# A Branch–and–Bound Approach to the Optimization of Redundant Software under Failure Correlation

Walter J. Gutjahr and Gabriele Uchida
Department of Statistics and Decision Support Systems
University of Vienna
Universitaetsstrasse 5/9, A-1010 Wien, Austria

May 19, 2000

**Abstract.**

A technique for computing optimal redundancy levels for the components of a fault–tolerant system containing recovery blocks with dependently failing module versions is presented. Optimization is done under a given budget constraint and with the overall reliability as the objective. The failure probabilities of the test segments are assumed to be negligible, and within each recovery block, price and estimated reliability of each redundant version as well as the failure correlation between the versions is assumed to be constant. For solving the optimization problem, a Branch–and–Bound technique, using the solutions of continuous problem relaxations for the determination of bounds, is developed. Computational experiments with randomly generated test instances show that the suggested algorithm still performs well in situations where a computation based on dynamic programming is not feasible anymore.

**Scope and purpose.**

Using redundant program versions and increasing thereby the system reliability is a well–established way to build fault–tolerant software systems, as they are required in safety–critical computing (flight control, power plants, medical technique, defense systems). Given a certain budget for software development or purchase of software components, it makes sense to ask how this budget can be spent in order to maximize the achieved reliability. A problem occurring in this context is that different versions of a program, even if they have been developed independently from each other, do not *fail* independently. Our approach takes this correlation between failures into account and allows an efficient computation of optimal redundancy levels for problem instance sizes of practical interest.

*Keywords:* Branch–and–Bound; Correlated failures; Fault–tolerant software; Optimization; Recovery blocks; Software reliability

# 1 Introduction

Modern software–intensive systems form essential technological building blocks in several areas where high dependability is required, such as flight control, nuclear power plants, medical technique, or defense systems. The insight is growing that investing software development costs into such applications of high criticality in an effective way demands methods of decision–making under uncertainty, including techniques from statistics and operations research (cf. Poore and Trammell [10]).

One specific aspect in this context is the attempt to increase the reliability of systems by using *software redundancy*. Optimization techniques for the reliability of basic redundant system structures under given budget constraints have been developed by Belli and Jedrzejowicz [1] and by Berman and Ashrafi [2]. Later, these approaches have also been extended to some more complex redundant structures (see, e.g., Berman and Kumar [3]). All these articles are based on the assumption of *independently* failing module versions, an assumption which is usually considered harmfully because it is not in accordance with empirical results (see Knight and Leveson [7]). The mentioned authors are well aware of this drawback (see the concluding sections in the three publications cited above) and suggest to extend their approaches to models where there is a (positive) failure correlation between different redundant software versions.

One first attempt to follow such suggestions has been presented in [5]. It has been shown there that two basic reliability optimization models described in Berman's and Ashrafi's inspiring paper [2], namely Model 2 and Model 4, can be extended to the case of positive failure correlation. (The two other models described in [2], Model 1 and Model 3, do not use redundancy.) As the statistical framework for representing failure correlation, the concept developed by Eckhardt and Lee [4] and specified by Nicola and Goyal [8] has been used. For the optimization itself, two approaches have been proposed: dynamic programming, and a partial enumeration technique using bounds for the decision variables.

Although the partial enumeration technique suggested in [5] works well for small examples, it has turned out in our experiments that it fails to work in larger applications: the described bounds are too weak for a drastic reduction of the solution space, as it would be required for performing the final enumeration step within reasonable time. There remained the hope that dynamic programming would be able to cope with medium–size problem instances. As it will be shown in this article, however, this expectation was only partially satisfied by the experimental results.

Therefore, as an alternative, we have adapted the well–known *Branch–and–Bound* approach to the given problem context. Branch–and–Bound has also been used in [3] for related purposes, but with a different strategy of bound computation. We found that in our context, very efficient bounds for the objective function (instead, as in [5], for the decision variables) can be derived by a *continuous relaxation* of the optimization problem and an exploitation of the specific structure of the objective function: Showing that it is log–convex, we can apply an iterative search procedure for computing lower bounds of high quality within relatively few iterations. By means of this technique, problems of a realistic order of magnitude can be solved within moderate time limits, as demonstrated by our experiments.

The paper is organized as follows: In Section 2, we recapitulate the basic optimization model, as formulated in [5]. In Section 3, the dynamic programming approach is shortly outlined, and our new Branch–and–Bound–based technique is presented. Section 4 contains the experimental results, and the concluding Section 5 discusses restrictions and issues for future research.

## 2 Recapitulation of the Model

The question we want to deal with is how to distribute software redundancy over a given software system consisting of several modules, if a certain additional budget for developing or purchasing redundant module versions is available, and the overall reliability of the system is to be maximized. As input data, we need information on estimated reliabilities of module versions, as well as on their costs.

A sound formal model for computing the reliability of the system from that of the used versions is the necessary base for the optimization. In the present paper, we restrict ourselves to a dependent–failure modification of Model 2 in [2]. (Model 4 in [2] is a generalization of Model 2 taking the possibility of a re–use of versions in different programs into account; here, we do not consider this aspect.) The partial extension of Berman's and Ashrafi's Model 2 to dependently failing module versions has been called Model 2' in [5]. Let us mention that in another sense, Model 2' is more restricted than Model 2, since it makes the additional assumption that different versions of a module are available at the same cost and have the same estimated reliability, an assumption which is a consequence of the limits of the Eckhardt–Lee resp. Nicola–Goyal approach for modeling dependent failures. Recent results [6] indicate that this restriction may possibly be overcome. Nevertheless, since the focus of this paper is rather on optimization techniques than on statistical modeling, we shall stay here within the framework of Model 2'.

The assumptions of this model are the following:

1. The software system consists of $n$ modules (labeled by $1, \ldots, n$).

2. For module $i$, it is possible to use $x_i \geq 1$ redundant versions, each at a cost of $c_i$ currency units. The estimated unreliability (probability of failing) of each version of this module is $\pi_i \in ]0, 1[$. Combining these $x_i$ versions leads to a module reliability $R_i$. Failures of the $x_i$ versions (at a random input according to the usage distribution) are positively correlated; the number of failing versions satisfies a *Beta–binomial* distribution (see [8]). The probability that the entire module fails is equal to the probability that all $x_i$ redundant versions contained in it fail. (This assumption reflects the situation of a *recovery block* with fault free test segment.) Under the Beta–binomial distribution, the last–mentioned probability that module $i$ fails is given by

$$\varphi_i(x_i) = \frac{B(\alpha_i + x_i, \beta_i)}{B(\alpha_i, \beta_i)} = \frac{(\alpha_i + x_i - 1)\,(\alpha_i + x_i - 2)\,\ldots\,\alpha_i}{(\alpha_i + \beta_i + x_i - 1)\,(\alpha_i + \beta_i + x_i - 2)\,\ldots\,(\alpha_i + \beta_i)}, \qquad (1)$$

where

$$\alpha_i = \frac{\pi_i}{\theta_i}, \quad \beta_i = \frac{1 - \pi_i}{\theta_i}, \qquad (2)$$

and the parameter $\theta_i$ is a *correlation level*, the limiting cases $\theta_i = 0$ and $\theta_i = \infty$ being the case of independence and of identical failure behavior, respectively. The reliability

