A MULTI-AGENT BASED APPROACH FOR SOLVING THE REDUNDANCY ALLOCATION PROBLEM

THESIS

Submitted to

Temple University Graduate Board

in Partial Fulfillment

of the Requirements for the Degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

by

Zhuo Li

August 2011

Abstract

Redundancy Allocation Problem (RAP) is a well known mathematical problem for modeling series-parallel systems. It is a combinatorial optimization problem which focuses on determining an optimal assignment of components in a system design. Due to the diverse possible selection of components, the RAP is proved to be NP-hard. Therefore, many algorithms, especially heuristic algorithms were proposed and implemented in the past several decades, committed to provide innovative methods or better solutions.

In recent years, multi-agent system (MAS) is proposed for modeling complex systems and solving large scale problems. It is a relatively new programming concept with the ability of self-organizing, self-adaptive, autonomous administrating, etc. These features of MAS inspire us to look at the RAP from another point of view. An RAP can be divided into multiple smaller problems that are solved by multiple agents. The agents can collaboratively solve optimal RAP solutions quickly and efficiently.

In this research, we proposed to solve RAP using MAS. This novel approach, to the best of our knowledge, has not been proposed before, although multi-agent approaches have been widely used for solving other large and complex nonlinear problems. To demonstrate the capability of this approach, we analyzed and evaluated four benchmark RAP problems in the literature. From the results, the MAS approach is shown as an effective and extendable method for solving the RAP problems.

Subject Category:

Natural, and Physical Sciences − Computer science

Key words:

RAP, Redundancy allocation problem, algorithm, combinatorial optimization, multi-agent system.

Acknowledgements

I would like to express my gratefulness to my academic advisor Dr. Li Bai, my program advisor Dr. Olga Vilceanu, and thesis committee members Drs. Saroj Biswas, and Joseph Picone. Thanks to their supports and every aspects of help. I really appreciate it. I would also like to thank Department of Electrical and Computer Engineering, Temple University. Thanks to my parents and all my lab colleagues James Ren, Michael Korostelev, Yaoyao Huang, Zhao Cheng, etc. I will treasure my experience and friendship with everyone here throughout my life.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction and Background

This thesis addresses an important aspect of optimization search using multi-agent approach. The goal is to develop an efficient, simple and modulized algorithm which can be extended in agents and concurrently on agent threads. This novel approach, to the best of our knowledge, has not been proposed before, although multi-agent approaches have been widely used for solving other large and complex nonlinear problems.

1.1 Redundancy Allocation Problem

As humans' ordinary life more and more relies on advanced technology, e.g. GPS, internet, and sensor networks, the reliability of either a hardware system or a software service becomes one of the most critical concerns in a system design.

Generally, system reliability can be improved either by incremental improvements of component reliability or by provision of redundancy components in parallel; both methods result in an increase in system cost.

Redundancy Allocation Problem (RAP) is a mathematical model for evaluating system reliability under some given constraints such as cost and weight. It is broadly used in a variety of practical circumstances, especially in the field of electrical engineering and industrial engineering. The practical application of RAP is usually involved in circuit design, power plant components replacement, consumer electronics industry, etc. Take the launch of rocket as an example. Engineers put redundancy at some critical parts to ensure the success of launching. However, due to the budget and carrying capacity of a rocket, the redundancy must be addressed with care.

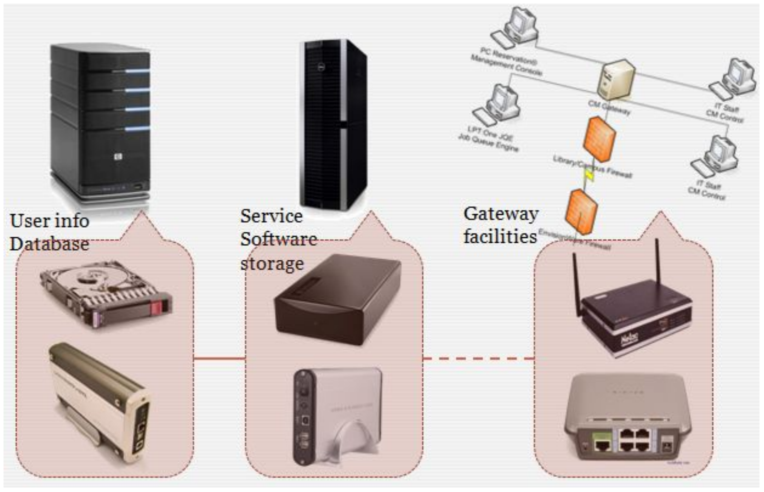


Figure 1.1: An Example of RAP application on server systems

The consumer electronics industry is another such application where new system designs are composed largely of standard component types. Shown in figure 1.1, a portal web site like Yahoo which provides highly integrated services, cannot afford losing users information nor suspending service resulted by the crash of servers. To guarantee the quality of service, they separate the business of registration, downloading, email service etc, onto different servers and use multiple servers to provide the identical functionality. When purchasing these servers, they have vast choices of brands which vary on the aspects of capacity, price, power consumption, etc. The company would like to increase the system reliability while limiting the redundancy to a certain level.

Figure 1.2: Google data container center (left) and its server design (right)

Google is famous of its data storage solution “Google data container center” 1.2. As we read from some of their released technical reports for 2010 [3], they use three power types for each of their data station: grid, generator and a 12-v battery in each server. When they unlocked this once-secret server design, Microsoft and Intel were all amazed with saying “it is really a unique design”. From this, we can see how much these IT companies emphasize on redundancy.

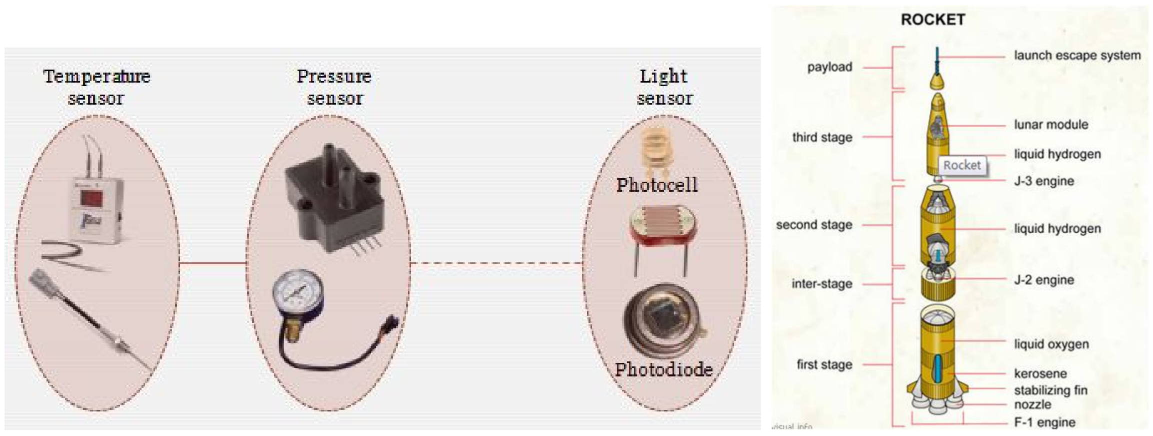


Figure 1.3: Examples of RAP application on sensor networks and rockets

Another practical scenario is sensor networks in figure 1.3. Engineers may deploy dozens of functionally identical sensors to a part of a building or a water dam because after construction, some parts are sealed with cement which makes it extremely hard to replace the broken sensors by reconstructing. Hence, a well designed combination of sensors will both increase the system reliability and reduce the cost of future maintenance.

From these examples, we can see that redundancy allocation is usually helpful and sometimes even necessary in ordinary life. RAP is a mathematical model abstracted from the physical world, whose goal is to maximize the reliability of a system under multiple constraints. Conventionally, constraints specify cost and weight of the available components.

The RAP is classified into combinatorial optimization problem [4]. Due to the diverse combination of components, RAP is known to be NP-hard, proved in [5]. To solve this type of problems, conventionally, there are three types of algorithms: exact algorithms, approximate algorithms and heuristics.

1) Exact algorithms are able to determine the global optimum. However, since the RAP is NP-hard, one challenge for exact algorithms is, as the input size of the RAP increases, they are powerless to find the optimization in limited time because the computation required increases exponentially [6]. Another challenge which prevents the implementation of regular exact algorithms is that the objective function of RAP is nonlinear.

2) Approximation algorithms seek an approximation that is close to the optimal solution. It may use either a deterministic or a random strategy.

3) Heuristics are a family of algorithms which try to solve a problem by "hypothesis $\rightarrow$ examining $\rightarrow$ hypothesis $\rightarrow$ examining $\cdots$ " mechanism. Instead of promising to provide an optimal solution, they usually find reasonably good solutions reasonably quickly.

One major difference between exact algorithms and heuristic algorithms is that, given a certain initial condition, one can always predict the result obtained by exact algorithm in each round of iteration. Thus, if we run the program several times using the same initial condition, we will get the same results in same amount of steps. In contrast for heuristic algorithms, one cannot do so because heuristic involves guessing and randomization. For example, if we run a word puzzle programm based on genetic algorithm twice for the same input, we may get the same output in totally different steps or even get a different output.

1.2 Meta-heuristic

In computer science, a heuristic is a technique designed to solve a problem that ignores whether the solution can be proven to be correct, but which usually produces a good solution or solves a simpler problem that contains or intersects with the solution of the more complex problem.

A meta-heuristic is a set of concepts which are used to define heuristic methods that can be applied to a wide set of different problems, In other words, a meta-heuristic can be seen as a general algorithmic framework which can be applied to different optimization problems with relatively few modifications to adapt them to a specific problem. [7]

Heuristic search techniques becomes a hot topic beginning in the 1990's. There are a family of heuristic search algorithms, including *genetic algorithm*(GA), *Tabu Search*(TS), *Simulated Annealing*(SA), *ant colony optimization*(ACO), and *Neural Networks*, etc.

1.3 Multi-agent Programming

The definition of *multi-agent system*(MAS) in the book "Multi-agent system" [8] is a system composed of multiple interacting intelligent agents. Multi-agent systems can be used to solve problems which are difficult or impossible for an individual agent or a monolithic system to solve. Examples of problems which are appropriate to multi-agent systems research include online trading, [9] disaster response,[2] and modeling social structures, etc.

Multi-agent systems are also referred to as "self-organized systems". They usually consist of multiple autonomous entities having different information and/or diverging interests. [8] MAS tends to find the best solution for their problems "without intervention", which have a high similarity to physical phenomena. The applications of MAS touch all aspects from artificial intelligence to algorithms, robotics to game theory, control theory to logic, just to name a few.

The main feature which is achieved when developing multi-agent systems, if they work, is flexibility, since a multi-agent system can be added to, modified and reconstructed, without the need for detailed rewriting of the application. These systems also tend to be rapidly self-recovering from failure, usually due to the heavy redundancy of components and the self-managed features referred to the above.

1.4 Proposed Approach

By analyzing the problem and investigating the multi-agent programming technique, we propose an approach based on multi-agent programming, hoping to reach the optimization in competitive time or shorter time using simple computation logics. Also, the extendability of MAS can be used for other RAP systems without any modification of source codes of MAS core logic reasoning engines in order to find a good optimal allocation of redundancy components.

1.5 Organization of the Thesis

The overall thesis is organized as follows. firstly we will give an introduction to RAP and multi-agent concept, in which mathematic formulation and literature review will be covered, I will then discuss in detail using our multi-agent based approach to solve the RAP. Comparison of results and analysis will be provided at the end.

Chapter 2

Mathematical Formulation

According to (Fyffe 1968) [6], the original mathematical formulation of RAP is as follows:

It is assumed that the system under consideration consists of s functional units. (In this proposal, we treat each unit as a subsystem as shown in Figure 2.1) Different functional units are connected in series. The failure of any functional unit will cause system failure, and failure of any unit in the system is assumed to be independent. For each of the s functional units there exist several choices of design alternatives that the system designer can employ in order to meet the allocated reliability requirement, which means there are multiple, functionally equivalent components available to be used in each subsystem. Such scheme is referred to as a serial-parallel system. Specifically, each component has a reliability associated with a certain amount of cost and weight.

