AMERICAN UNIVERSITY OF BEIRUT

THE SIMULTANEOUS OPTIMIZATION OF PRODUCTS, PROCESSES, AND TEAMS IN PRODUCT DEVELOPMENT ORGANIZATIONS

by
RAWIA HANNA CHIDIAC

A thesis
submitted in partial fulfillment of the requirements
for the degree of Master of Engineering Management
to the Engineering Management Program
of the Faculty of Engineering and Architecture
at the American University of Beirut

Beirut, Lebanon
February 2011

AMERICAN UNIVERSITY OF BEIRUT

THE SIMULTANEOUS OPTIMIZATION OF PRODUCTS, PROCESSES, AND TEAMS IN PRODUCT DEVELOPMENT ORGANIZATIONS

by
RAWIA HANNA CHIDIAC

Approved by:

Dr. Ali Yassine, Associate Professor Engineering Management Program	Advisor
Dr. Ibrahim Osman, Professor Suliman S. Olayan School of Business	Member of Committee
Dr Bacel Maddah, Assistant Professor Engineering Management Program	Member of Committee

Date of thesis defense: February 18, 2011

AMERICAN UNIVERSITY OF BEIRUT

THESIS RELEASE FORM

I, Rawia Hanna Chidiac

☐ authorize the American University of Beirut to supply copies of my thesis to libraries or individuals upon request.

☐ do not authorize the American University of Beirut to supply copies of my thesis to libraries or individuals for a period of two years starting with the date of the thesis defense.

Signature

Date

ACKNOWLEDGMENTS

This work would not have been possible without any help and cooperation. Hence, it is of great pleasure to acknowledge the enthusiastic support and assistance given to me by Professor Ali Yassine, Professor Ibrahim Osman and Professor Bacel Maddah and to whom I would like to express my gratitude and respect.

My deep appreciation goes, however, to my advisor Ali Yassine for his guidance, advice and patience; and whose skill, hard work and high level of professionalism made it possible to produce this work along with Professor Ibrahim Osman to whom I hand a special appreciation. I also thank Dr Walid Nasr for his help and support.

My family support was of great benefit to this work outcome and especially my brother Hicham Chidiac.

Last but not least, a special gratitude is sent overseas to Marc Moukarzel for his help and support.

I thank you all for being such humble supporters of mine.

AN ABSTRACT OF THE THESIS OF

Rawia Hanna Chidiac for Master of Engineering Management
Major: Engineering Management

Title: The Simultaneous Optimization of Products, Processes, and Teams in Product Development Organizations

Organizations involved in product development (PD) constantly introduce new products using mainly development teams within the organization. These teams carry out product development activities using established development processes in order to produce a new product. Traditionally, these domains (product, process, and team) are treated separately and individual optimization occurs for each domain disregarding the other two domains. The result is a group of three local optimal solutions instead of a single global optimal one. The main goal of this research is to be able to formulate and solve a global optimal solution for the product development organization problem. The inter- and intra-dependencies within and between the three domains are captured using a matrix-based technique called the design structure matrix (DSM). Then three relational rules that relate the domains together are proposed to help formulate a global optimization objective function for the three domains. However, as the domains grow in size, finding an optimal solution becomes computationally prohibitive. Therefore, to overcome this difficulty, ten different methods were designed using heuristics (constructive and improvement) and meta-heuristic (simulated annealing) techniques. A software program using the JAVA language is designed to simulate the approaches. Six hundred random test instances were analyzed using these ten methods in order to recommend a single approach. The analysis showed the existence of a tradeoff between cheap and dirty approaches versus more accurate but expensive ones.

CONTENTS

ILLUSTRATIONS

TABLES

DEDICATION

I dedicate this study to the most precious parents who sacrificed their lives

to raise my special family

Warde R.Chidiac & Hanna Chidiac

CHAPTER 1

INTRODUCTION

1.1. Problem Statement

Product development (PD) organizations continuously introduce new products using mainly development teams within the organization. These *teams* carry out product development activities using established development *processes* in order to produce a new *product*. The problem addressed in this thesis is how to manage such organizations through simultaneous optimization of the above-mentioned organizational domains: team, process and product.

The current techniques target the optimization of the above domains separately and in isolation, thus do not achieve a global optimal solution which is the core of the work. The Design Structure Matrix (DSM) tool has successfully represented and analyzed the architecture of a product, a process schedule and an organization, yet still it has treated the three domains independently.

As a matter of fact, DSM models have been applied to perform either of the following:

(a) optimize the sequence of activities or tasks within a process or project;

(b) optimize the architecture of a product through increasing its modularity;

(c) form optimal product development (PD) teams.

These models have not been applied, however, to optimize all the three DSM domains simultaneously in a single approach. The present thesis proposes a new DSM-based method that allows for a simultaneous analysis and optimization of the three multi-DSM domains as well as establishing rules to connect the said domains and get an optimal solution collectively.

The achievement of this new DSM-based method is illustrated by (i) decomposing the problem into significant subsystems and elements and (ii) collecting the domains information using the help of the organizations expertise. The information required to construct the three DSM domains will be collected either by interviewing experts from various disciplines involved in PD, or by reviewing the design manuals and procedures of the targeted organizations. Although applicable in small problem contexts, such new multi-domain DSM-based method tends to be inapplicable in large organizations where exhaustive searching for an optimal solution is computationally inefficient and expensive. In this case, this method will be replaced by *simulated annealing* (SA) which is a meta-heuristic technique distinguished from different search algorithms by its ability to accept non-improving solutions as a means to avoid a local optimal solution.

In sum, as the domains grow in size, finding an optimal solution becomes computationally prohibitive. Therefore, to overcome this difficulty, ten different methods were designed using heuristics (constructive and improvement techniques) and meta-heuristics (simulated annealing). Six hundred random test instances were analyzed using these ten methods in order to recommend a single approach. The analyses showed the existence of a tradeoff between cheap and dirty approaches versus more accurate but expensive approaches.

1.2. Scope of work and Significance of Study

This thesis is directed towards PD managers in various size development organizations while managing a PD process in running their organization. It provides them with (i) a software solution to define and organize their product modules, (ii) the order of the processes they have to follow, and (iii) the most possible adequate PD team across the organization.