$R_i$ of module $i$ is then given by

$$R_i = 1 - \varphi_i(x_i). \tag{3}$$

3. Contrary to versions of one and the same module, different modules fail independently from each other — a reasonably assumptions since different modules are implemented to perform different functions. As a consequence, the system reliability is

$$R = \prod_{i=1}^{n} R_i. \tag{4}$$

4. The invested overall cost for developing or purchasing module versions is limited by a budget $C$.

In total, these assumptions lead to the following optimization problem:

$$\text{Maximize } R = \prod_{i=1}^{n}(1 - \varphi_i(x_i)) \tag{5}$$

subject to

$$\sum_{i=1}^{n} c_i x_i \le C, \quad x_i \in \mathbb{N} = \{1, 2, \ldots\} \quad (i = 1, \ldots, n), \tag{6}$$

where $\varphi_i(x_i)$ is given by (1).

# 3 Solving the Optimization Problem

## 3.1 Dynamic Programming

The application of Dynamic Programming to reliability optimization of redundant systems has already been described in [2] and, modified for the case of correlated failures, in [5], so we may keep the presentation very concise. By $R_i(s)$, we denote the optimal reliability of the subsystem consisting of the modules $i, i+1, \ldots, n$, if there is a remaining budget of $s$ currency units for these $n - i + 1$ modules. The optimal reliability of the entire system is then given by $R_1(C)$. From the equations (4), (3) and (1), we obtain the recurrence

$$R_i(s) = \max\left(\left[1 - \frac{B(\alpha_i + x_i, \beta_i)}{B(\alpha_i, \beta_i)}\right] \cdot R_{i+1}(s - c_i x_i)\right) \quad (i = 1, \ldots, n),$$

where the maximum is to be taken over all $x_i \in \mathbb{N}$ with

$$c_i x_i \le s - \sum_{k=i+1}^{n} c_k,$$

because the budget $s$ is still available, and for each of the modules $i + 1, \ldots, n$, at least one version will be required. Therefore,

$$1 \le x_i \le \left\lfloor \left(s - \sum_{k=i+1}^{n} c_k\right) / c_i \right\rfloor.$$

The case $i = n$ yields the base of the recursion:

$$R_n(s) = \max \left[ 1 - \frac{B(\alpha_n + x_n, \beta_n)}{B(\alpha_n, \beta_n)} \right],$$

where the maximum is to be taken over all $x_n \in \mathbb{N}$ with $x_n \leq \lfloor s/c_n \rfloor$.

By computing the values $R_i(s)$ for each $s \in \mathbb{N}$ with

$$\sum_{k=i}^{n} c_k \leq s \leq C - \sum_{k=1}^{i-1} c_k$$

and for $i = n, n-1, \ldots, 1$, the optimal numbers $x_i$ can be determined successively.

## 3.2 The New Branch–and–Bound Approach

For specifying a Branch–and–Bound algorithm (see, e.g., [9]), two requisites are necessary:

1. a strategy for partitioning the feasible set $S$ into subsets, and
2. a method for computing a bound (a lower resp. upper bound in the case of a minimization resp. maximization problem) for the objective function in a subset of $S$.

### 3.2.1 Partitioning Strategy

A straightforward partitioning strategy for the problem under consideration is the following: Let

$$S = \{(x_1, \ldots, x_n) \mid \sum_{i=1}^{n} c_i x_i \leq C,\ x_i \in \mathbb{N}\ (i = 1, \ldots, n)\}$$

be the feasible set, and let

$$S_{z_1,\ldots,z_k} = \{(z_1, \ldots, z_k, x_{k+1}, \ldots, x_n) \mid \sum_{i=k+1}^{n} c_i x_i \leq C - \sum_{j=1}^{k} c_j z_j,\ x_i \in \mathbb{N}\ (i = k+1, \ldots, n)\}$$

be the set of feasible solutions $(x_1, \ldots, x_n)$ with $x_1 = z_1, \ldots, x_k = z_k$. The subset $S_{z_1,\ldots,z_k}$ of $S$ is partitioned into the sets $S_{z_1,\ldots,z_k,z_{k+1}}$, where the number $z_{k+1}$ runs through all integer values $1, 2, \ldots$ such that

$$c_{k+1} z_{k+1} \leq C - \sum_{j=1}^{k} c_j z_j - \sum_{i=k+2}^{n} c_i, \tag{7}$$

i.e., such that each of the modules $k+2, \ldots, n$ can still obtain at least one version without violation of the budget constraint. So, if the numbers $z_1, \ldots, z_k$ of versions for module $1, \ldots, k$, respectively, have already been fixed, the next partitioning step consists in distinguishing all feasible version numbers for module $k+1$.

### 3.2.2 Bound Computation

The more delicate part of our task is the determination of a good bound for the objective function on a subset $S_{z_1,\dots,z_k}$. For this purpose, it is convenient to replace the objective function in (5) by its *logarithm*, such that the product is transformed into a sum. Because of the monotonicity of the logarithm, this does not change the optimal solution(s) $x = (x_1, \dots, x_n)$. In this way, we obtain, as an equivalent formulation to (5) – (6), the optimization problem

$$\text{Minimize } \; -\log R = -\sum_{i=1}^{n} \log\left(1 - \varphi_i(x_i)\right) \tag{8}$$

subject to the constraints (6). Therein, we have followed the general convention in nonlinear optimization to express the problem rather as a minimization than as a maximization problem.

The last formulation has a very fortunate property:

**Proposition 3.1.** For $x_i \in \mathbb{R}^+$ $(i = 1, \dots, n)$, the objective function in (8) is decreasing in each $x_i$, and convex in $(x_1, \dots, x_n)$.