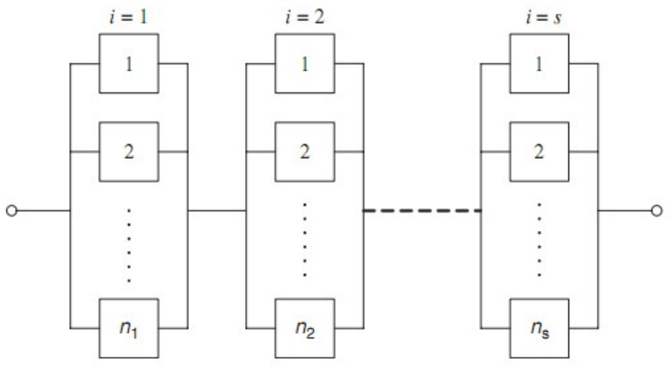


Figure 2.1: A series-parallel system configuration

In the original formulation, it is modeled as only a single type of identical components are allowed to selected in one subsystem configuration. Once a component type is selected to provide a required function, only components of the same type can be used to provide redundancy. Let m_i denote the number of design alternatives or say number of types of components available for the ith subsystem with a specified inherent unit reliability r_{j_i}, x_{j_i} denote the number of components used in subsystem i, and use R_i to denote the known reliability function of the ith subsystem. So, according to the above schematic, the system reliability is:

$$R = \prod_{i=1}^{s} R_i = \prod_{i=1}^{s} [1 - (1 - r_{j_i})^{x_{j_i}}] \tag{1}$$

where s denotes the total number of subsystems, and $1 \leq j_i \leq M_i$.

The constraints are:

$$\sum_{i=1}^{s} c_{j_i} \times x_{j_i} \leq C$$

$$\sum_{i=1}^{s} w_{j_i} \times x_{j_i} \leq W \tag{2}$$

In some developed formulations, *e.g. Coit et al.* [1], multiple components may be selected and arranged in parallel for a subsystem configuration. Thus, the formulation for computing the reliability of subsystem differs a little bit.

In this proposal, we follow the formulation in [1], and still use m_i to denote the number of design alternatives available for the ith subsystem with a specified inherent unit reliability r_{ij} for each component. We use x_{ij} to denote the number of components of type j used in subsystem i, where ($1 \leq j \leq m_i$, and $1 \leq i \leq s$). We let $R_i(r_{ij}, x_{ij})$ denote the known reliability function of the ith subsystem. Thus, the reliability of ith subsystem is:

$$R_i(r_{ij}, x_{ij}) = 1 - \prod_{j=1}^{m_i} (1 - r_{ij})^{x_{ij}} \tag{3}$$

and the reliability of the whole system can be depicted in the form of the following function:

$$R = \prod_{i=1}^{s} R_i(r_{ij}, x_{ij}) = \prod_{i=1}^{s} \left(1 - \prod_{j=1}^{m_i} (1 - r_{ij})^{x_{ij}} \right) \tag{4}$$

The main goal of solving the RAP is to find an assignment of components, which maximizes the reliability of the system while keeping the total cost and weight of the components to meet the constraint requirements. This is expressed in the following formulations:

$$\mathbf{max} \prod_{i=1}^{s} \left(1 - \prod_{j=1}^{m_i} (1 - r_{ij})^{x_{ij}} \right)$$

subjected to the constraints:

$$\sum_{i=1}^{s} \sum_{j=1}^{m_i} c_{ij} \times x_{ij} \leq C$$

$$\sum_{i=1}^{s} \sum_{j=1}^{m_i} w_{ij} \times x_{ij} \leq W$$

$$x_{ij} \in Z^{+}$$

From the formulation, we can see that the decision variables are the component choices (from the available discrete choices), and the redundancy levels (the number of functionally identical components used in one subsystem).

There are more variant RAPs proposed in the past decades than presented here. Before we go any further, we would like to specify the RAP we are investigating.

What is redundancy? As shown in the figure, units in parallel are referred to as redundant units [10]. There are two redundancy schemes, named active and standby. If all redundant components operate simultaneously from time zero, even though the system needs only one at any given time, such an arrangement is called active redundancy [11]. Otherwise, the redundancy scheme is called standby redundancy. There are three variants of the standby redundancy, referred to as cold standby, warm standby, and hot standby. In cold standby redundancy, the component does not fail before it operates. In warm standby redundancy, the component is more prone to failure before operation than the cold standby components [1]. In hot standby redundancy, the failure pattern of component does not depend on whether the component is idle or in operation [12]. Most Reliability design problems are formulated considering active redundancy.

On the aspect of component operating state, there are different cases of the series-parallel allocation problem. Explained in [13], components can be multi state other than just binary, thus, the system can have a range of different performance states. The RAP we consider in this proposal only involves components with binary state, that is operate or fail.

Thus, the specification is followed by the typical assumptions:

- The states of components and the system are either good or failed;
- Failure of any unit in the system is assumed to be an independent event;
- Failed components do not damage the system, and are not repaired;
- The failure rates of components when not in use are the same as when in use (i.e., active redundancy);
- Component attributes (reliability, weight, and cost) are known and deterministic;
- The supply of components is unlimited. [1] [14] [2]

Chapter 3

Literature Review

3.1 Literature Review

As mentioned previously, there are usually several approaches to increase system reliability, including enhancement of component reliability, provision of redundant components in parallel or a combination of both. The first approach has been well developed in the 1960s and 1970s as is documented in [15]. The second approach is the concern of RAP which is going to be discussed in this thesis.

The redundancy allocation problem dates back to the 60's last century and was appeared as reliability allocation problem in [6]. A dynamic programming formulation of the RAP was given in 1971 [16] and a hybrid "dynamic programming/depth-first search" algorithm is presented by Kevin *et al.* [17]. A stochastic formulations of the RAP was proposed by Coit *et al.* in 1996 [18], and a linear approximation formulation was proposed in 2002 [19]. Different algorithms have been implemented to solve the RAP. Fyffe *et al.* gives an optimal solution by using dynamic programming with Lagrange multiplier [6]. However, since dynamic programming is unsuitable for problems with multiple constraints, there are some inconvenience of using it, which will be discussed in next section. Luus [20] gives an exact nonlinear integer programming method in 1975. used a Billionnet gives an approximate feasible solution by linearization method and integer programming [10].

A varieties of heuristic algorithms are utilized as well. Coit *et al.* utilized genetic algorithm (GA) in different ways to solve RAP in [21], [22], [23], and implementing Tabu search (TS) is discussed in [24], [25], and [26]. Besides these, other heuristics have been attempted in the past decades, such as variable neighborhood search algorithm (VNS)

[2] [27] [28], ant colony optimization algorithm(ACO, introduced by Dorigo in his Ph.D. thesis) [14] [29] [30], surrogate constraint method [31], scaling method [32], Max-Min approach [33], etc. Most of these algorithms uses the test problem given by Fyffe *et al.* in [6] and the modified version given by Nakagawa *et al.* in [34] as benchmarks and contribute solutions or better solutions in shorter time.

In recent years, as new programming techniques and algorithms develop, a Multi-Agent Ant System for RAP was introduced by Bendjeghaba *et al.* in [35]. This algorithm concerns more on ant system.

More detailed literature surveys on this problem are presented in Kuo's review Refs. [11] and [36]

3.2 Invested Methods

Among the methods in the literature, we chose some representative approaches and addressed further investigation in order to learn some experience about how other algorithms solve the RAP.

3.2.1 The VNS algorithm

In the paper [2], Y.C Liang *et al.* proposed a *variable neighborhood search*(VNS) algorithm to solve the RAP. By "neighborhood", they mean the set of components allocation derived from an existing set under certain rules. Hence, these generated sets are "around" the original set which titles this algorithm with the name "VNS". They proposed 3 types of structures for constructing neighborhoods and a well defined penalty function to guide the search procedure. They use a so called shaking operator to add randomization to the algorithm. Thus the perturbation of the current solution provides the VNS a good opportunity to escape from the local optimum while a deterministic algorithm usually has no such luck on this issue. The following figure shows an example of type I method for building a neighborhood structure. In this structure type, based on the initial allocation

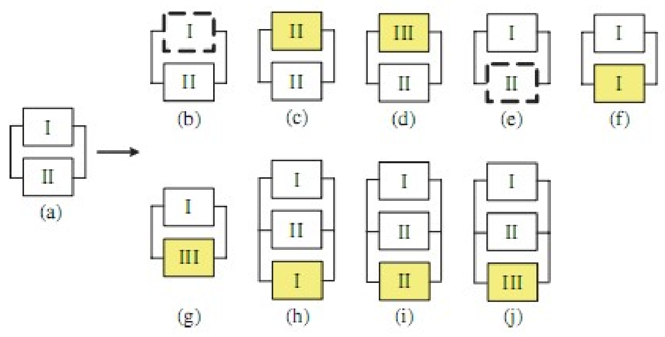


Figure 3.1: An example of neighborhood structure type I [2]

which is the set of one type I component and one type II component, nine different neighboring sets are generated by adding, deleting, or replacing one component in the initial set. The other 2 types of structure varies in the way that two or three components are exchanged simultaneously.

The procedure of the algorithm is as follows:

I. Construct a set of neighborhood structures N$_l$ according to the rules;

II. Generate an initial solution denoted by y;

Repeat the following steps until the maximum number of iterations is reached:

III. **Shaking:** Randomly generate a solution from the neighborhood;

IV. **Neighborhood search:** Find the best neighboring solution y' from the neighborhood with the penalty function applied;

V. If y' is better than y, y' $\rightarrow$ y, reset l=1 and go to **Step III**
else set $l = l + 1$ and go to **Step III**.

They use a "shaking" operator to add the randomization to the algorithm. Thus, with this perturbation, they provided VNS a good opportunity to escape from the local optimum while a deterministic algorithm usually has no such luck on this issue.

The penalty function designed in this algorithm is:

$$R_{up} = R_u \times \left(\frac{C}{C_u}\right)^{\gamma_c} \times \left(\frac{W}{W_u}\right)^{\gamma_w}$$

Where R_u denotes the unpenalized reliability with the corresponding cost and weight consumption, and γ_c, γ_w are derived from the feasible solution ratio in each round of iteration.

This penalty function is reasonable because the search procedure increases the magnitude of penalty heavily on infeasible solutions in order to move toward the border of the feasible region; Otherwise, encourage movement further into the infeasible region. Intuitively, the optimal solution usually happens approaching the infeasible region.

3.2.2 The dynamic programming approach

We also investigated the dynamic programming approach provided in [6]. Generally, since dynamic programming is more appropriate to deal with single constraints optimization problems, they interpreted the weight constraint into the objective function by using a *Lagrange multiplier*. Thus the problem is converted into the following form:

$$R_n = \left[\prod_{i=1}^{s} R_i(m_i, x_i)\right] \cdot e^{-\lambda \sum_{i=1}^{s} W_i(m_i, x_i)}$$

subject to the constraint:

$$\sum_{i=1}^{s} C_i(m_i, x_i) \leq C.$$

where λ is the Lagrange multiplier [37].

Then they carry out dynamic programming for this converted problem.

The advantages of this approach is that it generates results which associate with each Lagrange multiplier. So, in industry, when sub optimums or a big pool of alternative solutions are needed, this approach will reveal its superiority. The drawbacks of this approach are that it is very trivial to pick an appropriate Lagrange multiplier and the right range for cost constraint in each optimization step. There is no easy way to do this but to manually try lots of times. Since the real optimum only appears when the appropriate Lagrange multiplier is hit, the step of varying the Lagrange multiplier needs to be small enough to avoid missing the right one, which make this approach more computationally complex.

Chapter 4

Preliminary Investigation

In this chapter, we will present our analysis of the RAP, some comprehension on the approaches in the literature, and introduction of our approach.

4.1 Brief analysis

The search space for the RAP is huge, and it is difficult to implement traversing. As the problem scale goes big, it is impractical and impossible to traverse the solution space in finite time. We evaluate the time complexity for traversing the RAP by the following approximation.

To guarantee the conduction of the whole system, at least one component needs to be used in each subsystem. To satisfy this basic condition, the number of possible allocation is $\prod_{i=1}^{s} m_i$. Then the rest of the cost and weight can be allocated freely onto other components. Thus, we convert the approximation to compute the number of non-negative solutions of the following equation:

$$x_1 + x_2 + \cdots + x_k + \cdots + x_t = \bar{x}$$

where $t = \sum_{i=1}^{s} m_i$ denotes the total number of component types, and $\bar{x}$ denotes the average amount of components if only one type of identical components are put into the whole system. $\bar{x}$ is derived by $(x_{1max} + x_{2max} + \cdots x_{kmax} + \cdots + x_{tmax})/t$, and

$$x_{kmax} = \min\left\{\frac{C - \sum_{i=1,i\neq k}^{s} \min(c_{ij})}{c_k}, \frac{W - \sum_{i=1,i\neq k}^{s} \min(w_{ij})}{w_k}\right\}$$

As an example, for test problem 1 in table 4.1,

the total number of component types t is 48, and $\bar{x}$ is 40. We now compute the number

Table 4.1: Data for test problem 1.