Engineering, business, and human resource expertise will all contribute to collecting information/data needed for each of the three domains. Engineering experts manage the product domain, project management experts manage the process domain, and the organizational design experts manage the team domain. The JAVA tool is developed through a friendly interface where the data collected are entered in an excel file according to a predefined template (Appendix-A). This will give the above-mentioned experts the ease of using the software to manage optimally the three domains simultaneously. The optimal solution achieved will be shown finally in a new excel file.

CHAPTER 1 has stated the problem along with the scope of work and significance of study. The literature review will be highlighted in CHAPTER 2 and CHAPTER 3 describing the design structure matrix and the existing optimization techniques using the heuristic and meta-heuristic search methods and models CHAPTER 4 illustrates the work to formulate the simultaneous optimization objective function of the three domains using a multi-domain DSM model and represents the three relational rules of inter and intra dependencies within and between the domains. Moreover, CHAPTER 5 describes the ten hybrid methods combining heuristics and meta-heuristic techniques to solve large size problems. The performance of these ten approaches is tested in CHAPTER 6 which includes the analysis of six hundred random test instances. The most recommended method shows a tradeoff between the quality of the solution and the required computational CPU time.

CHAPTER 2

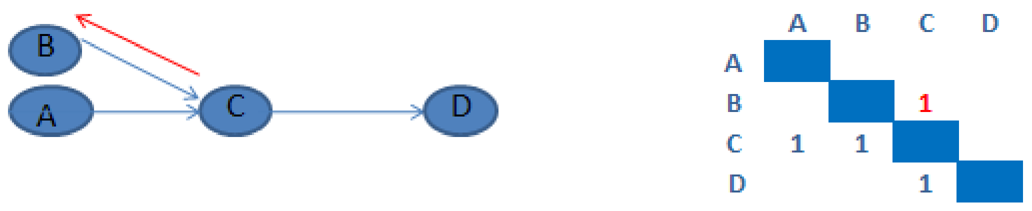
LITERATURE REVIEW

2.1. Introduction

Product development is a vital activity for organizations. The key factor for determining the corporate health and profitability is the ability to launch quickly new salable products into the market (Clark & Fujimoto, 1991). The difficulty of managing a new product development process is due to, among others, increased global competition, frequent consumer taste changes, and rapid advancements in science and technology. To overcome these difficulties, a variety of tools is put to application.

2.2. Design Structure Matrix

One of the tools that help in the proper management of PD projects is the Design Structure Matrix (DSM) which has proved efficient in representing and analyzing the architecture of a process, a product, or a team (Danilovic & Browning, 2007). Unlike traditional project management tools, such as PERT, Gantt and CPM methods, the DSM can capture the dependency relationship between hundreds and thousands of elements and can also provide means of analysis for feedback. The traditional management project tools address work flows only, while the DSM focuses on representing the information flows among its elements and characterizes the complex relations between tasks, members of teams and components. The DSM is also able to determine sequences of tasks as well as to group teams or products into modules (Yassine, 2004).



(a) Directed graph (b) DSM model

Fig. 2.1.The DSM model and its underlying graph

The DSM representation illustrates the information flow captured in a directed graph shown in Fig. 2.1.a. The DSM is actually a binary square matrix having, in the same sequence, the same title headings in the rows and columns. This matrix input is illustrated by either a 1 or a 0 and is meaningless on its diagonal. The diagonal mark could be either blacked out as shown in Fig. 2.1.b or left empty. If any two elements are related such as 'A' and 'B', then this relation should be shown in the DSM. Hence, if element 'i' feeds element 'j', then the value of the element 'ij' (column i, row j) is 1. Accordingly, by using a design structure matrix, it is easy to represent information relations among (a) teams concurrently working on a project, (b) activities, and (c) components of a product.

2.2.1. Task Based Domain

The task based domain is represented by input/output relationships where a certain task can be an input for another task, an output fed by other tasks, or a standalone task. Hence, the task relations illustrated through the DSM may vary from parallel to sequential to coupled relations. The DSM model in Fig. 2.1.b is a project composed of activities to be executed in the following sequence: A, B, C, and D.

Parallel relation exists between A and B; no relational dependency exists between them. "A" and "B" are executed concurrently. In the event C feeds B, the red '1' shown in the DSM model represents a feedback mark which means that the needed inputs for B are not available the first time B is executed. This would require B to be re-executed once the output of C becomes available and would increase the development lead time (Meier, Yassine & Browning, 2007). The dependency exiting between B and C indicates also that a coupled relationship joins both activities. Information cycle is thus presented between both activities where each activity requires input from the other activity to be able to start (Yassine, 2004). Yet, the relationship between C and D is sequential. D is dependent on the result of C; this means that task D is fed by the result of C before its execution.

As a result, the execution arrangement of the tasks affects the solution given the fact that the optimality of the task domain depends on reducing feedback mark, increasing concurrent tasks and reducing the development lead times and cost (Meier et al., 2007). Some of the methods used in analyzing and optimizing this domain are: Sorting, Partitioning, Tearing, Banding, Simulation and Eigenvalue Analysis (Browning, 2001). The Sorting and Partitioning methods will be later discussed in detail.

2.2.2. Team Based Domain

The team based domain is represented by person-to-person interface characteristics and is mainly used in organizational design, interface management, and team integration. In the team DSM, the rows and the columns of the matrix identify the individuals or groups participating in a project.

The information of a team-based DSM is constructed by identifying the communication skills and their factors. The following are taken into consideration: (i) the level of detail (emails and documents sharing versus models or face to face communication), (ii) the frequency of communication between members, and (iii) the direction of the information flow (where one way or two way talks can happen) (Yassine, 2004).

The matrix built from the information acquired is then used for optimization by applying the clustering techniques in order to gather highly interacting groups and minimize scattered groups. In this way, the organization teams obtained represent a useful structure for the organization where the communication needs of each member are justified.

2.2.3. Product Based Domain

The product based domain may be represented by multi-component relationships; it is mainly used in system architecting, engineering and design. Such DSMs represent the product architectures by pointing and analyzing relationships between subsystems and components included in a product. Irrespective of the complexity of the product, decomposition into smaller sub-problems (Eppinger, 1997) along with understanding the interactions among components is quite essential.