**Proof.** Since sums of convex functions are convex again, it suffices to show that

$$\lambda_i(x_i) = -\log\left(1 - \varphi_i(x_i)\right)$$

is decreasing and convex for each $i$. We compute

$$\lambda_i'(x_i) = \frac{\varphi_i'(x_i)}{1 - \varphi_i(x_i)}$$

and

$$\lambda_i''(x_i) = \frac{1}{(1 - \varphi_i(x_i))^2}\,[\varphi_i''(x_i)\,(1 - \varphi_i(x_i)) + (\varphi_i'(x_i))^2].$$

Let us now use (1) and denote by $B_x(x, y)$ and $B_{xx}(x, y)$ the first resp. the second partial derivative of the Beta function $B(x, y)$ to the first argument variable $x$. The Beta function is defined as

$$B(x, y) = \int_0^1 t^{x-1}\,(1 - t)^{y-1} dt > 0, \tag{9}$$

hence

$$B_x(x, y) = \int_0^1 (\log t) \cdot t^{x-1}\,(1 - t)^{y-1} dt < 0 \tag{10}$$

and

$$B_{xx}(x, y) = \int_0^1 (\log t)^2 \cdot t^{x-1}(1 - t)^{y-1} dt > 0. \tag{11}$$

From (1), (9) and (11), we get $\varphi_i''(x_i) > 0$. Moreover, because of (9) and (10), the Beta function is positive and decreasing in its first argument, such that by (1), we have $0 < \varphi_i(x_i) < 1$ for $x_i > 0$. Hence $\lambda_i'(x_i) < 0$, which proves the first assertion of the proposition, and $\lambda_i''(x_i) > 0$, which proves the second assertion. □

Now consider a fixed subset $S_{z_1,\dots,z_k}$. For applying Branch–and–Bound, we need a lower bound for the objective in (8) on $S_{z_1,\dots,z_k}$. We obtain such a bound by *continuous relaxation* of the problem, replacing the constraints $x_i \in \mathrm{I\!N}$ in (6) by the weaker constraints $x_i \in [1, \infty[$ $(i = k+1, \ldots, n)$. Let

$$\tilde{S}_{z_1,\ldots,z_k} = \{(z_1, \ldots, z_k, x_{k+1}, \ldots, x_n) \mid \sum_{i=k+1}^{n} c_i x_i \leq C - \sum_{j=1}^{k} c_j z_j,\ x_i \in [1, \infty[\ (i = k+1, \ldots, n)\}$$

be the feasible set of the relaxed subproblem. Obviously, the objective of the optimal solution $(x_{k+1}^*, \ldots, x_n^*)$ of (8) on $\tilde{S}_{z_1,\ldots,z_k}$ is a lower bound for the objective of the optimal solution of (8) on $S_{z_1,\ldots,z_k}$. Moreover, $\tilde{S}_{z_1,\ldots,z_k}$ is a convex polyhedron. Therefore and because of Proposition 3.1, the auxiliary problem

$$\text{Minimize} \ -\sum_{i=k+1}^{n} \log\left(1 - \varphi_i(x_i)\right) \quad \text{on } \tilde{S}_{z_1,\ldots,z_k} \tag{12}$$

is a convex optimization problem. In (12), we have omitted the term

$$\sum_{i=1}^{k} \log\left(1 - \varphi_i(x_i)\right) = \sum_{i=1}^{k} \log\left(1 - \varphi_i(z_i)\right),$$

since it is constant.

Using the fact that the objective of (12) is decreasing in each $x_i$ (see Proposition 3.1), we can restrict the feasible set $\tilde{S}_{z_1,\ldots,z_k}$ of (12) to the facet

$$\bar{S}_{z_1,\ldots,z_k} = \{(z_1, \ldots, z_k, x_{k+1}, \ldots, x_n) \mid \sum_{i=k+1}^{n} c_i x_i = C - \sum_{j=1}^{k} c_j z_j,\ x_i \in [1, \infty[\ (i = k+1, \ldots, n)\}$$

where the budget $C$ is fully exhausted: If $\sum_{j=1}^{k} c_j z_j + \sum_{i=k+1}^{n} c_i x_i$ should be smaller than $C$ in an optimum, increasing some component $x_i$ until the cost limit is reached would produce a solution which is at least equally good, so we need not consider the solutions lying *not* on this facet. $\bar{S}_{z_1,\ldots,z_k}$ is still a convex polyhedron, therefore the restricted problem is convex again.

The constraint in the definition of $\bar{S}_{z_1,\ldots,z_k}$ gets simpler by a linear variable transformation: Let

$$C_{z_1,\ldots,z_k} = C - \sum_{j=1}^{k} c_j z_j - \sum_{i=k+1}^{n} c_i$$

be the remaining budget for redundancy in module $k+1, \ldots, n$, given that $z_j$ versions for module $j$ $(j = 1, \ldots, k)$ and the minimum of one version for each module $k+1, \ldots, n$ have already been paid, and let

$$y_i = y_i(x_i) = \frac{1}{C_{z_1,\ldots,z_k}} c_i(x_i - 1) \tag{13}$$

be the relative portion of $C_{z_1,\ldots,z_k}$ invested into redundancy for module $i$ $(i = k+1, \ldots, n)$. A vector $(z_1, \ldots, z_k, x_{k+1}, \ldots, x_n)$ lies on the facet $\bar{S}_{z_1,\ldots,z_k}$ if and only if for the corresponding variables $y_i$ the relation

$$\sum_{i=k+1}^{n} y_i = \frac{1}{C_{z_1,\ldots,z_k}} \left[\sum_{i=k+1}^{n} c_i x_i - \sum_{i=k+1}^{n} c_i\right] = \frac{1}{C_{z_1,\ldots,z_k}} \left[C - \sum_{j=1}^{k} c_j z_j - \sum_{i=k+1}^{n} c_i\right] = 1$$

holds. Thus we may rewrite (12) as

$$\text{Minimize } -\sum_{i=k+1}^{n} \log\left(1-u_i(y_i)\right) \tag{14}$$

subject to

$$\sum_{i=k+1}^{n} y_i = 1, \quad y_i \geq 0 \quad (i = k+1, \ldots, n), \tag{15}$$

where, with

$$x_i = x_i(y_i) = \frac{C_{z_1,\ldots,z_k}}{c_i}\, y_i + 1$$

denoting the inverse linear transformation to (13),

$$u_i(y_i) = \varphi_i(x_i(y_i)) \quad (i = k+1, \ldots, n). \tag{16}$$

The feasible set of the problem (14) – (15) is the $(n-k)$–dimensional standard simplex, and its objective is convex.

Convex problems can be solved by efficient algorithms, most of them being gradient procedures. In principle, we could apply such a procedure. The reader should notice, however, that the computation of the objective in (14) requires the evaluation of the Beta function for (in general) noninteger values of $x_i$ (cf. (16) and (1)), which takes much computation time. In order to avoid this, we shall demonstrate below that it is possible to get an exact lower bound also by using a piecewise linear interpolation of the objective, obtained from its values at *integer* argument values $x_i$. These values can be pre–computed for each feasible $x_i$ (i.e., for $x_i = 1, \ldots, \lfloor C/c_i \rfloor$) and stored in a table, such that access to them during the optimization phase is very fast. As it has turned out in our experiments, the runtime of the optimization algorithm can be considerably reduced in this way.