Subsystem	components	components	components	components
i	r c w	r c w	r c w	r c w
1	0.90 1 3	0.93 1 4	0.91 2 2	0.95 2 5
2	0.95 2 8	0.94 1 10	0.93 1 9	
3	0.85 2 7	0.90 3 5	0.87 1 6	0.92 4 4
4	0.83 3 5	0.87 4 6	0.85 5 4	
5	0.94 2 4	0.93 2 3	0.95 3 5	
6	0.99 3 5	0.98 3 4	0.97 2 5	0.96 2 4
7	0.91 4 7	0.92 4 8	0.94 5 9	
8	0.81 3 4	0.90 5 7	0.91 6 6	
9	0.97 2 8	0.99 3 9	0.96 4 7	0.91 3 8
10	0.83 4 6	0.85 4 5	0.90 5 6	
11	0.94 3 5	0.95 4 6	0.96 5 6	
12	0.79 2 4	0.82 3 5	0.85 4 6	0.90 5 7
13	0.98 2 5	0.99 3 5	0.97 2 6	
14	0.90 4 6	0.92 4 7	0.95 5 6	0.99 6 9

of *non-negative* integer solutions of

$$x_1 + x_2 + x_3 + \cdots + x_{48} = 40$$

We use the technique in combinatorial mathematics, and treat the 40 component as "1" and the 48 component types as "$\star$". Then, we are actually computing how many situations can appear if we put all these ones and 47(48-1) stars in a row or say separate these ones and stars into 48 heaps. As an example, for the circumstance of one type 1 component, two type 3 components, etc:

$$1 \star \star 1\,1 \star 1 \star \star \star \star \star 1 \star \cdots\cdots \star 1\,1 \star \star$$

that is $\binom{87}{40}$. So magnitude of the total number of feasible solution is :

$$[\tbinom{87}{40} + \tbinom{86}{39} + \tbinom{85}{38} + \cdots + \tbinom{48}{1}] \times 3^8 \times 4^6$$

So, we estimate the search space to have the magnitude of 1.08×10^{33} by incomplete estimation.

In [14], it is estimated that the the search space size is larger than 7.6×10^{33}. For more explicit illustration about the complexity, refer to [5].

4.2 Initialization strategies

We referenced Coit's linearization in [1], and a linear approximation for RAP in [19], and decomposed the problem in the following procedure.

From the assumption "failure of any unit in the system is assumed to be an independent event", we can divide the objective function into:

$$\mathbf{max}\ \{R_1(r_{1j}, x_{1j}),\ R_2(r_{2j}, x_{2j}),\ \cdots,\ R_s(r_{sj}, x_{sj})\}$$

Since $R_i(r_{ij}, x_{ij}) = 1 - \prod_{j=1}^{m_i}(1 - r_{ij})^{x_{ij}}$, the objective function is transformed to:

$$\mathbf{min} \left\{ \prod_{j=1}^{m_1}(1 - r_{1j})^{x_{1j}},\ \prod_{j=1}^{m_2}(1 - r_{2j})^{x_{2j}},\ \cdots,\ \prod_{j=1}^{m_s}(1 - r_{sj})^{x_{sj}} \right\}$$

By taking natural log to each expression in the braces above, we further transform the problem to:

$$\mathbf{min}\left\{\sum_{j=1}^{m_1} x_{1j}\ln(1-r_{1j}), \sum_{j=1}^{m_2} x_{2j}\ln(1-r_{2j}), \cdots, \sum_{j=1}^{m_s} x_{sj}\ln(1-r_{sj})\right\}$$

with the same overall constraint functions.

If we divide the overall cost and weight constraints into s shares in a reasonable way, we can decompose the overall objective function into s part as well. Thus, for arbitrary ith subsystem, the objective function can be expressed in the form of:

$$\mathbf{min} \quad a_{i1}x_{i1} + a_{i2}x_{i2} + \cdots + a_{im_i}x_{im_i}$$

where $a_{ij} = \ln(1-r_{ij})$. The constraints functions are:

$$c_{i1}x_{i1} + c_{i2}x_{i2} + \cdots + c_{im_i}x_{im_i} \leq \widetilde{c}_i,$$

$$w_{i1}x_{i1} + w_{i2}x_{i2} + \cdots + w_{im_i}x_{im_i} \leq \widetilde{w}_i$$

Up to now, we have decomposed the big non-linear optimization problem into a set of small scale linear optimization problem. So, for each subsystem, if we assign a certain amount of cost and weight, the optimization can be performed by implementing mixed integer linear programming(MILP). It is this decomposition that enables us to implement multi-agent programming concept to solve this problem. Since there are different kinds of integer linear programming tools or softwares for LP problems available such as CPlex, Matlab LP function, etc. [38] [39], it is not a difficult job to solve the LP problem. It becomes our main concern to find out which is a proper way to assign rationally right cost and weight constraints.

In that the RAP is a two constraints optimization problem, we can not evaluate the cost-effectiveness of a component as what we do to the items in some other problems, for instance, the Knapsack Problems(evaluating the ratio of profit to weight) [40]. Hence, we tested the following strategies for initially allocating the cost and weight, in order to eliminate some unnecessary attempts and get a good start:

1. an intuitive thinking is to divide the cost and weight equally by 14 and assign them to the 14 subsystem;

2. computing $\frac{r_{ij}}{\frac{C}{C+W}c_{ij}+\frac{W}{C+W}w_{ij}}$ to express the cost-effectiveness of the components, meaning the reliability which lies on a weighed cost and weight constraints, and afterwards, pick the component with the highest cost-effectiveness in each subsystem;

3. computing $\left|\frac{c_{ij}}{w_ij}-\frac{C}{W}\right|$ and pick the components with lowest value which means they have a cost-weight pattern more coherent to the cost and weight constrains. The term of "weight tight" was introduced in the paper [21]. Here we explain "cost tight" and "weight tight" with our own comprehension. A problem is cost tight if the optimal solution consumes all the cost constrains with some weight quota not being used and vice versa.

Next, we adjust the cost and weight assignment for each subsystem and carry out the MILP for the subsystem iteratively. As a matter of fact, in the beginning, the bottleneck which restricts the overall performance of the whole system is the subsystem with the lowest reliability. The logic is simple, even if all of the rest 13 subsystems achieve a reliability approaching 100%, as long as a subsystem with a reliability of 0.95 exists, the overall reliability is determined right by that subsystem. Hence, we let the subsystem with higher reliability share some cost and weight quota as a compromise to the subsystem with lower reliability so as to get overall improvement. We do obtain significant improvement at the first few rounds of iterations. The following figure shows the rapidly rising in the beginning using this strategy.

However, two problems of this intuition are:

1. After some iterations, as the reliability of subsystems tend to balance, each round of iteration does not promise to provide improvement, which results in waste of computational resource and ineffectiveness of the algorithm;

2. for some specific subsystems like subsystem 7 and 13 in test problem 1, components are so "heavy" that if one more is used, it becomes a subsystem with highest reliability

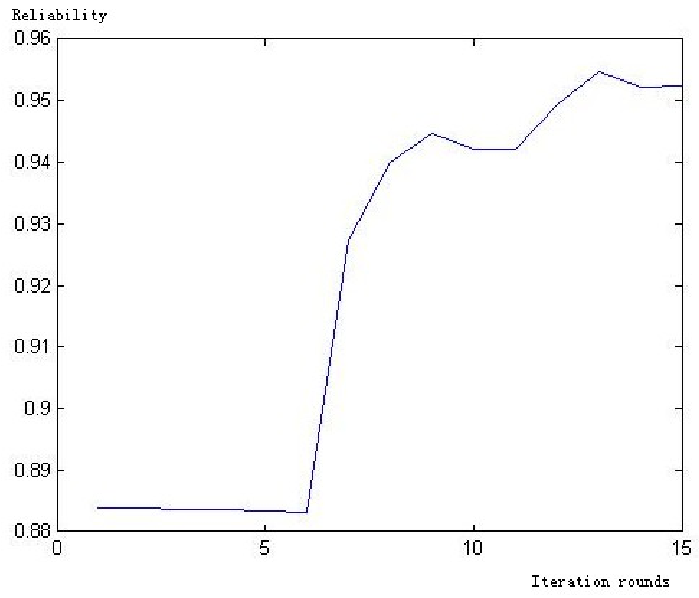


Figure 4.1: The improvement process of originally proposed approach

while vicely, if one component is not used, that subsystem becomes the one with lowest reliability. In this case, the iteration would goes into a dead loop and has no chance to jump out. The following figure is a schematic showing that the iteration stagnates at some local optimal:

The reason for these problems is that this proposed strategy is not mathematically proved to converge to the global optimal. The search procedure is able to get a good solution in some extent, however, it tends to stagnate at a level (usually not the optimal) and hard to clime up. To address this issue, we come up with the main part of our proposed multi-agent based approach and the detail will be discussed in Chapter 5.

4.3 Original proposed algorithm

Our originally proposed strategy can be organized into three stages: initialization, adjustment, and multi-agent trading. The previously stated strategy can be modulized as

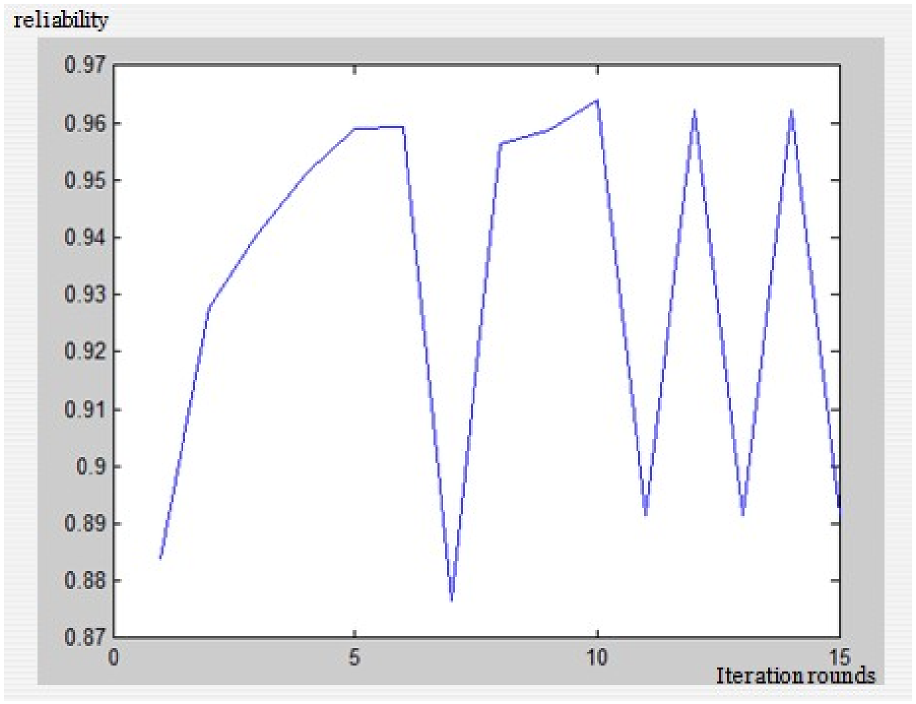


Figure 4.2: A schematic showing the search procedure stagnates at local optimum

an algorithm below:

INITIALIZATION

Compute the cost-effectiveness *ce* of each component .

Pick the component with highest *ce* for each subsystem as an initial solution and compute the total cost and weight consumed.

ADJUSTMENT:

Iterate the this section for a pre-fixed times.

Compute *Cleft* and *Wleft*;

Pick the subsystem with the highest reliability subRmax;

Compute *Cshare* and *Wshare* under the strategy;

IF the subsystem with subRmax falls to the lowest subR after adjusting,

pick the subsystem with the 2nd highest reliability;

Pick the subsystem with the lowest reliability subRmin;

Add *Cleft*+*Cshare* and *Wleft*+*Wshare* to its cost and weight constrains respectively.

Perform mixed integer LP for the changed two subsystems and get the optimal solution and allocation set for this subsystem;

Compute the overall reliability R;

IF R is improved

Update the record of R and optimal set.

MULTI-AGENT TRADING

Repeat the following steps until the preset number of iterations is reached

Create a "broker" agent and an agent for each subsystem, and compute the LP solution under a predefined range of cost and weight constraints around the result derived from last stagy;

Agents compute the trade of Cshare and Wshare, and sent the matching result to the "broker" agent;

The broker agent pick the best pair and perform the trade.

Record the overall reliability.

With this algorithm, the best result we obtained is 0.9630, which is not as good as that in the literature, shown in figure 4.1. We made big improvement in modified approach and this will be discussed in next chapter.