Different product characteristics, being design requirements, product components or design parameters, are represented in the DSM. They range from *spatial* (identifying the need for adjacency between two elements) to *energy* (where two elements need to exchange energy between them). Additionally, an *information* interaction will exist when two elements share information or a *material* exchange interaction occurs between them. Various clustering techniques are applied to maximize interactions between elements of the same cluster and minimize interactions between clusters, thus resulting in cluster modules which combine related products.

2.3. Domain Mapping Matrix (DMM) and Multi Domain Matrix (MDM)

Traditional project management techniques consider these three domains independently. Sometimes, however, development of complex products exhibit dependencies and conflicts among these domains. This has led to a shift in research towards multi-project environments where complex products and their interdependencies among different domains exist. The more the relations and interdependencies among domains, the larger is the complexity and the more crucial is the reduction and analysis of such complexity (Danilovic & Sandkull, 2005).

Complexity could stem from customer demands, functional requirements and specifications; it could also originate from the technology world where the product design is evolving and the tasks to solve technical problems are re-assigned. Diversity in personnel skills and the way to organize teams may be deemed another source of complexity. Management should thus consider, understand and solve such complexity sources (Danilovic & Sandkull, 2005).

One of the two approaches can be applied:

(i) *Closer consideration for subsets or system view extracted from the overall system*:

This approach reduces complexity, but the analysis will not be that beneficial because the analysis deals with only one subsystem (Lindemann et al., 2009). Actually, a complex product or system can be divided into sub-systems or components illustrating the design requirements, product components or design parameters. A complex process will be subdivided as well into phases or sub-processes that will be decomposed later into tasks and activities. The organization domain will be also divided into teams that will be further divided into working groups and individual actors (Tang et al., 2010).

(ii) *Consideration and analysis of abstract system level*:

This approach leads to general findings given that a wide scope is considered and details are neglected (Lindemann et al., 2009).

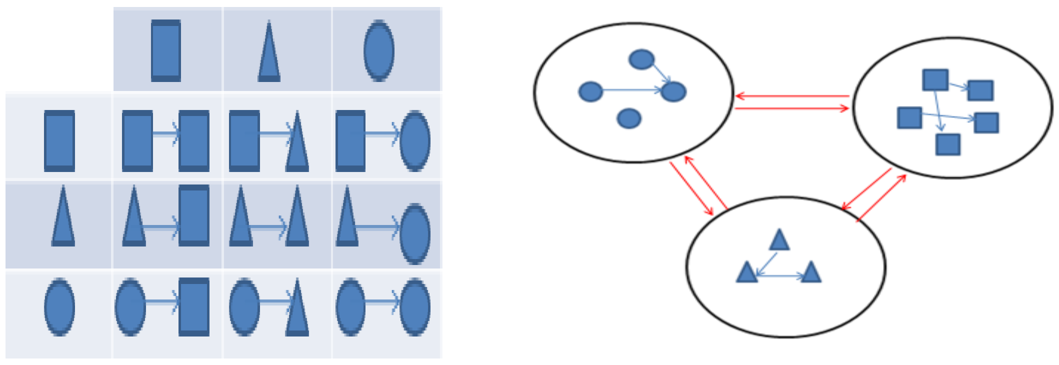
It is clear by now that product development is a multi project environment where uncertainty is a fact to be embraced. The main source of such uncertainty is the limitations put on the flow of information. These include, among others, limitations to understand the kind of information required to choose the information source and to be sure of the information availability when needed. As the uncertainty level increases, the assumption level increases as well involving ambiguity in approaches and a high risk factor caused by the creation of rework due to new knowledge or to change of requirements.

The understanding of the relations and the interdependencies between domains is studied by applying the design structure matrix DSM (N×N) and the domain mapping matrix DMM (N×M) (Danilovic & Sandkull, 2005). To show self-dependency of a certain domain, the DSM illustration is used. Nevertheless, by applying the DMM, Danilovic (2007) demonstrates the way one can study relations between two domains and how to relate elements of one domain to another.

DMM is in fact a rectangular matrix relating two DSMs. Its analysis holds benefits in capturing the dynamics of PD along with showing traceability of constraints and providing transparency among domains. Decisions among domains are synchronized and the sense of communication across domains is improved (Danilovic & Browning, 2007).

The analysis techniques used in the DMM are (i) the clustering algorithm (across two domains i.e. not across its diagonal as used in the DSM) in addition to (ii) the sequencing analysis technique. The combination of these techniques contributes in reducing the uncertainty factor by visualizing the interdependencies and relations and by exploring the need for information exchange (Danilovic & Browning, 2007). However, information needed for the DMM is collected by using existing databases, modeling tools or interviews. Still, the difficulty lies in the ability to collect efficient and high quality data (Lindemann et al., 2009).

The dialogues and the meetings formed, while generating the DSM and the DMM, create a strong responsibility and commitment to the organization as well as deep understanding of the organization work. Dependencies among domains are moreover deduced. These dependencies are represented using the MDM presentation (Lindemann et al., 2009) as shown in Fig. 2.2 below:



a) Multi domain matrix   b) Dependencies inside and among domains

Fig. 2.2. MDM presentation

MDM is actually a square matrix with row and column headings being the names of the domains. The MDM shown in Fig. 2.2.a illustrates different combinations among domains according to specific dependency types.   Various types of dependencies are displayed in the directed graph shown in Fig. 2.2.b. Dependencies between same domains are shown in a blue arrows whereas dependencies among different domains are represented by red arrows. Dependency types vary between information flow, change impact, geometric and thermal dependencies (Lindemann et al., 2009).

The MDM can thus be divided into DSMs and DMMs according to inherent domains. If a dependency connects elements from the same domain, such information will be used in the DSM. If a dependency connects elements from a different domain, dependency information will be used in the DMM.
These representations are helpful to analyze the domains and to dilute the uncertainty factor existing among them (Lindemann et al., 2009).