## 3.3 Piecewise Linear Interpolation

Let us start with the following observation: If the functions

$$f_i(y_i) = -\log(1 - u_i(y_i))$$

in (14) are replaced by any functions $\bar{f}_i(y_i)$ with the property that $f_i(y_i) = \bar{f}_i(y_i)$ for each $y_i$ corresponding to an *integer* $x_i$ via the linear transformation (13), then also the solution value of

$$\text{Minimize } \sum_{i=k+1}^{n} \bar{f}_i(y_i) \tag{17}$$

subject to

$$\sum_{i=k+1}^{n} y_i \leq 1, \quad y_i \geq 0 \quad (i = k+1, \ldots, n), \tag{18}$$

is a lower bound for the optimal solution value of (8) on $S_{z_1,\ldots,z_k}$. This is true because the optimal solution vector of (8) under (6) is contained (after transformation from $x_i$ to

$y_i$) in the feasible set given by (18), and the corresponding objectives $\sum f_i(y_i)$ and $\sum \bar{f}_i(y_i)$ are identical. Moreover, if the objective in (17) is decreasing in each variable $y_i$, then the observation above holds even if the constraint (18) is replaced by (15), by the same argument as it was used in Subsection 3.2.2 for replacing $\tilde{S}_{z_1,\ldots,z_k}$ by $\bar{S}_{z_1,\ldots,z_k}$.

We define $\bar{f}_i(y_i)$ as follows: Let, for $k+1 \le i \le n$,

$$\lambda_i(x_i) = -\log(1-\varphi_i(x_i)) = -\log(1-u_i(y_i(x_i))),$$

such that $f_i(y_i) = \lambda_i(x_i(y_i))$, and let

$$\bar{\lambda}_i(x_i) = \begin{cases} \lambda_i(x_i), & \text{if } x_i \in \mathbb{N}, \\ (x_i - \lfloor x_i \rfloor)\,\lambda_i(\lfloor x_i \rfloor + 1) + (\lfloor x_i \rfloor + 1 - x_i)\,\lambda_i(\lfloor x_i \rfloor), & \text{if } x_i \notin \mathbb{N}, \end{cases}$$

be the piecewise linear interpolation of $\lambda_i(x_i)$ ($x_i \ge 1$) with the integers as support points. Now we set

$$\bar{f}_i(y_i) = \bar{\lambda}_i(x_i(y_i)) \quad (i = k+1, \ldots, n). \tag{19}$$

Since, by Proposition 3.1, $\lambda_i(x_i)$ is decreasing and convex, it is immediately clear that also $\bar{\lambda}_i(x_i)$ and hence also $\bar{f}_i(y_i)$ are decreasing and convex, so we may obtain a lower bound by solving (17) subject to (15). If we do this by a gradient procedure, however, some care must be taken, since the objective is now only piecewise differentiable. In the remainder of this subsection, we shall outline that it is possible to overcome this difficulty by a suitable algorithm.

Because the functions $\bar{f}_i$ given by (19) are piecewise linear and decreasing, there exist at each $y_i \ge 0$ a left– and a right–sided derivative $\bar{f}_i'(y_i - 0) \le 0$ and $\bar{f}_i'(y_i + 0) \le 0$, respectively. It is easy to see that both one–sided derivatives must even be strictly negative. If $y_i$ corresponds to a $x_i \notin \mathbb{N}$, then $\bar{f}_i'(y_i - 0) = \bar{f}_i'(y_i + 0) = \bar{f}_i'(y_i)$.

For the sake of simplicity, let us assume in the sequel that $k = 0$ in (17); the modification for $k > 0$ is straightforward.

To a given $y$, let $m_1$ be an index with

$$\bar{f}_{m_1}'(y_{m_1} - 0) = \max_i \bar{f}_i'(y_i - 0) =: \Delta^{(1)}(y) < 0,$$

and $m_2$ an index with

$$\bar{f}_{m_2}'(y_{m_2} + 0) = \min_i \bar{f}_i'(y_i + 0) =: \Delta^{(2)}(y) < 0.$$

**Proposition 3.2.** For $k = 0$, the point $y = (y_1, \ldots, y_n)$ is local minimizer of (17) subject to (15), if and only if $\Delta^{(1)}(y) \le \Delta^{(2)}(y)$.

**Proof.** Assume $\Delta^{(1)}(y) \le \Delta^{(2)}(y)$, and consider a neighbor point $y_\epsilon = (y_1 + a_1\epsilon, \ldots, y_n + a_n\epsilon)$ of $y$, where $\sum_i a_i = 0$. W.l.o.g. we may choose the value $\epsilon$ that small that the intervals $]y_i, y_i + a_i\epsilon]$, re–transformed to the variables $x_i$, do not contain integers, i.e., that $\bar{f}_i(y_i)$ is linear in $[y_i, y_i + a_i\epsilon]$. Let

$$I^{(1)} := \{i \in \{1, \ldots, n\} \mid a_i < 0\},$$

and

$$I^{(2)} := \{i \in \{1, \dots, n\} \mid a_i > 0\}.$$

Furthermore, let $\bar{f}(y) = \sum_i \bar{f}_i(y_i)$ be the objective of (17). Then

$$\bar{f}(y_\epsilon) - \bar{f}(y) = \sum_{i \in I^{(1)}} \left(\bar{f}_i(y_i + a_i\epsilon) - \bar{f}_i(y_i)\right) + \sum_{i \in I^{(2)}} \left(\bar{f}_i(y_i + a_i\epsilon) - \bar{f}_i(y_i)\right)$$

$$= \sum_{i \in I^{(1)}} a_i\epsilon \cdot \bar{f}_i'(y_i - 0) + \sum_{i \in I^{(2)}} a_i\epsilon \cdot \bar{f}_i'(y_i + 0) \geq \epsilon \left[\Delta^{(1)}(y) \sum_{i \in I^{(1)}} a_i + \Delta^{(2)}(y) \sum_{i \in I^{(2)}} a_i\right]$$

$$\geq \epsilon\, \Delta^{(2)}(y) \left(\sum_{i \in I^{(1)}} a_i + \sum_{i \in I^{(2)}} a_i)\right) = 0.$$

So $\bar{f}(y_\epsilon) \geq \bar{f}(y)$ for each neighbor point $y_\epsilon$ of $y$, which means that $y$ is a local minimizer.