4.4 Test problems

Although Problem 1 is a well-known test problem, and it has been extensively used in the literature to evaluate alternative approaches to solve the RAP, it has some drawbacks as a test problem. For example, some of component options clearly dominate others, i.e., in subsystem 3, type III components have lower cost nd weight, but higher reliability than type I components. Moreover, as illustrated previously, it has a tight weight constraint. In fact, some component options can be eliminated if the problem data are carefully analyzed. This feature of Problem 1 favors meta-heuristic & heuristic approaches that

Table 4.2: Data for test problem 2,3,4. Provided in [1]

	Problem2			Problem3			Problem4		
i	$c_(i1)...c_(i4)$	$w_{i1}...w_(i4)$	$r_{i1}...r_(i4)$	$c_(i1)...c_(i4)$	$w_{i1}...w_(i4)$	$r_{i1}...r_(i4)$	$c_(i1)...c_(i4)$	$w_{i1}...w_(i4)$	$r_{i1}...r_(i4)$
1	3,5,8,10	3,5,8,10	.71,.82,.9,.99	3,5,6,10	9,7,4,1	.92,.9,.91,.92	3,4,7,10	4,3,10,7	.81.8.96.98
2	2,5,8,10	4,6,8,10	.7,.82,.92,.98	2,5,7,10	10,6,5,3	.92,.9,.92,.91	3,5,7,8	4,2,10,7	.82,.82,.97,.98
3	3,5,7,10	2,6,8,10	.7,.83,.91,.97	1,5,7,9	9,6,4,1	.91.93.93.92	1,5,6,9	4,1,10,7	.82,,.82,.97,.98
4	4,6,8,9	2,6,7,9	.72,.82,.93,.97	1,5,7,9	9,7,5,3	.93,.93,.92,.9	2,5,6,9	5,2,8,7	.83,.8,.96,.97
5	2,5,7,9	2,5,7,10	.73,.81,.92,.97	1,5,6,9	10,7,4,2	.93,.91,.9,.91	3,5,7,8	5,2,10,7	.82,.81,.96,.98
6	3,5,7,9	3,5,8,10	.71,.81,.92,.99	2,4,7,9	8,7,5,1	.91,.9,.9,.92	1,4,7,10	5,3,10,6	.8,.82,.98,.98
7	2,6,8,9	4,5,7,10	.73,.83,.92,.98	1,5,6,9	10,7,4,3	.93,.9,.9,.9	1,4,6,8	5,2,9,7	.81,.83,.98,.98
8	4,5,7,10	4,5,8,10	.73,.8,.91,.98	2,5,7,9	8,7,5,3	.92,.92,91,.92	3,4,6,9	4,1,19,7	.8,.83,98,98
9	4,6,7,9	4,6,8,10	.72,.82,.91,.99	1,4,6,8	8,7,5,2	.9,.91,.93,.9	2,4,6,10	5,3,9,6	.83,.82,.97,.96
10	3,6,8,10	2,6,7,9	.72,.83,.91,.99	1,5,7,9	10,7,5,1	.93,.91,.92,.91	2,4,6,9	4,3,9,7	.81,.81,.96,.98
11	4,5,7,9	2,5,7,10	.71,.83,.93,.97	1,4,6,9	10,6,4,2	.93,.93,.9,.91	1,5,6,8	5,1,10,6	.82,.81,.97,.98
12	2,5,8,9	3,6,8,9	.71,.81,.91,.97	1,5,7,8	9,6,4,1	.9,.93,.93,.9	2,5,6,8	5,3,9,6	.83,.83,.98,97
13	2,6,8,9	2,5,7,10	.72,.83,.91,.98	1,4,6,10	9,6,4,1	.93,.9,.91,.92	1,5,6,8	4,3,8,6	.81,.81,.96,.98
14	2,6,7,10	4,6,8,10	.73,.83,.9,.98	2,5,6,9	10,6,5,1	.93,.92,.91,.91	3,5,6,9	5,3,8,6	.81,.8,.97,.96
15	3,6,8,10	2,5,8,10	.73,.83,.93,.98	2,4,6,9	8,7,5,3	.93,.93,.91,.9	2,4,6,10	5,1,8,7	.8,.82,.98,.98
16	4,6,7,10	4,5,8,10	.71,.83,.92,.98	2,4,6,10	9,6,4,3	.91,.9,.93,.91	1,5,6,8	4,2,9,7	.82,.82,.98,.98
17	3,5,7,10	2,6,7,9	.7,.8,.92,.97	2,5,7,9	10,6,4,3	.9,.9,.92,.92	1,4,7,8	5,2,10,7	.8,.83,.97,.96
18	2,6,7,10	3,5,8,9	.72,.8,.93,.	1,5,6,9	8,7,5,2	.91,.92,.9,.91	3,5,6,10	4,2,9,7	.8,.8,.97,.96
19	2,6,8,9	4,5,8,10	.71,.8,.93,.97	3,5,7,8	8,6,4,3	.92,.9,.9,.93	2,4,7,9	4,2,8,6	.81,.83,.98,.98
20	4,6,8,10	3,5,8,9	.7,.83,.9,.99	1,5,6,8	9,7,5,2	.91,.91,.9,.93	1,4,6,9	5,3,9,6	.8,.8,.98,.98

depend on local perturbation operators because difficult tradeoffs do not exist among the component options.

Thus, Nakagawa *et al.* [34] extended the test problem to vary the weight constraint form 159 to 191. Coit gives other 3 sets of components as refined test problems. As shown in table 4.2, each problem has 20 subsystems with four available component options for each subsystem. The maximum number of components allowed in each subsystem is 8. These problems have an even larger solution space with possible configurations, which is more qualified to be a challenge for newly developed algorithms.

Chapter 5

Proposed Method

In the problem analysis section, the linearization and decomposition of computing the reliability of subsystems make it possible to perform optimization autonomously in each subsystem, which furthermore makes it possible to implement multi-agent programming to the RAP. Logically, we can actually carry out parallel computing on different computing devices simultaneously. For simplicity, we will use test problem 1 as our example throughout the chapter. As proposed previously, besides assigning an agent for each subsystem, in addition, we have a *centralized broker agent* (CBA) to inform all agents how trading should be performed. Here, we consider all agents to be honest and delivery all authenticate messages to other agents. Once a decision is made by the CBA, all agents are conceded with the decision. For the trading procedure, there are three steps:

1. Determine reliability, cost and weight data sets,
2. Trade among subsystems, and
3. Winner determination

5.1 Determine reliability, cost and weight data sets

As we know, we have m_i types of components in the i-th subsystem. In this chapter, we express their parameters in the form of set Ψ_i:

$$\Psi_i = \{(r_{i1}, c_{i1}, w_{i1}), (r_{i2}, c_{i2}, w_{i2}), \ldots, (r_{im_i}, c_{im_i}, w_{im_i})\},$$

where r_{ij}, c_{ij} and w_{ij} are the components' reliability, cost and weight respectively, for $j = 1, 2, \ldots, m_i$ and $i = 1, 2, \ldots, s$. For example, we have Ψ_1 for the first subsystem and

Ψ_2 for the second subsystem in test problem 1 as

$$\begin{aligned}\Psi_1 &= \{(0.93,1,3),(0.93,1,4),(0.91,2,2),(0.95,2,5)\},\\ \Psi_2 &= \{(0.95,2,8),(0.94,1,10),(0.93,1,9)\}.\end{aligned}$$

From component parameter sets, we now define three parameter extracting functions $\mathcal{R}s$, $\mathcal{C}s$ and $\mathcal{W}s$ as

$$\begin{aligned}\mathcal{R}s(\Psi_i) &= \{r_{i1}, r_{i2}, \ldots r_{im_i}\},\\ \mathcal{C}s(\Psi_i) &= \{c_{i1}, c_{i2}, \ldots c_{im_i}\}, \text{ and}\\ \mathcal{W}s(\Psi_i) &= \{w_{i1}, w_{i2}, \ldots w_{im_i}\}.\end{aligned}$$

Continued from the previous example, we have

$$\begin{aligned}\mathcal{R}s(\Psi_1) &= \{0.90, 0.93, 0.91, 0.95\},\\ \mathcal{C}s(\Psi_1) &= \{1,1,2,2\}, \text{ and}\\ \mathcal{W}s(\Psi_1) &= \{3,4,2,5\}.\end{aligned}$$

From these parameter sets, we can find each subsystem's permissible performance set Γ_i, and it can be determined by solving

$$\mathbf{min}\left\{\sum_{j=1}^{m_i} x_{ij}\ln(1-r_{ij})\right\} \tag{5.1}$$

subject to

$$\sum_{j=1}^{m_i} x_{ij}c_{ij} \le \widetilde{c}_i \ \ \textit{and} \tag{5.2}$$
$$\sum_{j=1}^{m_i} x_{ij}w_{ij} \le \widetilde{w}_i, \tag{5.3}$$

where

$$\widetilde{c}_i = \min\left(\mathcal{C}(\Psi_i)\right), \min\left(\mathcal{C}(\Psi_i)\right)+1, \ldots, C - \sum_{\substack{l=1\\ l\neq i}}^{s} \min\left(\mathcal{C}(\Psi_l)\right)$$

and

$$\widetilde{w}_i = \min\left(\mathcal{W}(\Psi_i)\right), \min\left(\mathcal{W}(\Psi_i)\right) + 1, \ldots, W - \sum_{\substack{l=1 \\ l \neq i}}^{s} \min\left(\mathcal{W}(\Psi_l)\right)$$

As we can see, by varying $\widetilde{c}_i$ and $\widetilde{w}_i$, there are $(C - \sum_{i=1}^{s} \min\left(\mathcal{C}(\Psi_i)\right)) \times (W - \sum_{i=1}^{s} \min\left(\mathcal{W}(\Psi_i)\right))$ linear integer programming problems that we need to solve to get all possible x_{ij} for each subsystem, and the permissible performance set Γ_i can be denoted as,

$$\Gamma_i = \{(r_1^{(i)}, c_1^{(i)}, w_1^{(i)}, \mathbf{X}_1^{(i)}), \ldots, (r_l^{(i)}, c_l^{(i)}, w_l^{(i)}, \mathbf{X}_l^{(i)}), \ldots, (r_{L_i}^{(i)}, c_{L_i}^{(i)}, w_{L_i}^{(i)}, \mathbf{X}_{L_i}^{(i)})\},$$

where $l = 1, 2, \ldots, L_i$, and

$$\begin{aligned} r_l^{(i)} &= 1 - e^{\left\{\sum_{j=1}^{m_i} \mathbf{X}_l^{(i)}(j) \ln(1 - r_{ij})\right\}} \\ &= 1 - \prod_{j=1}^{m_i} (1 - r_{ij})^{\mathbf{X}_l^{(i)}(j)}, \\ c_l^{(i)} &= \sum_{j=1}^{m_i} c_{ij} \mathbf{X}_l^{(i)}(j), \text{ and} \\ w_l^{(i)} &= \sum_{j=1}^{m_i} w_{ij} \mathbf{X}_l^{(i)}(j), \end{aligned}$$

$r_l^{(i)}$ is actually the reliability of subsystem i when the l-th performance set is used. $\mathbf{X}_l^{(i)}$ is a vector of the component assignment $(x_{i1}, x_{i2}, ..., x_{im_i})_l$.

For any $1 \le l_1 < l_2 \le L_i$, $0 < r_{l_1}^{(i)} \le r_{l_2}^{(i)} \le 1$. In other words, the set Γ_i is sorted according to the ascending order of reliability. The cost and weight in the data set are not exactly placed in the ascending order likewise, however, they satisfy the following fact

$$c_l^{(i)} \le c_k^{(i)} \text{ and } w_l^{(i)} \le w_k^{(i)} \;\Rightarrow\; r_l^{(i)} \le r_k^{(i)}$$

because logically, we can find a more reliable or at least equivalently reliable assignment if higher cost and weight constraints are given.

From equation 5.1 to 5.3, we can determine cost and weight ranges for subsystem 1 as $\widetilde{c}_1 = 1, 2, \ldots, 97$ and $\widetilde{w}_1 = 2, 3, \ldots, 104$. We then solve all

$$\mathbf{min}\left\{x_{11} \ln(1 - 0.9) + x_{12} \ln(1 - 0.93) + x_{13} \ln(1 - 0.91) + x_{14} \ln(1 - 0.95)\right\}$$

for each $\widetilde{c}_1$ and $\widetilde{c}_1$, subject to

$$\begin{aligned} x_{11} + x_{12} + 2x_{13} + 2x_{14} &\leq \widetilde{c}_1, \text{ and} \\ 3x_{11} + 4x_{12} + 2x_{13} + 5x_{14} &\leq \widetilde{w}_1, \end{aligned}$$

By varying cost and weight constraints $(\widetilde{c}_1, \widetilde{w}_1)$ $97 \times (104 - 1) = 9,991$ times, (*e.g.* $(1,2), (1,3), \ldots, (1,104), (2,2), (2,3), \ldots, (97,104)$), and after eliminating identical results, there are $3,720$ uniquely optimal allocations based on different cost and weight constraints in subsystem 1, expressed as

$$\begin{aligned} \Gamma_1 &= \{(0.9, 1, 3, (1,0,0,0)), (0.91, 2, 2, (0,1,0,0), (0.93, 1, 4, (0,0,1,0)), \\ &\quad (0.95, 2, 5, (0,0,0,1)), \ldots, (\simeq 1, 96, 102, (4,0,30,0))\}. \end{aligned} \tag{5.4}$$

Similar, we get 428 optimal allocation for subsystem 2 as

$$\begin{aligned} \Gamma_2 &= \{(0.93, 1, 9, (1,0,0)), (0.94, 1, 10, (0,1,0)), (0.95, 2, 8, (0,0,1)), \\ &\quad (0.9951, 2, 18, (0,0,2)), \ldots, (\simeq 1, 26, 104, (13,0,0))\}. \end{aligned}$$

A complete list of possible permissible data set for subsystems I and II can be found in Appendix B. The data set of subsystems 3$\sim$14 optimal allocation can be obtained using similar approach.