Different papers tackle different methodologies to optimize multi-domain architectures. Eppinger (1997), for example, discusses the possible relationship between DSMs in the three domains: product, process and team. These papers   show that there is a one-to-one mapping between one domain and the other. A direct comparison is then straightforward. On the other line of the spectrum, Dan Braha (2002) worked on the task partitioning problem and tried to make the hard non-deterministic polynomial time problem (NP hard problem) more lenient. Braha limited the number of tasks assigned to teams as he considered that each team has a certain capacity limit, and that each task should be performed by exactly one team.

2.4. DSM Analysis Techniques

Various techniques and approaches evolved in the product development system. In this literature review, we will shed light in detail on just two: the *sorting and partitioning method* used as an optimization technique in the process domain along with the *clustering technique* used to get the best grouping in the team and product domains.

2.4.1. Sorting and Partitioning Method

The objective in optimizing the process domain is to reduce the number of feedback marks which holds a negative connotation for the project in terms of time and cost due to the potential of rework. For this reason, partitioning method is used where the sequence of DSM rows and columns are reordered to try to get a new DSM arrangement with no feedback marks. This method transforms the DSM into a lower triangular form unattainable most of the time because of the complexity in relations among tasks in engineering systems (Yassine, 2004).

Therefore, a modified approach known as block triangular is used in order to put the feedback marks as close as possible to the diagonal. This approach decreases the iteration cycle time and results in a faster development process. Fig. 2.3 below shows the result of optimizing a 4×4 Process Matrix.

$$
\begin{array}{ccccc}
 & 1 & 2 & 3 & 4 \\
1 & \blacksquare & 0 & 1 & 1 \\
2 & 0 & \blacksquare & 0 & 0 \\
3 & 0 & 0 & \blacksquare & 0 \\
4 & 0 & 1 & 0 & \blacksquare
\end{array}
\qquad\qquad
\begin{array}{ccccc}
 & 2 & 3 & 4 & 1 \\
2 & \blacksquare & 0 & 0 & 0 \\
3 & 0 & \blacksquare & 0 & 0 \\
4 & 1 & 0 & \blacksquare & 0 \\
1 & 0 & 1 & 1 & \blacksquare
\end{array}
$$

Fig. 2.3. Result of optimizing a 4×4 process matrix

Manipulating large DSM is a hard task that takes a lot of time and consumes a lot of computer memory. Algorithms are built to manipulate large DSMs in an easier and quicker way. In 1976, Lawler developed an efficient and quick sorting algorithm to order a DSM with no cyclic information if possible; otherwise, the partition method is applied (Yassine et al., 1999).

2.4.1.1. <u>DSM Sorting</u>

Lawler algorithm starts by finding the sum of all rows of each task in the DSM. A ranking of DSM is then made by putting the row with sum equal zero to be first in the DSM. This row, along with all its connections, is removed from the DSM and the approach is repeated till no more zero sum of row is found. Hence, if the approach ends with no cyclic information, then the optimized solution is obtained; otherwise, partitioning of the DSM is performed.

2.4.1.2. <u>DSM Partitioning</u>

The availability of cyclic information leads to terminate the sorting method and to target a DSM block triangular form (instead of a lower triangular one) for a faster development process. A block is the largest subset of a diagraph in which every subset has a path to every other node in the subset (Yassine et al., 1999). Partitioning analysis is used to identify the tasks in a loop and cluster them in a block along the diagonal of the DSM so that all predecessors of a block appear somewhere before that block.

There are different partitioning ways to identify a cycle where the elements involved in it are diluted into one node and the process is repeated to find other cycles. These cycles can be identified using either the path searching or the powers of the adjacency matrix methods. The path searching is traced backward until a node is encountered twice. The information cycle in this case constitutes the entire tasks that were passed through in this cycle. The power of the adjacency matrix method raises the binary DSM to a power n in order to trace which element can reach itself in n steps by looking to a non-zero entry for the relevant task along the diagonal of the matrix (Yassine et al., 1999).

As mentioned, Fig. 2.3 above shows an optimization result of a 4×4 DSM example. The independent task 2 and task 3 are sorted in the first rank of DSM and the other feedback marks were partitioned in blocks near the diagonal giving an optimized matrix with no feedbacks. The pseudocode of both procedures is shown in table (Yassine et al., 1999).

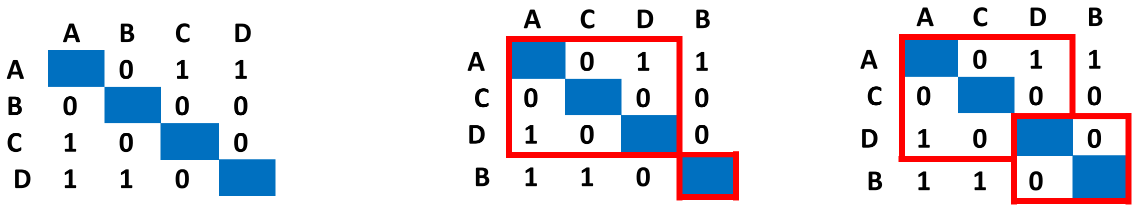
Table. 2.1. Sorting and partitioning pseudocode

```
Sorting
Step 1. (start: Find row-sum of tasks)
        Set $I_i = \sum_{j=1}^{n} aij$, $i = 1, . . . , n$.
        Set $N= \{1, 2, . . . , n\}$
        Set $m =1$
Step 2. (detection of node with 0 in-degree)
        Find $k \in N$ such that $I_k =0$. If there is no such $k$, stop; the digraph contains cycles.
        Set Rank $(k) = m$
        Set $m = m + 1$
        Set $N= N- k$
        If $N= \emptyset$ , stop; the computation is completed.
Step 3. (Revision of in-degrees)
        Set $I_i = I_i - a_{ik}$ for all $i \in N$
        Return to step 2.

Partitioning
Step 1. Determine all the circuits that exist in the DSM using either path searching or
        powers of the matrix.
Step 2. Collapse all tasks within the same circuit into a single representative task.
Step 3. Order the remaining tasks using procedure 1. If a cycle is detected, then go to
        step 1; otherwise, the procedure is complete.
```