Conversely, assume that $\Delta^{(1)}(y) > \Delta^{(2)}(y)$, and consider the neighbor point

$$y_\epsilon = (y_1, \dots, y_{m_1-1}, y_{m_1} - \epsilon, y_{m_1+1}, \dots, y_{m_2-1}, y_{m_2} + \epsilon, y_{m_2+1}, \dots, y_n)$$

of $y$. For this point,

$$\bar{f}(y_\epsilon) - \bar{f}(y) = \epsilon\,[-\bar{f}_{m_1}'(y_{m_1} - 0) + \bar{f}_{m_2}'(y_{m_2} + 0)] = \epsilon\,[-\Delta^{(1)}(y) + \Delta^{(2)}(y)] < 0,$$

so $y$ is not a local minimizer. □

The last lemma leads immediately to an iterative algorithm for solving (17) (with $k = 0$) under (15):

**procedure** `SolveAuxiliaryProblem` {
  choose an arbitrary initial point $y$ in the simplex (15);
  **repeat** {
    for the current $y$, determine the indices $m_1$ and $m_2$ and the corresponding
    values $\Delta^{(1)}(y)$ and $\Delta^{(2)}(y)$;
    **if** $\Delta^{(1)}(y) \leq \Delta^{(2)}(y)$, stop ($y$ is then local minimizer);
    **else** do line search along the line
      $(y_1, \dots, y_{m_1-1}, y_{m_1} - \eta, y_{m_1+1}, \dots, y_{m_2-1}, y_{m_2} + \eta, y_{m_2+1}, \dots, y_n)$
      where $\eta \geq 0$ and the simplex is not left, i.e., $\eta \leq \min(y_{m_1}, 1 - y_{m_2})$,
      and set $y$ equal to the minimizer along this line;
  }
}

By the Corollary to Lemma 3.1, the local minimum determined in this way is a global minimum and can therefore be used as a lower bound.

# 4 Experimental Results

For experimental computations, 10 test cases were randomly generated for each of the combinations of parameters described below (see Table 1 to Table 6).

Since the Dynamical Programming approach needs integer values, all cost variables $c_i$ and the budget $C$ were converted to integers, and a parameter $SD$ was introduced, giving the number of possible significant digits. E.g., if one calculates in thousands of Dollars with 4 significant digits, the range of 1 to 9999 thousands of Dollars may be used for cost variables or budget. This "granularity" parameter showed to be crucial for the performance of Dynamic Programming (DP) versus Branch–and–Bound (BAB). In our experiments, we tested $SD = 3$ and $SD = 5$.

As values for the number of modules, $n = 10$, $n = 20$ and $n = 30$ were chosen. The correlation levels were set to $\theta = 0.0001$ (near to independence), $\theta = 0.01$ (moderate dependency) and $\theta = 100$ (high dependency) to cover the whole range of possibilities.

Another crucial parameter is the ratio between the budget $C$ and the sum of the costs $c_i$, which we call the *cost factor* $CF = C \,/\, \sum c_i$. A cost factor of 2, e.g., means that the available budget is the double amount of the cost for providing only the minimum of one version for each module. For low cost factor, the calculations are relatively easy as the number of possible combinations is restricted, but for only slightly higher cost factors, combinatorial explosion occurs. We used cost factors in the range between 1.25 and 3. However, not for all combinations of cost factors with other parameters, the computations could be completed in reasonable time.

Furthermore, Complete Enumeration (CE) was also implemented for comparison reasons, but could be performed only for $n = 10$ and moderate cost factor.

Table 1 to Table 6 give the average computation time $t$ in milliseconds for the different combinations of parameters; missing values are due to computations exceeding the time limit. All computations were performed on a Pentium II 550 MHz, 512 MB memory, with code written in ANSI-C.

Chart 1 to Chart 9 below give a graphical overview (using a logarithmic scale for time) of the numerical results with:

| | | | |
|---|---|---|---|
| $\times$ | Branch-and-Bound (SD 3) | O | Branch-and-Bound (SD 5) |
| $*$ | Dynamical Programming (SD 3) | $\star$ | Dynamical Programming (SD 5) |
| $\bullet$ | Complete Enumeration (CE) | | |

Table 1: Significant Digits SD=3, Modules $n = 10$

| n=10 | $\theta = 0.0001$ | | | $\theta = 0.01$ | | | $\theta = 100$ | | |
|---|---|---|---|---|---|---|---|---|---|
| CF | CE | BAB | DP | CE | BAB | DP | CE | BAB | DP |
| 1.25 | 87.2 | 26.1 | 11.0 | 85.1 | 24.0 | 11.0 | 85.1 | 18.0 | 11.0 |
| 1.50 | 3318.7 | 41.0 | 34.1 | 3318.8 | 40.1 | 34.0 | 3286.8 | 33.0 | 36.0 |
| 1.75 | 27130.0 | 93.1 | 64.1 | 33911.8 | 85.1 | 65.1 | 34594.8 | 50.1 | 68.1 |
| 2.00 | 214780.8 | 106.2 | 111.2 | 194735.0 | 120.1 | 113.2 | 208094.2 | 74.1 | 118.2 |
| 2.25 | | 286.4 | 157.2 | | 162.2 | 160.2 | | 92.2 | 169.2 |
| 2.50 | | 2388.5 | 219.3 | | 189.3 | 222.3 | | 131.2 | 237.4 |
| 2.75 | | 24309.0 | 291.4 | | 403.6 | 294.4 | | 140.2 | 313.5 |
| 3.00 | | 701131.2 | 373.5 | | 973.4 | 378.5 | | 232.4 | 405.6 |

Table 2: Significant Digits SD=3, Modules $n = 20$

| n=20 | $\theta = 0.0001$ | | $\theta = 0.01$ | | $\theta = 100$ | |
|---|---|---|---|---|---|---|
| CF | BAB | DP | BAB | DP | BAB | DP |
| 1.25 | 201.3 | 66.1 | 206.3 | 67.1 | 147.2 | 67.1 |
| 1.50 | 451.6 | 226.3 | 413.6 | 228.3 | 392.5 | 234.3 |
| 1.75 | 851.3 | 443.6 | 1231.8 | 457.6 | 1045.5 | 468.7 |
| 2.00 | 1390.0 | 815.2 | 1246.8 | 813.1 | 1442.1 | 850.2 |
| 2.25 | 5974.6 | 1143.6 | 1816.6 | 1151.7 | 1932.7 | 1211.7 |
| 2.50 | 276269.3 | 1613.3 | 2217.2 | 1626.4 | 2904.2 | 1712.5 |
| 2.75 | | 2165.1 | 6095.8 | 2184.1 | 3201.6 | 2300.3 |
| 3.00 | | 2799.0 | 21140.4 | 2821.1 | 4203.0 | 2972.3 |