5.2 Trade among subsystems

Many MAS systems are implemented in computer simulations, stepping the system through discrete "time steps". There is a typical introduction about multi-agent system on "wikipedia" gives the following example of agent behavior paradigms:

> The MAS components communicate typically using a weighted request matrix, e.g. A challenge-response-contract scheme is common in MAS systems, where First a "Who can?" question is distributed. Only the relevant agents respond: "I can, at this price". Finally, a contract is set up, usually in several more short communication steps between agents.

According to this paradigm, we incorporate trading procedure in the agents' behavior.

Suppose each subsystem has chosen a performance set $\gamma_i(l)$ among its own Γ_i where

$$\gamma_i(l) = \left(r_l^{(i)}, c_l^{(i)}, w_l^{(i)}, \mathbf{X}_l^{(i)}\right),$$

In fact, $\gamma_i(l)$ can be considered as the l-th element in Γ_i. Thus, CBA can compute system reliability R, used cost C_u and weight W_u, remained cost C_r and weight W_r in the following way

$$\begin{aligned} R &= \prod_{i=1}^{s} r_{l_i}^{(i)} \\ C_u &= \sum_{i=1}^{s} c_{l_i}^{(i)} \\ W_u &= \sum_{i=1}^{s} w_{l_i}^{(i)} \\ C_r &= C - C_u, \text{ and} \\ W_r &= W - W_u. \end{aligned}$$

The i-th agent broadcasts its performance set above its current determination l_i(including $\gamma_i(l_i)$) $\Gamma_i^{l_i} = \{\gamma_i(1), \gamma_i(2), \ldots, \gamma_i(l_i)\}$ to other subsystems. As an example, if the first and second agents choose their performance indexes $l_1 = 7$ and $l_2 = 7$ for the subsystem they represent, the broadcast messages are

$$\begin{aligned} \Gamma_1^7 = &\{(0.9, 1, 3, 1), (0.91, 2, 2, 1), (0.93, 1, 4, 1) \\ &(0.95, 2, 5, 1), (0.99, 2, 6, 2), (0.991, 3, 5, 2) \\ &(0.9919, 4, 4, 2)\}, \text{ and} \\ \Gamma_2^7 = &\{(0.93, 1, 9, 1), (0.94, 1, 10, 1), (0.95, 2, 8, 1) \\ &(0.9951, 2, 18, 2), (0.9958, 2, 19, 2), (0.9964, 2, 20, 2) \\ &(0.9965, 3, 17, 2)\}. \end{aligned}$$

Without losing generality, assume that all other agents choose their performance indexes $l_i = 7$ as well, for $i = 3, 4, \ldots s$ and $s = 14$ in test problem 1. The CBA can determine the initial solution by

$$\begin{aligned}
R &= \prod_{i=1}^{s} r_{l_i}^{(i)} = 0.9919 \times 0.9965 \ldots \times 0.9950 = 0.90265118 \\
C_u &= \sum_{i=1}^{s} c_{l_i}^{(i)} = 98 \\
W_u &= \sum_{i=1}^{s} w_{l_i}^{(i)} = 169 \\
C_r &= C - C_u = 130 - 98 = 32, \ \textit{and} \\
W_r &= W - W_u = 170 - 169 = 1.
\end{aligned}$$

Next, every agent enters the proposing trading procedure. We say the trading has direction because from each agent's point of view, the "interest", or say "desire" is to make self improvement, which means that it only cares about how much cost and/or weight it can obtain from other agents to increase its own reliability. Thus, it only takes into consideration other agents' permissible performance sets above their current position while looking up its own set below its current position. For example, agent j uses the received messages to determine whether it would trade with another agent i or not by evaluating different performance set indexes q and p according to:

$$\mathbf{max}\left\{r_p^{(i)} \times r_q^{(j)}\right\}$$

subject to

$$\begin{aligned}
c_p^{(i)} + c_q^{(j)} &\leq c_{l_i}^{(i)} + c_{l_j}^{(j)} + C_r, \\
w_p^{(i)} + w_q^{(j)} &\leq w_{l_i}^{(i)} + w_{l_j}^{(j)} + W_r, \\
p < l_i, \quad &\text{and} \quad q > l_j.
\end{aligned}$$

We define the trading function to be:
For arbitrary agent j,

$$\delta(\gamma_j(q), \gamma_i(p)) = \begin{cases} 1 & \text{if } r_p^{(i)} \times r_q^{(j)} \geq r_{l_i}^i \times r_{l_j}^j, \\ & \quad c_p^{(i)} + c_q^{(j)} \leq c_{l_i}^i + c_{l_j}^j + C_r, and \\ & \quad w_p^{(i)} + w_q^{(j)} \leq w_{l_i}^i + w_{l_j}^j + W_r \\ 0 & \text{otherwise} \end{cases}$$

This trading function works as a comparator between a proposed trading with the current components assignment. We evaluate the proposed trading $(\gamma_j(q), \gamma_i(p))$ to be feasible if it makes improvement, which means it provides higher reliability than current assignment $(\gamma_{l_i}(i), \gamma_{l_j}(j))$ while the sum of cost and weight not exceeding the sum of current cost and weight plus the remained cost and weight quota. If there is improvement, then the attempt is feasible; otherwise, the attempt is infeasible. The input of this function is an attempt of trading $(\gamma_j(q), \gamma_i(p))$ proposed by agent j with agent i, and the output of this function is the feasibility of the attempt, which is 0 or 1.

For example, if agent 2 receives a message Γ_1^7 from agent 1, we can compute from performance data set 5.4 that

$$\begin{aligned} c_{l_2}^{(2)} + c_{l_1}^{(1)} + C_r &= 4 + 3 + 32 = 39, \\ w_{l_2}^{(2)} + w_{l_1}^{(1)} + W_r &= 4 + 17 + 1 = 22. \end{aligned}$$

Under these constraints, one can apply the trading function to check the feasibility of a proposed trading. Table 5.1 shows the results for the example. In this table, the involved agents are $i = 1$ and $j = 2$, representing the subsystem respectively. The current position of agent 1 is $l_1 = 7$, and that of agent 2 is $l_2 = 7$. From the results of feasibility, we can see that $(\gamma_9(2), \gamma_6(1))$ is a feasible proposed trading. If multiple feasible trading proposals exist for some agents, they select the trading which yields the highest reliability among all the feasible ones.

Table 5.1: The feasibility table of trading between agent1 and agent2 proposed by agent 1

attempts of trading	reliability	sum of cost	sum of weight	feasibility	
current	0.988428	7	21	-	
$(\gamma_6(1), \gamma_8(2))$	0.988027	6	23	0	
$(\gamma_5(1), \gamma_8(2))$	0.987030	5	24	0	
	...	...	...		
$(\gamma_6(1), \gamma_9(2))$	0.9885225	7	21	1	$\leftarrow$
$(\gamma_5(1), \gamma_9(2))$	0.9875250	6	22	0	
	...	...	...		
$(\gamma_6(1), \gamma_{10}(2))$	0.9906600	6	32	0	
	...	...	...		

Explicitly, One can find that when $p = 6$ and $q = 9$, it will result a maximum reliability.

$$\begin{aligned} r_p^{(i)} \times r_q^{(j)} &= 0.991 \times 0.9975 \\ &= 0.9885225 > r_{l_i}^{(i)} \times r_{l_j}^{(j)}, \\ c_p^{(i)} + c_q^{(j)} &= 3 + 4 = 7 < 39, \\ w_p^{(i)} + w_q^{(j)} &= 5 + 16 = 21 < 22. \end{aligned}$$

We record the feasible proposed trading in the message form:

$$\phi(\gamma_q(j), \gamma_p(i)) = (r_p^{(i)} \times r_q^{(j)}, c_p^{(i)} + c_q^{(j)}, w_p^{(i)} + w_q^{(j)}).$$

Note: the order in the brackets indicates roles in this trading, which means that trading is proposed by the former agent and latter one is passive.

The checking feasibility procedure completes when both sum of cost and sum of weight exceed the current sum of cost and sum of weight plus the remaind cost and weight quota, which means no more trading is possibly feasible to make improvement even if the attempts continue.

It is evident that if agent 2 performs trading with agent 1, the overall system will benefit with a higher reliability. However, to be more efficient, we design to perform the trading after evaluating all $s-1$ messages received from all other agents in order to decide which potential trading gives highest reliability. That means when agent 2 derives the following feasible proposed trading,

$$\begin{aligned}
\phi(\gamma_9(2), \gamma_6(1)) &= (0.988522, 7, 21), & \phi(\gamma_9(2), \gamma_6(3)) &= (0.98710, 9, 26),\\
\phi(\gamma_9(2), \gamma_6(4)) &= (0.995006, 14, 24), & \phi(\gamma_9(2), \gamma_6(5)) &= (0.993909, 8, 24),\\
\phi(\gamma_9(2), \gamma_6(6)) &= (0.996303, 8, 25), & \phi(\gamma_9(2), \gamma_6(9)) &= (0.996602, 8, 32),\\
\phi(\gamma_9(2), \gamma_6(11)) &= (0.995106, 12, 27), & \phi(\gamma_9(2), \gamma_6(13)) &= (0.997492, 10, 31).
\end{aligned}$$

it selects the one that produces the highest reliability, which is $\phi^*(\gamma_9(2), \gamma_6(13))$ here, and sends the message to CBA. If an agent i has no feasible proposed trading, which means that all the evaluated trading result 0 output for δ function, then it send out a failure message $\phi^* = 0$.

5.3 Winner determination

After autonomous computation, any agent who has feasible proposed trading will transmit a messages to CBA, for example, agent i will transmit $\phi^*(\gamma_p(i), \gamma_q(j))$. We call this agent behavior as *bidding*. After receiving all the bidding information from agents, CBA will then decide a trading as the winner of bidding by picking the one which produces highest overall system reliability:

$$\Phi^*(\gamma_p(i), \gamma_q(j)) = \phi^*(\gamma_p(i), \gamma_q(j))$$

$$\begin{aligned}
where \qquad & \arg\max_{i,j} \left\{ \mathcal{R}s(\phi^*(\gamma_p(i), \gamma_q(j)) \times \prod_{k=1, k\neq i,j}^{s} r_{l_k}^{(k)} \right\}\\
= \quad & \arg\max_{i,j} \left\{ r_p^{(i)} \times r_q^{(j)} \times \prod_{k=1, k\neq i,j}^{s} r_{l_k}^{(k)} \right\}
\end{aligned}$$

Note: the CBA can distinguish the sender of message by checking the first field in $\phi^* = (\gamma_p(i), \gamma_q(j))$.

For test problem 1, the CBA will receive the messages from all the agents, and evaluate the improvement to the whole system respectively in the following way:

$$\begin{aligned}
\phi^*(\gamma_{28}(1), \gamma_5(13))\ max(R) &= 0.9099, & \phi^*(\gamma_9(2), \gamma_6(13))\ max(R) &= 0.9036, \\
\phi^*(\gamma_{18}(3), \gamma_5(13))\ max(R) &= 0.91121, & \phi^*(\gamma_{26}(4), \gamma_5(13))\ max(R) &= 0.92249, \\
\phi^*(\gamma_{17}(5), \gamma_5(13))\ max(R) &= 0.90572, & \phi^*(\gamma_{18}(6), \gamma_5(13))\ max(R) &= 0.90338, \\
\phi^*(\gamma_{18}(7), \gamma_5(13))\ max(R) &= 0.9068, & \phi^*(\gamma_{18}(8), \gamma_5(13)) max(R) &= 0.90761, \\
\phi^*(\gamma_{18}(9), \gamma_5(13))\ max(R) &= 0.90287, & \phi^*(\gamma_{18}(10), \gamma_5(13))\ max(R) &= 0.90429, \\
\phi^*(\gamma_{18}(11), \gamma_5(13))\ max(R) &= 0.90429, & \phi^*(\gamma_{18}(12), \gamma_5(13))\ &\mathbf{max(R)=0.92781}, \\
\phi^* = 0(failure\ message) & & \phi^*(\gamma_{18}(14), \gamma_5(13))\ max(R) &= 0.90665.
\end{aligned}$$

The CBA will decide $\Phi^*(\gamma_{18}(12), \gamma_5(13)) = \phi^*(\gamma_{18}(12), \gamma_5(13))$ in this circumstance as the winner among bidding messages received from all agents.