2.4.2. Clustering Technique

By dealing with elements of DSM representing people in charge of (i) tasks or (ii) sub-systems and components of a larger system, one may manipulate the DSM in order to find subsets of DSM elements known as clusters. The foremost objective is to find subsets mutually exclusive or minimally interacting. Other objectives are considered such as, without limitation, minimizing cluster sizes, minimizing the size of the largest cluster, and allowing overlapping clusters. Clustering algorithm is a helpful integration analysis technique in this domain. Fig. 2.4.a shows a DSM example where the entries may represent the frequency or intensity of communication exchanged between different participants represented by person A, person B, etc.



a) Original DSM b) Clustered DSM c) Clustered DSM with overlapping

Fig. 2.4. Possible clustering solutions

The aim of clustering is to maximize interactions between elements of the same cluster and minimize interactions among clusters (Browning, 2001). For this reason, if the matrix is arranged in the following order: ACDB, the connections are clearly seen. As shown in Fig. 2.4.b, the original DSM was rearranged to contain most of the interactions within two separate blocks: ACD and B. The interactions, however, between clusters is not zero; still one interaction exists. Another alternative overlapping clustering is shown in Fig. 2.4.c. To decide which cluster option to choose is related to the targeted domain and the effect of its factors.

Yet, several computational clustering techniques that search for optimal solutions are based on tradeoffs between the importance of capturing intra block dependencies versus the importance of capturing inter block dependencies (Yassine et al., 1999). One such algorithm was proposed by Thebeau (2001) as a continuation of the work done by Carlos Fernanzed. This clustering algorithm calculates the cost of the proposed solution. The objective is to find the solution of the lowest cost. There is a higher cost for interactions occurring outside of clusters and a lower cost for interactions occurring within clusters. There are also penalties assigned to the size of clusters in order to avoid a solution where all elements are members of a single cluster.

The results are highly dependent on the parameters passed to the algorithm which is as follows: (Thebeau, 2001):

1. Place each element in its own cluster.

Coordination cost of the cluster matrix is calculated as indicated below:

Test whether any two elements are dependent. If yes, test if the two dependent elements are in the same cluster. If both elements are in the same cluster of index y, an IntraClusterCost is assigned:

$$\mathbf{IntraClusterCost} = \sum_{i=0}^{i=DSM\ size-1} \sum_{j=0}^{j\neq i\ =DSM\ size-1} \big(\mathrm{DSM}(\,\mathrm{j},\mathrm{k}) + \mathrm{DSM}(\mathrm{k},\mathrm{j})\big)\ (\mathrm{Cluster\ Size(y)})^{powcc}$$

If the elements are not dependent:

$$\mathbf{ExtraClusterCost} = \sum_{i=0}^{i=DSM\ size-1} \sum_{j=0}^{j\neq i\ =DSM\ size-1} \big(\mathrm{DSM}(\,\mathrm{j},\mathrm{k}) + \mathrm{DSM}(\mathrm{k},\mathrm{j})\big)\ (\mathrm{DSM\ size})^{powcc}$$

Where *powcc* is a cluster parameter assigned *a priori*.

2. Choose an element randomly.
3. Make bid calculation from all clusters for the selected element.

Check whether the elements existing in a certain cluster grouping have connection with the element bid. This is done by checking the dependency of the DSM matrix and assigning a cluster bid applying this formula:

$$\boldsymbol{ClusterBid\ (k)} = \sum_{k=0}^{k=cluster\ number-1} \frac{(inout)^{powdep}}{(ClusterSize\ of\ k)^{powbid}}$$

k = cluster number

ClusterBid(k) = Bid from cluster k for the chosen element

inout = sum of DSM interactions of the chosen element with each of the elements in cluster k

powdep = exponential to emphasize interactions

powbid = exponential to penalize size of the cluster

4. Choose a number between 1 and rand_bid (algorithm parameter) randomly.

5. Delete same clusters and then calculate the total coordination cost if the selected element becomes a member of the cluster with highest bid (use second highest bid if step 5 is equal to rand_bid)

6. Choose a number between 1 and rand_accept (algorithm parameter) randomly.

7. If the new coordination cost is lower than the old coordination cost or the number chosen in step 6 is equal to rand_accept, make the change permanent; otherwise, make no changes.

8. Go back to step 2 for n times. In case of overlapping elements in clusters, reorder the DSM before any calculations. Reordering adds the overlapping elements as if they were new elements.

After highlightting in CHAPTER 2the literature review of the DSM and its use in optimization techniques, we will discuss in CHAPTER 3 the heuristics and meta-heuristics search techniques.

CHAPTER 3

SEARCH TECHNIQUES AND HEURISTICS

3.1. Introduction

To solve optimization problems, various search methods are used. However, the size of the problem search space affects the method considered. It is not always feasible to obtain an optimal solution when dealing with NP-hard problems; this is because the large search space of the latter type problems is limited by time and computer memory capacity constraints.

This section provides a quick overview of the various search methods used in optimization problems taking into account the size of the problem search space. The exhaustive search method is a brute force technique applied to enumerate all possible candidates from which the best solution would be certainly found. The computation and time cost of this technique will increase as the search space increases. For this reason, this technique is normally used for small search space problems or when heuristics can be used to decrease the search space size according to specific problem criteria.

The branch and bound method is used to solve combinatorial and discrete global optimization of mainly medium size problems. This method is based on two parts as evident from its name. The first part (branch) is to divide the large problem into smaller ones. The second part (bound) is to test whether the optimal solution can be found in this branch. If not, the whole branch is discarded; otherwise, it should be saved. This algorithm is terminated when all smaller parts are tested. Yet, it yet will keep track of the best solution and neglect unimproved solutions.

The branch and bound algorithm, however, will not yield satisfactory results in case of large size problems (Clausen, 1999). As compromise between the time constraint and the computer memory capacity constraint, other algorithms are found to achieve good solutions instead of optimum solutions. Heuristics algorithms, such as branch and bound variants, problem specific heuristics, pure random search, and controlled random search, are efficient for these types of problems. Genetic algorithm (Chinneck, 2006), simulated annealing and tabu search (Glover, 1993) are considered as controlled random search methods.