Table 3: Significant Digits SD=3, Modules $n = 30$

| n=30 | $\theta = 0.0001$ | | $\theta = 0.01$ | | $\theta = 100$ | |
|---|---|---|---|---|---|---|
| CF | BAB | DP | BAB | DP | BAB | DP |
| 1.25 | 430.6 | 204.3 | 426.6 | 208.3 | 528.8 | 214.3 |
| 1.50 | 1525.2 | 706.0 | 1672.4 | 717.0 | 1551.3 | 743.1 |
| 1.75 | 2927.2 | 1495.1 | 3304.8 | 1493.2 | 3432.9 | 1556.2 |
| 2.00 | 5335.6 | 2556.7 | 7109.2 | 2591.7 | 6295.0 | 2677.9 |
| 2.25 | 12521.0 | 3922.6 | 6439.2 | 3936.6 | 12291.7 | 4104.9 |
| 2.50 | | 5563.0 | 10842.6 | 5611.1 | 10898.7 | 5839.4 |

Table 4: Significant Digits SD=5, Modules $n = 10$

| n=10 | $\theta = 0.0001$ | | $\theta = 0.01$ | | $\theta = 100$ | |
|---|---|---|---|---|---|---|
| CF | BAB | DP | BAB | DP | BAB | DP |
| 1.25 | 25.0 | 1195.7 | 24.0 | 1199.7 | 18.1 | 1214.7 |
| 1.50 | 41.1 | 3666.3 | 41.0 | 3656.3 | 33.0 | 3744.4 |
| 1.75 | 94.2 | 7082.2 | 85.2 | 7123.3 | 47.0 | 7429.7 |
| 2.00 | 106.2 | 12255.6 | 120.2 | 12476.0 | 83.2 | 12910.5 |
| 2.25 | 282.4 | 17556.2 | 165.3 | 17899.8 | 90.2 | 18811.1 |
| 2.50 | 2390.4 | 24642.5 | 189.2 | 25014.0 | 138.2 | 26366.9 |
| 2.75 | 24190.8 | 33214.8 | 403.6 | 33287.9 | 159.2 | 35395.9 |
| 3.00 | 692310.5 | 42721.5 | 978.5 | 42776.5 | 235.3 | 45131.9 |

Table 5: Significant Digits SD=5, Modules $n = 20$

| n=20 | $\theta = 0.0001$ | | $\theta = 0.01$ | | $\theta = 100$ | |
|---|---|---|---|---|---|---|
| CF | BAB | DP | BAB | DP | BAB | DP |
| 1.25 | 195.3 | 7137.2 | 207.3 | 7244.4 | 153.2 | 7364.6 |
| 1.50 | 450.6 | 25081.1 | 409.5 | 25087.0 | 415.6 | 25902.3 |
| 1.75 | 848.2 | 49858.6 | 1230.8 | 50401.5 | 956.4 | 53021.2 |
| 2.00 | 1391.0 | 94408.8 | 1234.7 | 93451.3 | 1332.9 | 95431.2 |
| 2.25 | 5974.6 | 142532.0 | 1815.6 | 148131.0 | 1966.8 | 145788.7 |
| 2.50 | 277633.2 | | 2202.1 | | 2987.3 | |
| 2.75 | | | 5982.6 | | 3673.3 | |
| 3.00 | | | 20955.1 | | 4895.0 | |

Table 6: Significant Digits SD=5, Modules $n = 30$

| n=30 | $\theta = 0.0001$ | | $\theta = 0.01$ | | $\theta = 100$ | |
|---|---|---|---|---|---|---|
| CF | BAB | DP | BAB | DP | BAB | DP |
| 1.25 | 420.6 | 22656.5 | 422.6 | 22586.5 | 515.6 | 23146.3 |
| 1.50 | 1586.3 | 92389.8 | 1710.5 | 93415.3 | 1960.9 | 97585.3 |
| 1.75 | 2898.2 | | 3176.6 | | 3866.6 | |
| 2.00 | 5408.8 | | 7108.2 | | 7782.2 | |
| 2.25 | 12510.0 | | 6101.8 | | 10267.7 | |
| 2.50 | | | 10845.6 | | | |