To summarize the logic of trading rules in short, each agent picks the best feasible trading ϕ^* autonomously among all the feasible ones ϕ between itself and all other agents, and reports to CBA in the form of a message ϕ^*. The CBA selects Φ^* among all the received bidding messages by picking the one that yields the highest improvement and perform the actual trading. After trading, the involved two agents update their current performance sets. Since all other subsystems remain unchanged, the overall system reliability R can be updated by

$$R = \left(\prod_{\substack{k=1 \\ k \neq i, k \neq j}}^{s} r_{l_k}^{(k)} \right) r_p^{(i)} r_q^{(j)},$$

Chapter 6

Simulation Results and Discussions

In this chapter, we will discuss the results obtained by the multi-agent based approach. Currently, the best result we get for test problem 1 is $R = 0.97076$, which equals the best result shown in the literature.

The computation is performed in Matlab and the agent framework used is Jade [41]. Compared with the results in the literature, this result is satisfying and encouraging. Figure 6.1 and 6.2 shows the iteration process:

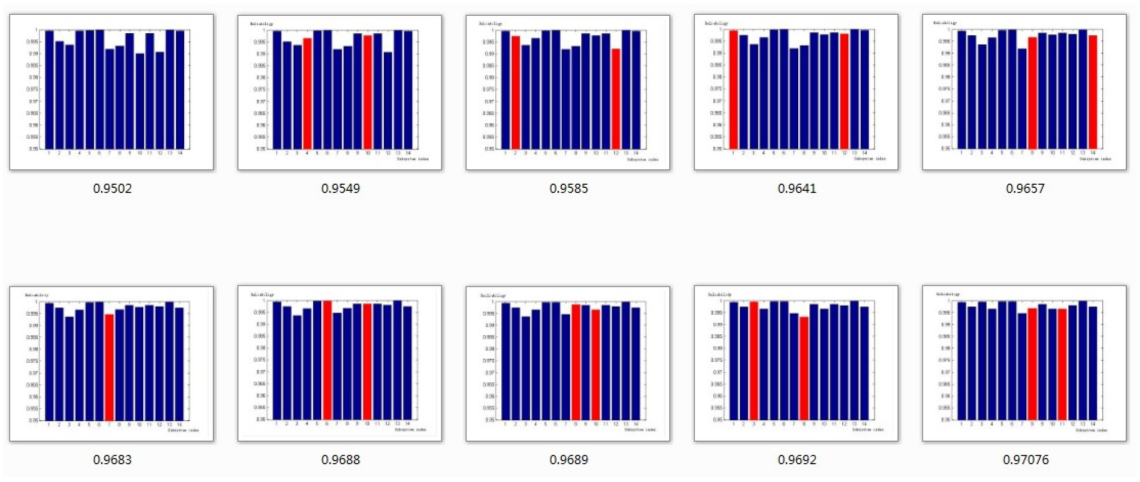


Figure 6.1: The schematic of iteration process 1

From these two figures, we can see explicitly how trading takes place between two agents in each round. To show the bit picture, we plot the procedure in the 3D bar figure, the X-axis is the subsystem index, Y-axis is the reliability respectively, and the Z-axis is the iteration rounds.

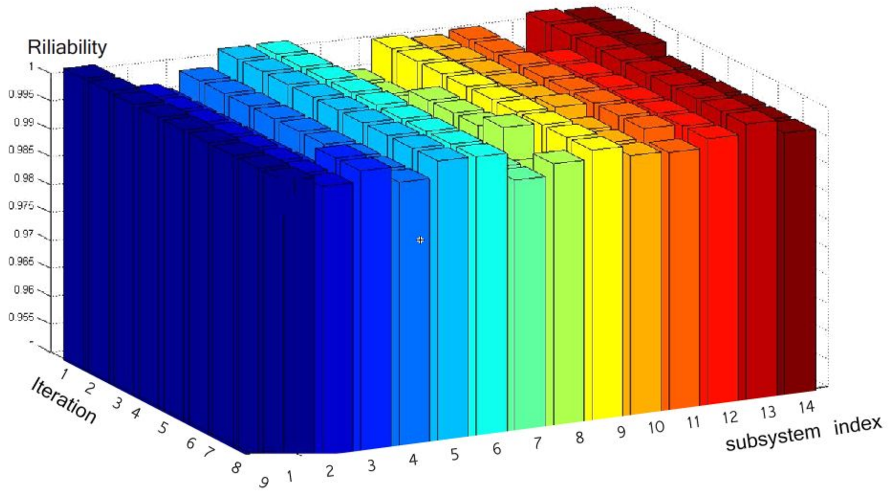


Figure 6.2: The schematic of iteration process 2

For reference, table 6.1 is the results obtained by different algorithms in the literature.

Table 6.1: Results in the literature for test problem 1 [2]

W	VND	GA	LP	ACO	TS	VNS_RAP
191	0.98506	0.98675	0.98671	0.98675	0.98681	0.98681
190	0.98358	0.98603	0.98632	0.98591	0.98642	0.98642
189	0.98348	0.98556	0.98572	0.98577	0.98592	0.98592
188	0.98302	0.98503	0.98503	0.98533	0.98538	0.98487
187	0.98131	0.98429	0.98415	0.98469	0.98469	0.98467
186	0.98082	0.98362	0.98388	0.98380	0.98418	0.98418
185	0.98034	0.98311	0.98339	0.98351	0.98351	0.98351
184	0.98046	0.98239	0.98220	0.98299	0.98299	0.98299
183	0.97919	0.98190	0.98147	0.98221	0.98226	0.98226
182	0.97940	0.98102	0.97969	0.98147	0.98152	0.98147
181	0.97850	0.98006	0.97928	0.98068	0.98103	0.98103
180	0.97732	0.97942	0.97833	0.98029	0.98029	0.98029
179	0.97670	0.97906	0.97806	0.97951	0.97951	0.97951
178	0.97600	0.97810	0.97688	0.97840	0.97840	0.97838
177	0.97540	0.97715	0.97540	0.97760	0.97747	0.97760
176	0.97379	0.97642	0.97498	0.97649	0.97669	0.97669
175	0.97389	0.97552	0.97350	0.97571	0.97571	0.97571
174	0.97275	0.97435	0.97233	0.97493	0.97479	0.97493
173	0.97039	0.97362	0.97053	0.97383	0.97383	0.97381
172	0.96945	0.97266	0.96923	0.97303	0.97303	0.97303
171	0.96872	0.97186	0.96790	0.97193	0.97193	0.97193
170	**0.96770**	**0.97076**	**0.96678**	**0.97076**	**0.97076**	**0.97076**
169	0.96495	0.96922	0.96561	0.96929	0.96929	0.96929
168	0.96374	0.96813	0.96415	0.96813	0.96813	0.96813
167	0.96302	0.96634	0.96299	0.96634	0.96634	0.96634
166	0.96200	0.96504	0.96121	0.96504	0.96504	0.96504
165	0.96128	0.96371	0.95992	0.96371	0.96371	0.96371
164	0.96051	0.96242	0.95860	0.96242	0.96242	0.96242
163	0.95942	0.96064	0.95732	0.96064	0.95998	0.96064
162	0.95515	0.95912	0.95555	0.95919	0.95821	0.95919
161	0.95682	0.95804	0.95410	0.95804	0.95692	0.95804

6.1 Comparison

We implemented the approach on test problems 2, 3, and 4, and got excellent performance on testproblem2, satisfying results on testproblem3 and to some extend good results on testproblem4. The data for test problems 2,3, and 4 were presented in Preliminary Investigation Chapter. Several problems were encountered and we will discuss this after the comparison. We compared our proposed method with Weighted Objectives Heuristic approach [1]. The results are shown in the following table. For short, we label our proposed approach with the name MABA.

Table 6.2: Comparison of results with Multiple Weighted Objectives Heuristic approach for test problems 2,3,4

			Problem 2		Problem 3		Problem 4	
i	Cost	Weight	MWO	MABA	MWO	MABA	MWO	MABA
1	100	100	0.05268	0.0478	0.17265	failed	0.14722	0.1424
2	100	130	**0.07702**	**0.0838**	0.25035	0.1871	0.32840	0.3267
3	100	160	**0.07702**	**0.0838**	0.31659	0.2833	0.55503	0.5577
4	100	190	**0.07702**	**0.0838**	0.39429	0.3635	0.64773	0.6375
5	100	220	**0.07702**	**0.0838**	**0.48209**	**0.4944**	0.72987	0.7335
6	100	250	**0.07702**	**0.0838**	**0.58655**	**0.5869**	0.76426	0.7750
7	130	100	0.08091	0.0650	0.24129	0.1600	0.31355	0.2060
8	130	130	**0.26884**	**0.2707**	0.31593	0.3004	0.56078	0.5563
9	130	160	**0.31506**	**0.3317**	0.39476	0.3526	0.66594	0.6582
10	130	190	**0.31061**	**0.3352**	0.48277	0.4696	0.74961	0.7499
11	130	220	**0.31470**	**0.3357**	0.59041	0.5821	0.83421	0.8358
12	130	250	**0.31061**	**0.3357**	0.71971	0.7182	0.89123	0.8921 ∗
13	160	100	0.08091	0.08091	0.30389	failed	0.53033	0.4696
14	160	130	0.28595	0.28595	0.38961	0.3806	0.64802	0.6474
15	160	160	**0.52941**	**0.55183**	0.48297	0.4549	0.74884	0.7470

Table 6.2 – continued from previous page

16	160	190	0.65762	0.6576	**0.59047**	**0.5955**	0.82922	0.8318
17	160	220	0.65762	0.6576	0.75473	0.7282	0.89751	0.8975
18	160	250	0.65762	0.6576	0.88177	0.8417	0.92416	0.9228
i	Cost	Weight	MWO	MABA	MWO	MABA	MWO	MABA
19	190	100	0.08091	0.08091	0.38067	0.3591	0.60364	0.5313
20	190	130	0.28595	0.2827	0.48074	0.4541	0.72425	0.7167
21	190	160	**0.60371**	**0.6159**	0.61995	0.6196	0.82697	0.8239
22	190	190	**0.75847**	**0.7616**	**0.75809**	**0.7657**	0.89718	0.8972
23	190	220	**0.79847**	**0.8205**	**0.88293**	**0.8881**	0.89718	0.9247
24	190	250	**0.80097**	**0.8129**	**0.90783**	**0.9083**	0.94725	0.9485
25	220	100	0.08091	0.08091	0.46579	0.4407	0.63909	0.5894
26	220	130	0.28595	0.2827	0.58585	0.5772	0.79882	0.7951
27	220	160	**0.60371**	**0.6081**	0.72885	0.7260	0.88788	0.8755
28	220	190	**0.79824**	**0.8081**	0.88214	0.8797	0.92239	0.9174
29	220	220	0.87281	0.8702	**0.90616**	**0.9148**	0.94678	0.9412∗
30	220	250	**0.89454**	**0.90005**	**0.92302**	**0.9327**	0.96779	0.9651
31	250	100	0.08091	0.08091	**0.54857**	**0.5704**	0.63909	0.5607
32	250	130	0.28595	0.28594	**0.71172**	**0.7222**	0.84744	0.8374
33	250	160	**0.60371**	**0.60806**	0.87444	0.8332	0.91345	0.9118
34	250	190	**0.79824**	**0.80807**	**0.90171**	**0.9047**	0.94285	0.9338
35	250	220	**0.88581**	**0.89061**	**0.92388**	**0.9240**	0.96401	0.9616
36	250	250	0.92957	0.92800	**0.94080**	**0.9461**	0.98031	0.9775

∗: multiple equivalent results were founded for this case.

As shown in table 6.2, and figure 6.3, we got better results in 21 out of 36 cases for test problem 2, with 8 other cases equivalent, which is very encouraging results. 13 out of 36 cases are better for test problem 3. The results are not so outstanding for test problem 4, but still, exceed in some cases.

More explicit results which include the final configuration of each subsystem (combination of components) are available in web page $http : //thoth.eng.temple.edu/?p = 392$

6.2 Failure attempts and improvements

As mentioned previously, the original program does not run perfectly under some cases. For example, if the initially picked solution uses more cost or weight (C_u, W_u) quota than the constraints, the remained cost and become negative, which results in problems to our program as an input. To fix this error, we added an initial solution adjusting function with randomization. Thus, the previously discussed strategies on how to pick cost-effective components as initial solution reveals its usefulness.

Thus, it makes sure that the used cost and weight of initial solution do not exceed the constraints directly. On the other hand, if one round of simulation fails to provide satisfying solution, it will begin from a new initial solution which could avoid stagnates at local optimization.

As a note of our approach, when the overall cost and weight constrains are very loose, for example, (250, 250). the remained cost weight (C_r, W_r) will be large at the first round. For instance, when the initial pick is 1 for every agent, the used cost C_u may be little, say 100, thus, C_r would be 150. In this case, each agent would look up a long list under its current position to check whether potential trading exists. This will expect relatively longer time for the first several iterations. As the iteration process goes, the used cost and weight (C_u, W_u) increase, and remaind cost and weight decrease. Each agent spends less time searching for potential trading compared with in earlier iterations.