In practical cases, a heuristic method is best used to generate good solutions at minor computational expense and within a specific amount of time. The currently available heuristics are classified as either constructive or improvement methods (Osman, 1989). A constructive heuristic is actually a building of a solution from the data by examining the characteristics of the problem to be solved. This construction is hard to figure out in different applications whereas improvement heuristic initially starts with a random solution and then endeavors to decrease the cost value of the objective function by allowing a series of local changes. The quality of the final solution depends on the starting solution and the rules used to generate neighboring solutions. A good approach, hence, is considered when an improvement method is applied to the result obtained from the constructive heuristic.

An improvement method starts initially with a random solution and then endeavors to decrease the cost value of the objective function by allowing a series of local changes. Descent method is an example of the improvement method. In a descent method, only combinations which decrease the value of the objective function are accepted. Yet, simulated annealing, which is a randomized improvement method, is used to accept solutions with a certain probability, even in case of no improvement in the objective function value. Descent method and simulated annealing will be discussed hereunder in detail.

3.2. Descent Method

A descent method is an improvement heuristic method that repeatedly endeavors to construct and improve a current solution starting initially from a current feasible solution. Such current solution might be randomly generated or be a result of using a constructive method that proved better results in less computational time (Osman, 1989).

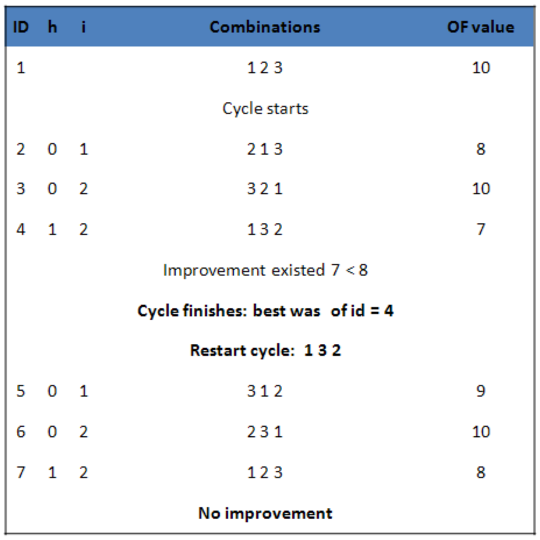
The main part of the method is to be able to define a neighborhood in order to give a new sequence or solution. Various researches tackle different neighborhood generators of which the interchange neighborhood approach is considered. If the current sequence is the following $(\sigma(1) \ldots\ldots \sigma(n))$, then a new sequence will be obtained by interchanging the positions h and i where h < i, of the σsequence $(\sigma(1) \ldots\ldots \sigma(h-1), \sigma(i), \sigma(h+1), \ldots, \sigma(i-1), \sigma(h), \sigma(i+1) \ldots\ldots \sigma(n))$. There are $n(n-1)/2$ neighbors possibilities for each sequence.

It is then necessarily to specify the order in which neighbors are searched. All possible values of h and i are considered in an ordered search after which the same cycle of values is repeated. Thus, the order of *(h,i)* values will be as follows: (1,2), (1, 3) . . . . . (1, n), (2, 3) . . . . . (n - 1, n). It is also possible to assign the values of *(h,i)* randomly.

Accordingly, the descent method starts by a starting sequence σ and then generates further solutions using a neighboring generator which is a σ' sequence. For each sequence, the objective function is evaluated, giving $C_{\max}(\sigma)$ and $C_{\max}(\sigma')$.

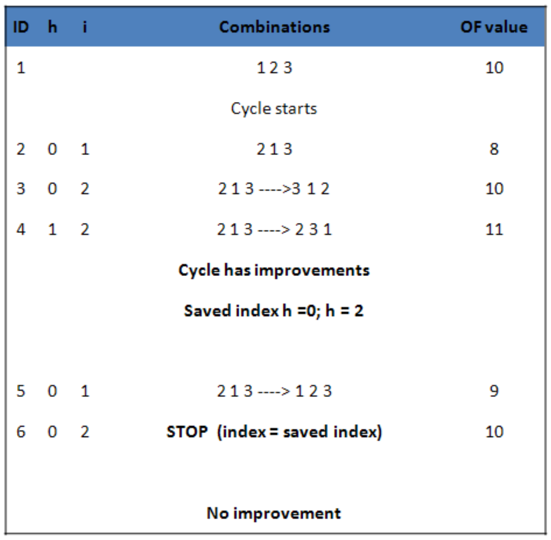
If $\Delta = C_{\max}(\sigma') - C_{\max}(\sigma) < 0$, then σ' is accepted as the current sequence. Yet, if $\Delta \geq 0$, then σ is retained as the current sequence. In both cases, the generation of new sequences is made and the process is repeated until all neighbors of the current sequence are searched without improving the objective value.

The descent method can follow either the first improvement (FI) or the best improvement (BI) method. These two approaches differ in the occasion where the best solution is saved. BI method searches all neighborhood in a cycle and then saves the combination with the best value, where as the FI method saves the first best combination obtained even though not all the cycle was tested. In Fig. 3.1 the initial example with the combination (1, 2, 3) and indexes (0, 1, 2) respectively is solved using the BI and FI methods.



ID	h	i	Combinations	OF value
1			1 2 3	10
			Cycle starts	
2	0	1	2 1 3	8
3	0	2	3 2 1	10
4	1	2	1 3 2	7
			Improvement existed 7 < 8	
			Cycle finishes: best was of id = 4	
			Restart cycle: 1 3 2	
5	0	1	3 1 2	9
6	0	2	2 3 1	10
7	1	2	1 2 3	8
			No improvement	

a) Best Improvement



ID	h	i	Combinations	OF value
1			1 2 3	10
			Cycle starts	
2	0	1	2 1 3	8
3	0	2	2 1 3 ---->3 1 2	10
4	1	2	2 1 3 ----> 2 3 1	11
			Cycle has improvements	
			Saved index h =0; h = 2	
5	0	1	2 1 3 ----> 1 2 3	9
6	0	2	**STOP (index = saved index)**	10
			No improvement	

b) First Improvement

Fig. 3.1. Descent method solved example

On one hand, although an improvement existed at ID = 2 stage, the BI method continues the cycle generation without any save for the best solution. As the cycle finishes (ID =4) the combination with the lowest solution is saved which is (1, 3, 2) in this case. A new cycle is restarted targeting a better combination with a better OF. If a cycle finishes without improvement the search stops. On the other hand, the FI method saves directly the combination with a lower OF value (ID =2) and does not wait for the cycle to end, but continues the indexing (h=0 and i =2) on the new saved combination (2, 1, 3) and not on the initial solution (1 2 3). If a cycle finishes with improvement, a new cycle is generated on the best solution saved with a reset for the indexing h and i.