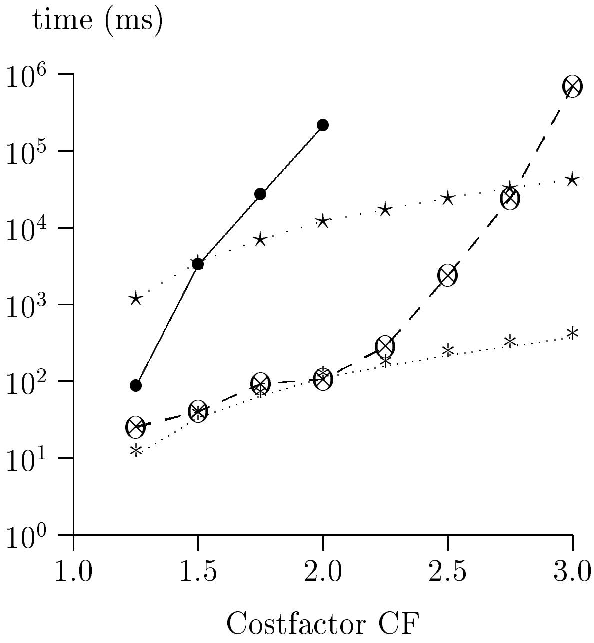


**Chart 1:** $n = 10, \theta = 0.0001$

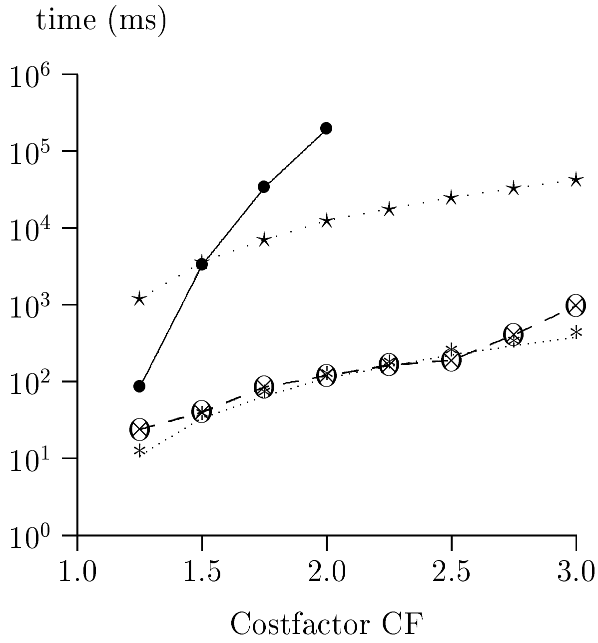


**Chart 2:** $n = 10, \theta = 0.01$

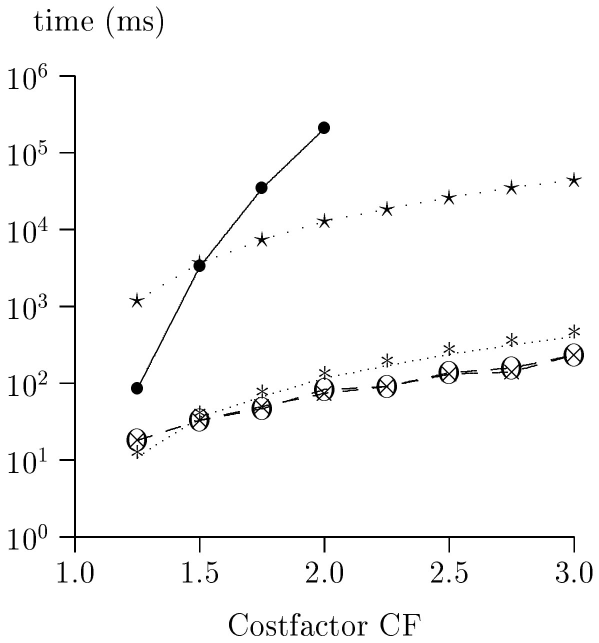


**Chart 3:** $n = 10, \theta = 100$

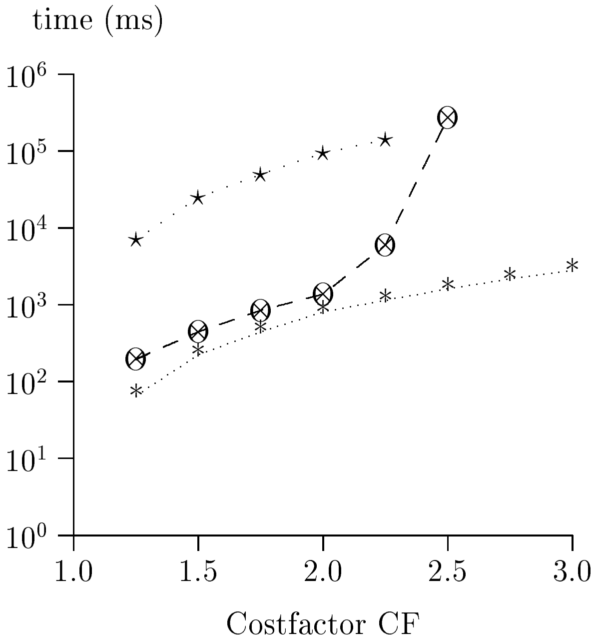


**Chart 4:** $n = 20, \theta = 0.0001$

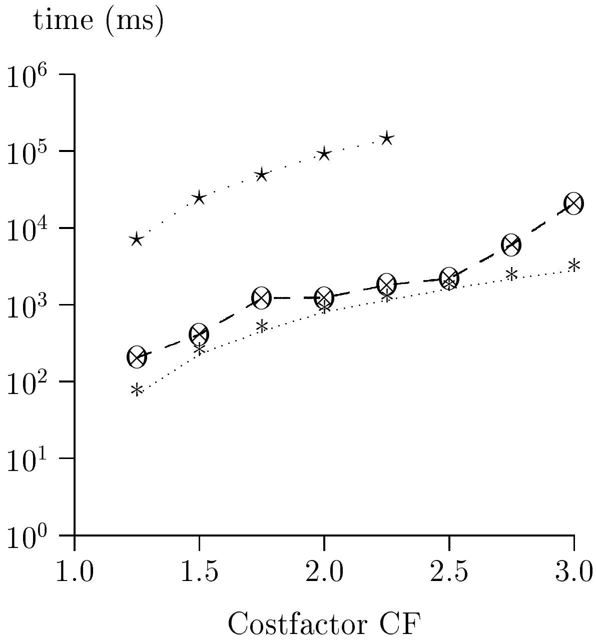


**Chart 5:** $n = 20, \theta = 0.01$

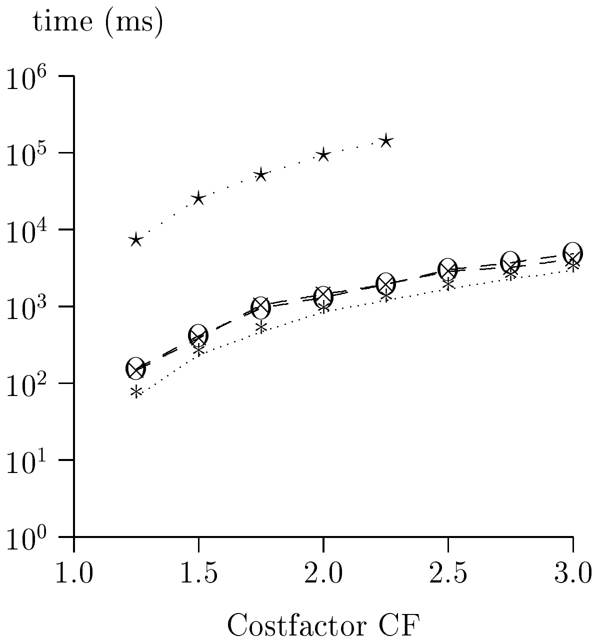


**Chart 6:** $n = 20, \theta = 100$

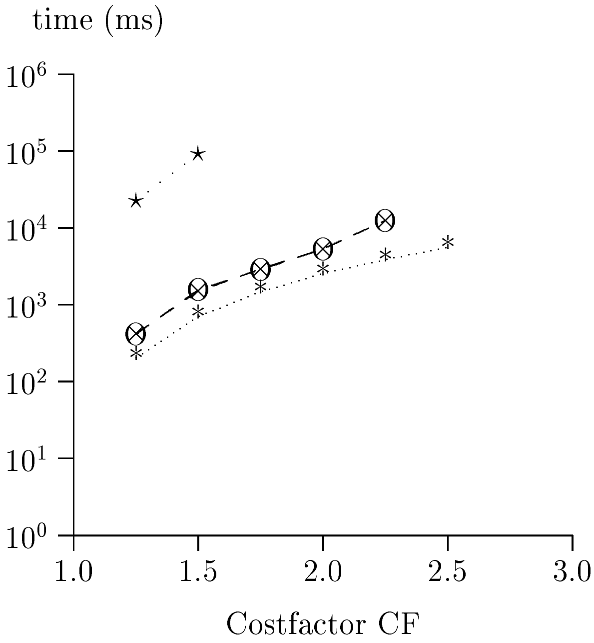


**Chart 7:** $n = 30, \theta = 0.0001$

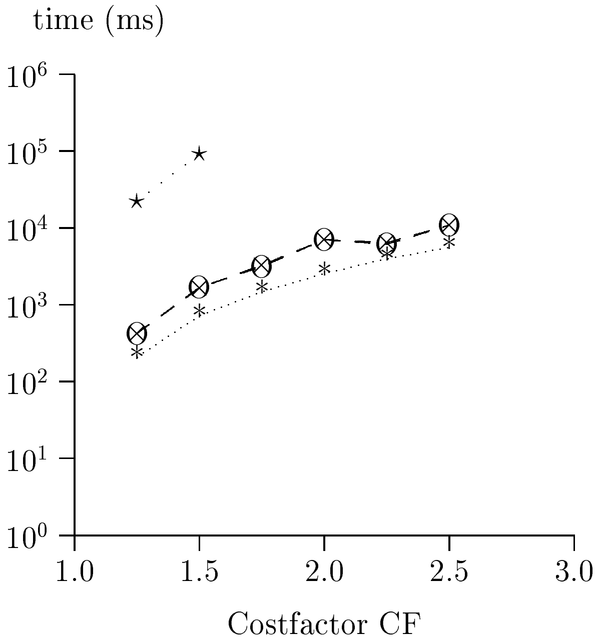


**Chart 8:** $n = 30, \theta = 0.01$

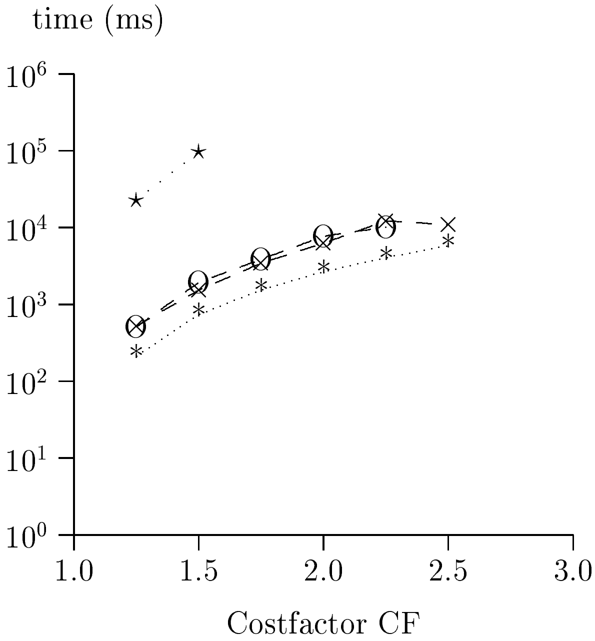


**Chart 9:** $n = 30, \theta = 100$

From the results above we conclude that — as could be expected — the Branch-and-Bound algorithm is robust against the number of significant digits, whereas the Dynamic Programming approach is very sensitive to this parameter. The curves for BAB with $SD = 3$ or $SD = 5$ are almost not discernable. While DP with $SD = 3$ performs very well under all tested conditions, DP with $SD = 5$ takes considerably more time, and in many cases, we were not able to complete the computations for this granularity of cost values.

Chart 1 and 4 show that the performance of BAB is not very good in the case near independence ($\theta = 0.0001$) with high cost factor $CF$. In this area (which is not the most challenging within our investigation, because the situation of independence has already been studied before by other authors), BAB can be outperformed by DP. For moderate and high dependency, on the other hand, BAB performs quite well and is able to solve problems with a number of significant digits that is out of reach for DP.

# 5 Conclusion

We have specified a Branch–and–Bound technique for computing optimal redundancy levels in fault–tolerant systems using recovery blocks with dependently failing module versions, and shown experimentally that the suggested technique is sufficiently fast for problem instances of an order of magnitude as it occurs in practice. Our investigation might be considered rather as a starting point for the reliability optimization of redundant systems under failure correlation; there are some limitations which will have to be overcome by future research. Let us shortly discuss the most important of them:

1. We have assumed that the test segment of each recovery block is fault–free or, at least, its failure probability is negligible compared to that of the module versions. There are situations where this is the case: often, checking a result is much easier than obtaining the result and hence distinctly less error–prone. There are, however, also situations where the mentioned assumption is not satisfied. For these cases, we would need a suitable extension of our approach modelling possible failures of the test segments.