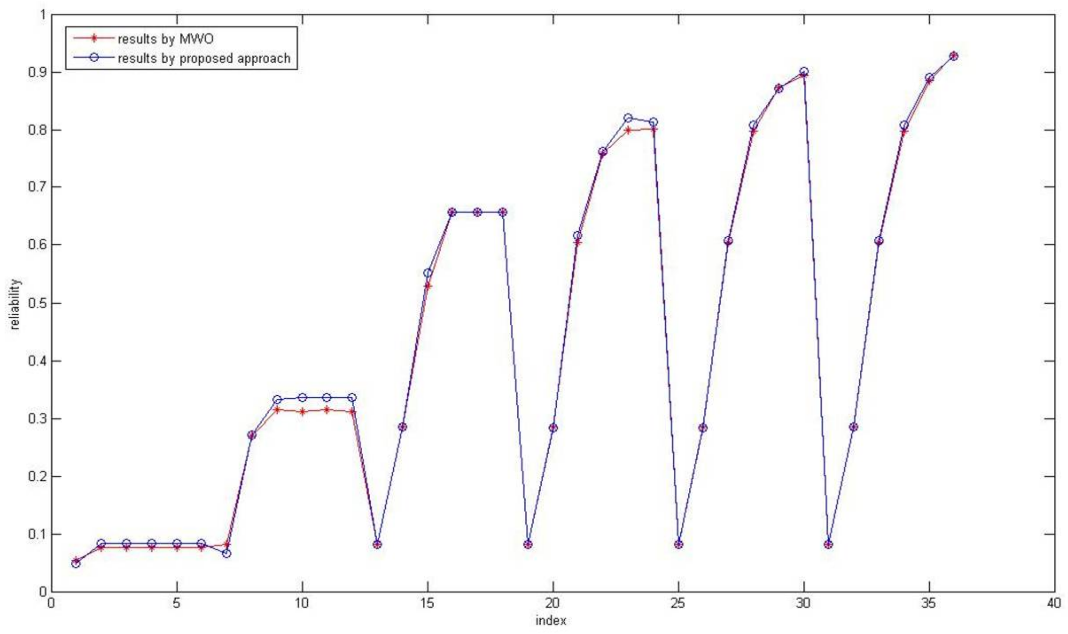
6.3 Analysis of complexity

The correctness of this approach is discussed in Chapter 5. Here, we will analyze the complexity. The computational time of this procedure is rational. We can break the whole process into three phases.

1. In the first phase, agents are created, one for each subsystem. These agents will take the components attributes as input, and carry out mixed linear programming autonomously under different cost and weight constraints. Thus, the possible permissible data sets are constructed. This phase is traverse computation, however, it is guaranteed to be linear time because there are at worst $C \times W \times s$ mixed integer linear programming problems to be solved. The scale of each linear programming is tiny. All of those are m_i variables, 2 constraints LP problems.

2. The second phase is the initial solution selection. For most cases, this part is not crucial and unnecessary, but it can improve efficiency. For the cases which need this phase, as mentioned in the failure attempts part, it involves s^2 comparison at worst. So, it is also linear time.

3. The third phase is the trading part. each agent computes fixed few trading attempts in each round, which is approximated to be $O((l_i)^2(s-1))$. The "broker" agent only update the new overall reliability, which involves s multiplications. The total time depends on the iteration rounds and the accuracy requirement.



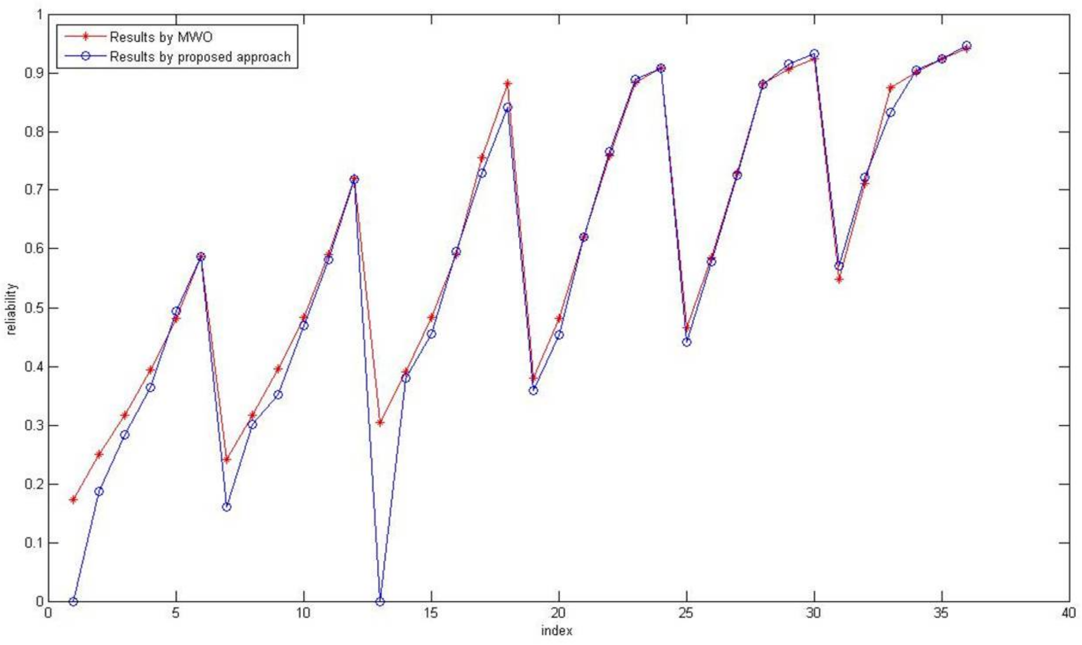


Figure 6.3: Comparison of results with Multiple Weighted Objectives Heuristic approach for test problems 2 and 3

Chapter 7

Summary and Future Work

7.1 Summary

Although the proposed approach could reach an encouraging result, it is still in initial investigation stage that need for improvement. The integer linear programming algorithm we are using now for subsystem optimization is for solving generalized LP problems. Since our problem is relatively more specific, that is decision variables x_{ij} are certain to be integer while the corresponding coefficients are certain to be fractional, we plan to simplify it to a more efficient algorithm, aiming to eliminate unnecessary computing time.

we are going to stress more focus on investigating multi-agent trading. We plan to test trading between multiple agents and explore how to improve efficiency. By intuition, different ways of trading may yield same optimal results, but accomplish the task in different steps and time span.

With experiences of the test problems, we know that to make the whole system functionally operate, at least one component is guaranteed to be used; and to improve the performance, no subsystem uses less than one component as redundancy in order to avoid the "bottleneck" situation which is discussed in the previous chapter. Thus, we propose to document these experiences into rules and set up a rule base for the agents. For extending and universal modeling, each agent is relatively naive at the beginning of solving one problem. As the time passes, the agent will become more sophisticated with this specified problem and also, any other RAP data.

As a whole evaluation of an algorithm, RAM taken by it is also a critical criteria. We will address focus on this issue in the future research.

To summarize, we addressed our research concern on the redundancy allocation problem. We proposed a novel approach based on multi-agent system.

7.2 Future work

For future extension, the agents can be deployed on different devices so that the computation can be carried out in parallel. Currently, due to the limitation of equipment, the program was run on one laptop. However, since the agent platform is utilized, the communication between agents are in message form. So, the capability of distributing the agents are out there.

If the computation is carried out on physically separated devices, the delay of transmitting and receiving messages will be involved. A problem we can foresee is the asynchronization and message loss. For example, due to the delay of propagating the bidding messages, if a bidding message regarding the previous bidding round arrives during the current round, the bidding process would fall into a mess. Hence, more perfect trading mechanism and bidding protocols need to be carefully designed.

The criteria of checking a trading feasibility could be refined. Currently, in the program, the checking ends when $C_u \geq C_r$ and $W_u \geq W_r$. However, C_u and W_u are updated after the computation of attempted trading, which results in the case that sometimes the finally returned solution exceeds the constraints. Only solution of last round can be returned as feasible solution. To make the program more robust, more work can be put on this issue.

As mentioned in the problem definition, there are many other variant RAPs. For instance, the reliability of a component is multi-state or proportional instead of bi-state (operates or fails). How to modify the agent framework to adapt other forms of RAP? This can be an interested topic for further exploration.

References

[1] D. Coit and A. Konak, “Multiple weighted objectives heuristic for the redundancy allocation problem,” *Reliability, IEEE Transactions on*, vol. 55, no. 3, pp. 551–558, Sep. 2006.

[2] Y.-C. Liang and Y.-C. Chen, “Redundancy allocation of series-parallel systems using a variable neighborhood search algorithm,” *Reliability Engineering and System Safety*, vol. 92, pp. 323–331, 2007.

[3] C. News, “Google uncloaks once-secret server,” $http : //news.cnet.com/8301 - 1001_3 - 10209580 - 92.html$.

[4] P. A. l. R. K.Thangavel, M.Karnan and G.Geetharamani, “Ant colony algorithms in diverse combinational optimization problems - a survey,” *ACSE Journal*, vol. 6, no. 1, pp. 7–26, Jan 2006.

[5] C. MS, “On the computational complexity of reliability redundancy allocation in a series system,” *Operation Research Letter*, vol. 11, no. 5, pp. 309–315, Jun 1992.

[6] D. E. Fyffe, W. W. Hines, and N. K. Lee, “System reliability allocation and a computational algorithm,” *Reliability, IEEE Transactions on*, vol. R-17, no. 2, pp. 64 –69, Jun. 1968.

[7] R. B. Günther Zäpfel and M. Bögl, *Metaheuristic Search Concepts*, 1st ed., C. Rauscher, Ed. Springer, 2010.

[8] Y. Shoham and K. Leyton-Brown, *Multi-agent Systems.* Cambridge University Press, 2009.

[9] E. D. Alex Rogers and N. R. Jennings, “The effects of proxy bidding and minimum bid increments within ebay auctions,” University of Southampton and JEREMY SCHIFF

Bar-Ilan University, Department of ECE, Southampton University, Southampton, SO17 1BJ, U.K, Tech. Rep., Aug 2007.

[10] A. Billionnet, “Redundancy allocation for series-parallel systems using integer linear programming,” *Reliability, IEEE Transactions on*, vol. 57, no. 3, pp. 507 –516, Sep. 2008.

[11] W. Kuo and R. Wan, “Recent advances in optimal reliability allocation,” *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, vol. 37, no. 2, pp. 143–156, Mar. 2007.

[12] R. Tavakkoli-Moghaddam and J. Safari, “A new mathematical model for a redundancy allocation problem with mixing components redundant and choice of redundancy strategies,” *Applied Mathematical Sciences*, vol. 1, no. 45, pp. 2221–2230, 2007.

[13] J. E. Ramirez-Marquez and D. W. Coit, “A heuristic for solving the redundancy allocation problem for multi-state series-parallel systems,” *Reliability Engineering and System Safety*, vol. 83, pp. 341–349, Oct 2004.

[14] Y.-C. Liang and A. Smith, “An ant colony optimization algorithm for the redundancy allocation problem (rap),” *Reliability, IEEE Transactions on*, vol. 53, no. 3, pp. 417–423, Sep 2004.

[15] L. T. Fan, C. S. Wang, F. A. Tillman, and C. L. Hwang, “Optimization of systems reliability,” *IEEE Transactions on Reliability*, vol. R-16, no. 2, pp. 81–86, Sep 1967.

[16] K. MISRA, “Dynamic programming formulation of the redundancy allocation problem,” *Int. Journal of Math. Edu. in Sci. and Tech.*, vol. 2, no. 3, pp. 207–215, 1971.

[17] K. Y. Ng and N. Sanch, “A hybrid 'dynamic programming/depth-first search' algorithm, with an application to redundancy allocation,” *IIE Transactions*, vol. 33, no. 12, pp. 1047–1058, 2001.

[18] D. W. Coit and A. E. Smith, "Stochastic formulations of the redundancy allocation problem," 1996.

[19] Y.-C. Hsieh, "A linear approximation for redundant reliability problems with multiple component choices," *Computer and Industrial Engineering*, vol. 44, pp. 91–103, 2002.

[20] R. Luus, "Optimization of system reliability by a new nonlinear integer programming procedure," *IEEE Transactions on Reliability*, vol. R-24, no. 1, pp. 14–16, Apr 1975.

[21] D. Coit and A. Smith, "Reliability optimization of series-parallel systems using a genetic algorithm," *Reliability, IEEE Transactions on*, vol. 45, no. 2, pp. 254–260, 266, Jun 1996.

[22] D. W. Coit and A. E. Smith, "Solving the redundancy allocation problem using a combined neural network/genetcic algorithm approach," *Computers Ops Res*, vol. 23, no. 6, pp. 515–526, 1996.

[23] L. Zia and D. W. Coit, "Redundancy allocation for series-parallel systems using a column generation approach," Industrial and Systems Engineering, Rutgers University, Tech. Rep., 2005.