3.3. Simulated Annealing

Simulated annealing (SA) was first mentioned by Nicholas Metropolis in 1953 as a solution for single or multi-objective, discrete or continuous, NP hard problems where the computational time increases exponentially (Johnson et al., 1989). As its name indicates, SA acts similarly to the physical heating process that involves heating the metal past its melting point and then cooling it according to a specific cooling rate. The cooling rate is preferred to be slow as quick cooling rate may lead to imperfections in the crystals formed.

SA was created mainly to avoid local optimum by accepting unimproved moves based on a probabilistic acceptance criterion. It starts with a current solution where the energy value is calculated and saved as the best solution attained at that time. A random or ordered neighboring solution is then generated as a potential to replace the current solution if it holds a lower objective function value. Otherwise, a certain acceptance probability function is used to accept this solution even though it holds no improvement. It is assumed that in early stages the probability of accepting unimproved solutions is high; as time passes, the temperature decreases and just improving solutions are accepted. According to the acceptance probability, the system will either move to a neighboring state or will remain in the same state. This iteration step will repeat until either a good enough steady state solution for the application is reached or a time factor is expired (Chinneck, 2006).

Neighboring states are found by applying different generating methods that vary from a random method to an ordered search method (shift process, interchange process, etc.). Special attention must be put on generating neighbors which is found to be the core of the solution result quality.

No general parameter functions may be applied to any kind of problems; each problem should define wisely its own SA parameters: the acceptance probability, the energy function, the candidate generator procedure, and the search space. An adequate definition of the cooling schedule guarantees the identification of near optimal solutions for many combinatorial problems (Osman, 1994).

3.3.1 Simulated Annealing Parameters

3.3.1.1 Acceptance Probability

The acceptance probability function P ($*$) is defined as = $e^{-\Delta/T}$ where Δ. Δ is the absolute value of the change between the new solution S_1 and the initial solution S_0 (Suman &Kumar, 2005); $T_{t+1} = r\ T_t$ where r is the cooling parameter which indicates the slope of decrease of the temperature T. Four parameters come along with the definition of the acceptance probability: starting temperature, final temperature, temperature decrement and a number of iterations to be performed at each temperature. The starting temperature should be assigned high so that the move to other neighbors, whether improved or unimproved, is allowed easily as if a random search is first applied. As the temperature cools, the acceptance of random solutions decreases. The starting temperature is assigned in different ways of which two are described below. If the maximum cost difference between two neighbors is predictable, then this difference will help assign the temperature. Sait and Youssef state that $T = - \Delta f_0 / \ln (X_0)$ where Δf_0 is the average increase in the objective function and X_0 is the ratio between the number of accepted moves and the number of attempted moves (Suman &Kumar, 2005). Another simple way of initializing T is to solve the formula $T = \Delta / (1-P)$ where P is chosen to be in range of 0.5 to 0.95 and Δ is the maximum difference between any two neighbors.

The final temperature is aimed to reach 0 theoretically. Yet, practically, this is not an obligation; it is enough to get close to zero or not higher than a certain low probability.

The temperature decrement function affects the success of the algorithm. In practice, either a simple linear method ($T = T-1$) or a geometric decrement method ($T = T\alpha$ where α could be an exponential or logarithmic expression normally < 1) (Suman & Kumar, 2005) is used. As α increases, the number of iteration increases.

The final parameter set is the number of iterations at each temperature. Theoretically, it is preferred to do as much iterations as possible on each temperature for the system to stabilize on that specific temperature. Large number of iterations should thus be performed and usually such number grows exponentially with the problem size. A trivial way is to set the number of iterations to 1 or to a specific constant. Another option is to vary the number of iterations with the change of temperature. This means that the number of iterations should increase as the temperature cools so that the local optimum can be entirely achieved.

3.3.1.2. Energy Function

The energy function is the cost value of a certain problem solution candidate. Such value is used to evaluate the difference between the current and the neighborhood solutions. It could be helpful in this case to have a threshold value where solutions can be neglected directly due to some problem constraints. Hard and soft constraints could be defined subject to the problem needs where different weights are assigned thus affecting the function cost (Johnson et al., 1989). Weights can be assigned dynamically as the process progresses. This leads to say that hard constraints can be violated at the beginning but not at the end.

3.3.1.3. Candidate Generator Procedure

An important criterion is how to move from one state to the other. Swapping, permutation or finding combinations are possible ways to generate neighbor candidates. Some results showed that the move should meet a symmetric criterion (Abdelsalm & Bao, 2006); if the move is from state 1 to state 2, for example, then there must be a way to move from state 2 to state 1.

3.3.1.4. Search Space

The smaller the search space is, the higher is the possibility to achieve optimal solutions. If the objective function definition accepts infeasible solutions, then the search space will increase accordingly. It is thus preferred to cut off large search spaces and keep the neighborhood as small as possible (Johnson et al., 1989). This will lead to a fast search but with low remarkable improvements.

3.3.2. Simulated Annealing Enhancement

While solving an optimization problem using the simulated annealing technique, contradictory interests are found. Such interests could be enhanced to improve the quality of the solution obtained in minimum amount of time. The contradiction found in simulated annealing could be reflected in the attempt to have simultaneously a quick, simple and adequate objective function in order to model the problem objective function or decrease the solution search space without restricting the search.

Given that simulated annealing may accept bad solutions, it is possible that the final solution might be worse than the best solution (Suman & Kumar, 2005). In this case, simulated annealing technique is merged with tabu search where the best results are saved to be set as the final solution in case the final solution was not the best. In addition, the candidate generator procedure can change along the algorithm progress and accordingly promise better solutions. The acceptance probability function can be replaced by a less expensive computational equation (Johnson et.al., 1989). This means that the exponential used is approximated by $P(\delta) = 1 - \delta/t$ thus enhancing the time of calculation without altering the quality of solution. Such approximation is proven to speed the calculation by 33% (Johnson et al., 1989).