2. In our model, all versions of a module are available at the same cost. In practice, it may happen that prices for the single versions differ. An extension of our technique to treat this situation should be rather straightforward, since in such a case it is possible to arrange the versions in an order of increasing costs, replacing the total cost $c_i x_i$ for the first $x_i$ versions by $c_i(x_i)$ with an increasing function $c_i$. This does not essentially affect our technique.

3. Much harder to treat is a situation where (also) the (estimated) *reliabilities* of the module versions may vary. Up to now, there exists not even a standard technique to *compute* the overall reliability of a system consisting of versions with different reliabilities and correlated failures; a possible approach has been suggested in [6].

4. We have only considered recovery blocks as fault–tolerant structures. Of course, also other structures, such as voting systems, may be investigated.

It is probable that in the near future, these open problems will be solved in a satisfactory way, such that reliability optimization of redundant systems on the realistic assumption of failure correlation between versions can proceed from theory to application.

# References


[1] Belli, F., Jedrzejowicz, P., "An approach to the reliability optimization of software with redundancy", *IEEE Trans. Software Eng.*, vol. SE-17, 310 – 312 (1991).

[2] Berman, O., Ashrafi, N., "Optimization models for reliability of modular software systems", *IEEE Trans. Software Eng.*, vol. SE-19, 119 – 1123 (1993).

[3] Berman, O., Kumar, U.D., "Optimization models for complex recovery block schemes", *Computers and Operations Research*, vol. 26, 525 – 544 (1999).

[4] Eckhardt, D. E., Lee, L. D., "A theoretical basis for the analysis of multiversion software subject to coincident errors", *IEEE Trans. Software Eng.*, vol. SE-11, 1511 – 1517 (1985).

[5] Gutjahr, W. J., "Reliability optimization of redundant software with correlated failures", *Proc. ISSRE '98* (International Symposium on Software Reliability Engineering, Paderborn, Nov. 1998), 293 – 302 (1998).

[6] Gutjahr, W. J., "A reliability model for nonhomogenous redundant software versions with correlated failures", Technical Report 02 (2000), Dept. of Statistics and Decision Support Systems, University of Vienna.

[7] Knight, J. C., Leveson, N. G., "An experimental evaluation of the assumption of independence in multiversion programming", *IEEE Trans. Software Eng.*, vol. SE-12, 96 – 109 (1986).

[8] Nicola, V. F., Goyal, A., "Modeling of correlated failures and community error recovery in multiversion software", *IEEE Trans. Software Eng.*, vol. SE-16, 350 – 359 (1990).

[9] Papadimitriou, H., Steiglitz, I., *Combinatorial Optimization: Algorithms and Complexity*, Prentice–Hall (1982).

[10] Poore, J. H., Trammell, C. J., "Application of statistical science to testing and evaluating software intensive systems", in: *Statistics, Testing, and Defense Acquisition*, Washington: National Academy Press (1998).

[11] Tomek, L. A., Muppala, J. K., Trivedi, K. S., "Modeling correlation in sofware recovery blocks", *IEEE Trans. Software Eng.*, vol. 19, no. 11, 1071 – 1086 (1993).