2)=R12(L1)+R34(L2)
5 NN1(L1, L2)=NN1(L1)
NN2(L1, L2)=NN2(L1)
NN3(L1, L2)=NN3(L1)
NN4(L1, L2)=NN4(L1)
DO 10 L1=1, 12
WRITE(0, 110) (NN1(L1, L2), L2=1, 12)
110 FORMAT(1H12(1, 6X))
WRITE(0, 115) (NN2(L1, L2), L2=1, 12)
115 FORMAT(1H12(2X, 11, 6X))
WRITE(0, 120) (NN3(L1, L2), L2=1, 12)
120 FORMAT(1H12(4X, 11, 6X))
WRITE(0, 125) (NN4(L1, L2), L2=1, 12)
125 FORMAT(1H12(6X, 11, 2X))
WRITE(0, 130) (C12341(L1, L2), L2=1, 12)
WRITE(0, 130) (C12342(L1, L2), L2=1, 12)
WRITE(0, 130) (F1234(L1, L2), L2=1, 12)
130 FORMAT(1H12(8X, 3, 1X))
WRITE(0, 135) (R1234(L1, L2), L2=1, 12)
135 FORMAT(1H12(8X, 6, 1X))
10 CONTINUE
STOP
END
SUMMARY

The present work deals with the problem of allocating to the stages of a system spare components in parallel in order to improve the reliability of this system under several cost-functions.

This problem can be regarded directly, that is to maximize the over-all reliability under several cost-function constraints (say system cost, system weight, system volume e.t.c.). Several papers have been written on this problem using various mathematical programming methods. But also, of interest could be the inverse problem which is to maintain the over-all system reliability over some safety level and it must be done with the cheapest several cost-functions. This topic has been treated by Kettelle (1962) and Barlow and Proschan (1965) in the case of a single cost-function but multiobjective programming techniques are involved in order to describe and solve the same problem in the case of multiple cost-functions to be simultaneously minimized.

What the work on hand does is that it considers a systematical multiobjective formulation of this inverse redundancy allocation problem based on various relations introduced in the Cartesian space of the cost-functions, and then solves this problem using the "trade-off" technique.
REFERENCE

30. Proschan and Bray, Optimum Redundancy under Multiple Constraints, O.R. (USA), 1965.
CONTENTS

CHAPTER 1. RELIABILITY OF A SYSTEM 227

1.1 Introduction and System Reliability Definition 227
1.2 Failure Distributions Related to Reliability 228
1.3 Some of the most common in use Failure Distributions 229
1.4 System Reliability Calculation 232

1.4.1 Series and Parallel Systems 232
1.4.2 Parallel-Series Systems 234
1.4.3 More Complicated Systems 236

CHAPTER 2. RELIABILITY OPTIMIZATION 239

2.1 Introduction to Redundancy 239
2.2 Parallel Redundancy Effects on the Over-all Reliability 240
2.3 The Problem of Parallel Redundancy Optimum Allocation 243
2.4 Reliability Maximization under Constraints 245

2.4.1 Series Systems; Single Constraint 245
2.4.2 Series Systems; Multiple Constraints 249
2.4.3 More complicated Situations 253

2.5 Cost Functions minimization under a Reliability Constraint 254

CHAPTER 3. MULTIOBJECTIVE TECHNIQUES IN PARALLEL REDUNDANCY ALLOCATION 258

3.1 Multiobjective Programming Methods 258
3.2 Formulation of the Parallel Redundancy Allocation Problem 263
3.3 Trade-off Application 264
3.4 Barlow and Proschan’s Procedure Application 267
3.5 Kettelle's Algorithm Application
3.6 Discussion of the Results and Conclusions

APPENDIX I
APPENDIX II
SUMMARY
REFERENCE