[24] A. E. S. Sadan Kulturel-Konak and D. W. Coit, "Efficiently solving the redundancy allocation problem using tabu search," *IIE Transactions*, vol. 35, no. 6, pp. 515–526, 2003.

[25] M. N. Mohamed Ouzineb and M. Gendreau, "Availability optimization of series-parallel multi-state systems using a tabu search meta-heuristic," *International Conference Service Systems and Service Management*, vol. 1, no. 4244-0451, pp. 953–958, Oct 2006.

[26] S.-H. C. Tsen-I Kuo and J.-E. Chiu, "Tabu search in the redundancy allocation optimization for multi-state series-parallel systems," *Journal of the Chinese Institute of Industrial Engineers*, vol. 24, no. 3, pp. 210–218, 2007.

[27] M.-H. L. Liang Y-C and Y.-C. Chen, "Variable neighbourhood search for redundancy allocation problems," *IMA Journal of Management Mathematics*, vol. 18, pp. 135–155, 2007.

[28] L. Y-C and W. C-C, "A variable neighborhood descent algorithm for the redundancy allocation problem," *Ind Eng Manage Syst*, vol. 4, no. 1, pp. 109–16, 2005.

[29] Y.-C. Liang and A. Smith, "An ant system approach to redundancy allocation," in *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, 1999.

[30] M. Dorigo, "Optimization, learning and natural algorithms," Ph.D. dissertation, Politecnico di Milano, Italy, 1992.

[31] J. Onishi, S. Kimura, R. James, and Y. Nakagawa, "Solving the redundancy allocation problem with a mix of components using the improved surrogate constraint method," *IEEE Transactions on Reliability*, vol. 56, no. 1, pp. 94–101, Mar 2007.

[32] C. Ha and W. Kuo, "Multi-path approach for reliability-redundancy allocation using a scaling method," *Journal of Heuristics*, vol. 11, pp. 201–217, 2005.

[33] R.-M. J and K. A. Coit D, "Redundancy allocation for series-parallel systems using a maxcmin approach," *IIE Trans*, vol. 36, no. 9, pp. 891–898, 2004.

[34] Y. Nakagawa and S. Miyazaki, "Surrogate constraints algorithm for reliability optimization problems with two constraints," *IEEE Transactions on Reliability*, vol. R-30, no. 2, pp. 175–180, Jun. 1981.

[35] O. Bendjeghaba and D. Ouahdi, "Multi-agent ant system for redundancy allocation problem of multi states power system," Dec 2008, pp. 1270–1274.

[36] W. Kuo and V. Prasad, "An annotated overview of system-reliability optimization," *Reliability, IEEE Transactions on*, vol. 49, no. 2, pp. 176 –187, Jun. 2000.

[37] R. Bellman, "Dynamic programming and lagrange multiplier," *Proc. Nat'l Acad. Sci. (USA)*, vol. 42(10), pp. 767–769, 1956.

[38] I. S.A. and I. ILOG, *ILOG CPLEX 10.0 Users Manual*, ILOG S.A. and ILOG, Inc, Jan 2006.

[39] T. S. Ferguson, *Linear programming - A Concise Introduction.*

[40] E. lawler, "Fast approximation algorithms for knapsack problems," University of California at Berkeley, and in part by the Mathematical Center, Amsterdam, Berkeley, California 94720, National Science Foundation grant MCS76-17605, January-February 1977.

[41] G. C. Fabio Bellifemine and G. Rimassa, "Java agent development framework," $http://jade.tilab.com/$.

Appendix A

Notations

s	the number of subsystems connected in parallel;
m_i	the number of available component choices for the i-th subsystem;
c_{ij},w_{ij}	the cost and weight of the jth component in the ith subsystem.
Ψ_i	the parameter set for the i-th subsystem;
$\mathcal{R}s, \mathcal{C}s, \mathcal{W}s$	the extract function for reliability, cost and weight;
$\bar{x}$	the average amount of identical components that can be used if all other subsystems use only one component;
t	the total number of component types in the overall system;
Γ_i	the performance set of subsystem i;
$\phi(\gamma_q(j), \gamma_p(i))$	the bidding message sent by agent j;
R	the overall system reliability;
C, W	total cost and total weight constraints;
$subC_i$,$subW_i$	the cost and weight constraints for subsystem i;
$c_l^{(i)}$, $w_l^{(i)}$	the actually used cost and weight in the i-th subsystem after linear optimization;
$\mathbf{X}_l^{(i)}$	the component assignment of the l-th performance set in the i-th subsystem;
$\widetilde{c}$, $\widetilde{w}$	the cost and weight constraints which are used to in subsystem's optimization;
C_u, W_u	the actually used cost and weight in the overall system optimization;
C_r, W_r	the remained cost and weight that are not used after each round of optimization.

Appendix B

Data Sets

The permissible performance data sets Γ_1 for subsystem 1, where the bolded set is the initially selected set

(0.9, 1 , 3 , 1) (0.91, 2 , 2 , 1)

(0.93, 1 , 4 , 1) (0.95, 2 , 5 , 1) (0.99, 2 , 6 , 2)

(0.991, 3 , 5 , 2) **(0.9919, 4 , 4 , 2)** (0.993, 2 , 7 , 2)

(0.9937, 3 , 6 , 2) (0.9951, 2 , 8 , 2) (0.9955, 4 , 7 , 2)

(0.999, 3 , 9 , 3) (0.9991, 4 , 8 , 3) (0.99919, 5 , 7 , 3)

(0.999271, 6 , 6 , 3) (0.9993, 3 , 10 , 3) (0.99937, 4 , 9 , 3)

(0.999433, 5 , 8 , 3) (0.99951, 3 , 11 , 3) (0.999559, 4 , 10 , 3)

(0.999595, 6 , 9 , 3) (0.999657, 3 , 12 , 3) (0.9999, 4 , 12 , 4)

(0.99991, 5 , 11 , 4) (0.999919, 6 , 10 , 4) (.9999271, 7 , 9 , 4)

(0.99993, 4 , 13 , 4) (.9999344, 8 , 8 , 4) (0.999937, 5 , 12 , 4)

(.9999433, 6 , 11 , 4) (0.999949, 7 , 10 , 4) (0.999951, 4 , 14 , 4)

(.9999559, 5 , 13 , 4) (.9999603, 6 , 12 , 4) (.9999636, 8 , 11 , 4)

(.9999657, 4 , 15 , 4) (.9999691, 5 , 14 , 4) (0.999976, 4 , 16 , 4)

(0.99999, 5 , 15 , 5) (0.999991, 6 , 14 , 5) (.9999919, 7 , 13 , 5)

(.9999927, 8 , 12 , 5) (0.999993, 5 , 16 , 5) (0.9999934, 9 , 11 , 5)

(0.9999937, 6 , 15 , 5) (0.9999941, 10 , 10 , 5) (0.9999943, 7 , 14 , 5)

(0.9999949, 8 , 13 , 5) (0.9999951, 5 , 17 , 5) (0.9999954, 9 , 12 , 5)

(0.9999956, 6 , 16 , 5) (0.999996, 7 , 15 , 5) (0.9999964, 8 , 14 , 5)

(0.9999966, 5 , 18 , 5) (0.9999967, 10 , 13 , 5) (0.9999969, 6 , 17 , 5)

(0.9999972, 7 , 16 , 5) (0.9999976, 5 , 19 , 5) (0.9999983, 5 , 20 , 5)

(0.999999, 6 , 18 , 6) (0.9999991, 7 , 17 , 6) (0.9999992, 8 , 16 , 6)
(0.9999993, 9 , 15 , 6) (0.9999993, 6 , 19 , 6) (0.9999993, 10 , 14 , 6)
(0.9999994, 7 , 18 , 6) (0.9999994, 11 , 13 , 6) (0.9999994, 8 , 17 , 6)
(0.9999995, 12 , 12 , 6) (0.9999995, 9 , 16 , 6) (0.9999995, 6 , 20 , 6)
(0.9999995, 10 , 15 , 6) (0.9999996, 7 , 19 , 6) (0.9999996, 11 , 14 , 6)
(0.9999996, 8 , 18 , 6) (0.9999996, 9 , 17 , 6) (0.9999997, 6 , 21 , 6)
(0.9999997, 10 , 16 , 6) (0.9999997, 7 , 20 , 6) (0.9999997, 12 , 15 , 6)
(0.9999997, 8 , 19 , 6) (0.9999997, 9 , 18 , 6) (0.9999998, 6 , 22 , 6)
(0.9999998, 6 , 23 , 6) (0.9999999, 6 , 24 , 6) (0.9999999, 7 , 21 , 7)
(0.9999999, 8 , 20 , 7) (0.9999999, 9 , 19 , 7) (0.9999999, 10 , 18 , 7)
(0.9999999, 7 , 22 , 7) (0.9999999, 11 , 17 , 7) (0.9999999, 8 , 21 , 7)
(0.9999999, 12 , 16 , 7) (0.9999999, 9 , 20 , 7) (0.9999999, 13 , 15 , 7)
(0.9999999, 10 , 19 , 7) (1, 7 , 23 , 7) (1, 14 , 14 , 7)
(1, 11 , 18 , 7) (1, 8 , 22 , 7) (1, 12 , 17 , 7)
(1, 9 , 21 , 7) (1, 13 , 16 , 7) (1, 10 , 20 , 7)
(1, 7 , 24 , 7) (1, 11 , 19 , 7) (1, 8 , 23 , 7)
The rest more than 3000 sets have reliability approximate to 1

The permissible performance data sets Γ_2 for subsystem 1, where the bolded set is the initially selected set
(0.93, 1 , 9 , 1) (0.94, 1 , 10 , 1)
(0.95, 2 , 8 , 1) (0.9951, 2 , 18 , 2) (0.9958, 2 , 19 , 2)
(0.9964, 2 , 20 , 2) **(0.9965, 3 , 17 , 2)** (0.997, 3 , 18 , 2)
(0.9975, 4 , 16 , 2) (0.999657, 3 , 27 , 3) (0.999706, 3 , 28 , 3)
(0.999748, 3 , 29 , 3) (0.999755, 4 , 26 , 3) (0.999784, 3 , 30 , 3)
(0.99979, 4 , 27 , 3) (0.99982, 4 , 28 , 3) (0.999825, 5 , 25 , 3)
(0.99985, 5 , 26 , 3) (0.999875, 6 , 24 , 3) (0.999976, 4 , 36 , 4)
(0.9999794, 4 , 37 , 4) (0.9999824, 4 , 38 , 4) (0.9999828, 5 , 35 , 4)

(0.9999849, 4 , 39 , 4) (0.9999853, 5 , 36 , 4) (0.9999870, 4 , 40 , 4)

(0.9999874, 5 , 37 , 4) (0.9999877, 6 , 34 , 4) (0.9999892, 5 , 38 , 4)

(0.9999895, 6 , 35 , 4) (0.9999910, 6 , 36 , 4) (0.9999913, 7 , 33 , 4)

(0.9999925, 7 , 34 , 4) (0.9999938, 8 , 32 , 4) (0.9999983, 5 , 45 , 5)

(0.9999986, 5 , 46 , 5) (0.9999988, 5 , 47 , 5) (0.9999988, 6 , 44 , 5)

(0.9999989, 5 , 48 , 5) (0.9999990, 6 , 45 , 5) (0.9999991, 5 , 49 , 5)

(0.9999991, 6 , 46 , 5) (0.9999991, 7 , 43 , 5) (0.9999992, 5 , 50 , 5)

(0.9999992, 6 , 47 , 5) (0.9999993, 7 , 44 , 5) (0.9999994, 6 , 48 , 5)

(0.9999994, 7 , 45 , 5) (0.9999994, 8 , 42 , 5) (0.9999995, 7 , 46 , 5)

(0.9999995, 8 , 43 , 5) (0.9999995, 8 , 44 , 5) (0.9999996, 9 , 41 , 5)

(0.9999996, 9 , 42 , 5) (0.9999997, 10 , 40 , 5) (0.9999999, 6 , 54 , 6)

(0.9999999, 6 , 55 , 6) (0.9999999, 6 , 56 , 6) (0.9999999, 7 , 53 , 6)

(0.9999999, 6 , 57 , 6) (0.9999999, 7 , 54 , 6) (0.9999999, 6 , 58 , 6)

(0.9999999, 7 , 55 , 6) (0.9999999, 8 , 52 , 6) (0.9999999, 6 , 59 , 6)

(0.9999999, 7 , 56 , 6) (0.9999999, 8 , 53 , 6) (1, 6 , 60 , 6)

(1, 7 , 57 , 6) (1, 8 , 54 , 6) (1, 9 , 51 , 6)

(1, 7 , 58 , 6) (1, 8 , 55 , 6) (1, 9 , 52 , 6)

(1, 8 , 56 , 6) (1, 9 , 53 , 6) (1, 10 , 50 , 6)

The rest more than 300 sets have reliability approximate to 1