It should be known, in sum, that simulated annealing is more an approach where parameters are set uniquely for specific problems rather than a generic algorithm to be followed. The best way to achieve optimal solutions is by long running simulated annealing, if feasible, instead of taking the best of time equivalent collector of smaller runs.

3.3.3. SA Cooling Schedule Models

The literature was able to define the SA parameters by considering each problem alone. Hence, identification of models to define the cooling schedule parameters took great effort especially when the theoretical annealing schedules could not guarantee convergence to near optimal solution in practical cases. The temperature reduction schemes are classified into three categories as shown below in a pictorial representation (Fig. 3.2) where the cooling schedule is based upon theoretical derivations and successful practical results.

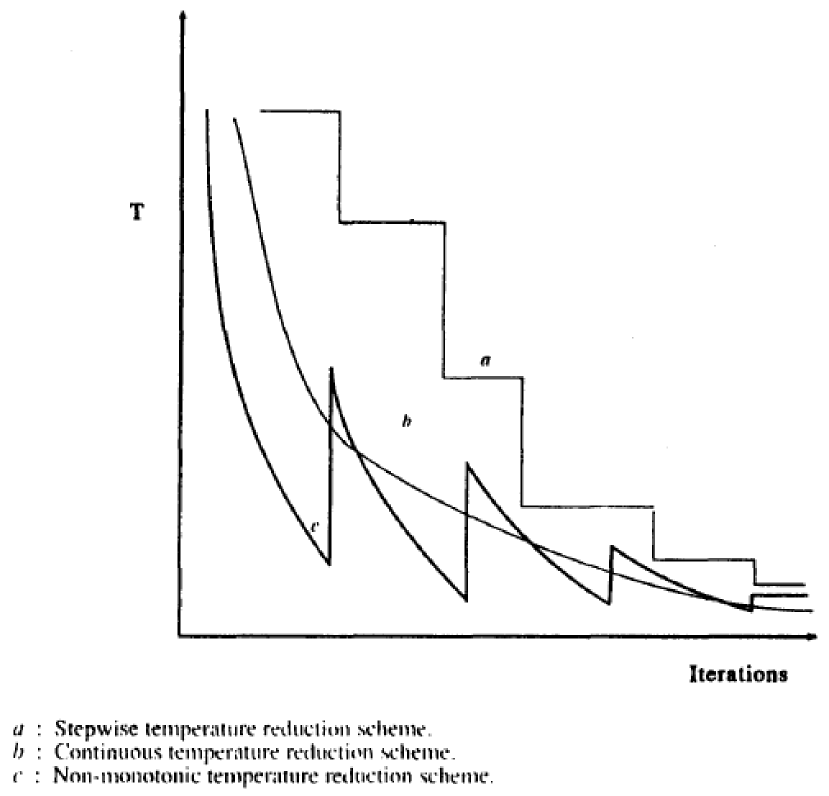


Fig. 3.2. Three temperature reduction schemes (Osman, 1994)

3.3.3.1. Stepwise Temperature Reduction Scheme

A Markov chain model is used in this category to specify the SA cooling schedule. Finite sequence of homogeneous Markov chains with finite length are generated at monotonic decreasing values of the temperature (Osman, 1994). A fixed or a predefined value is used to define the length L_k of the kth Markov chain. L_k can be assigned to the number of accepted moves (iterations) at the corresponding T_k value of the temperature. This simple cooling schedule matches the cooling schedule definition of Kirkpartick et al. (1983), Johnson et al. (1989), White (1984), Aarts & Van Laarhoven (1985), and Huang et al. (1986).

The initial temperature Ts is predefined by monitoring a separate run of m moves of the problem before the real optimization process starts. The equation below is used where m is the number of random moves, m+ is the number of cost increase, Δ^+ is the average cost increase over the moves, and χ is an acceptance ratio, $0 < \chi < 1$.

$$T_S = \Delta^+ \times \left(\ln \frac{m^+}{\chi \times m^+ - (1-\chi) \times (m - m^+)}\right)^{-1}$$

The decrement rule is determined by a small constant decrement value δ and the standard deviation σ_k of the objective function values generated at the kth Markov chain as follows:

$$T_{k+1} = T_k \times \left(1 + \frac{T_k \times \ln(1+\delta)}{3 \times \sigma_k}\right)^{-1}$$

The stopping criterion is so defined to depend on the difference between the average objective value at the kth Markov chain and the average of objective value of the optimal solution. The search terminates when this value is determined as small ε .

3.3.3.2. Continuous Temperature Reduction Schemes

This category includes the work on the cooling schedules by Hajek (1988) and Lundy & Mees (1986) which specifies a finite sequence of inhomogeneous Markov chains. The $\mathbf{L}_k$ in this case is equal to one iteration where the temperature decrement rule is applied to each iteration using the equation below (T_s is set to a value so that $T_s \gg U$ where U is an upper bound on Δ_{max}):

$$T_{k+1} = \frac{T_k}{(1+\beta \times T_k)}$$

β is defined as shown for a given T_s.T_f if the total number of iterations to terminate the search is predefined:

$$\beta = \frac{Ts-Tf}{M \times Ts \times Tf}$$

3.3.3.3. Non-monotonic Temperature Reduction Scheme

This category initially includes the work of Conolly (1990; 1992) and a new cooling schedule established by Osman and Christofides (1989). The philosophy of this category is not just decreasing the temperature along the way, but occasionally increasing or resetting the temperature whenever a special cycle is found. A special cycle is determined when a complete search of the neighborhood for an improved solution is unachievable. Conolly introduced the philosophy of this category by decrementing the temperature until a specified number of rejected moves occurs. At this stage, T is set to T_{found} where the best solution was found, β is set to zero and the search completes the rest of the iterations using T_{found}.

Osman and Christofides new cooling schedule came as an attempt to solve issues found in the previous annealing schemes and as a generalization of Connolly's simulated annealing scheme. This schedule solved the trap of having an optimal solution existing in some neighborhoods as the previous methods generate initially random solutions and may miss the optimal solution or take a longer time to achieve it. Moreover, a waste of time occurs as the value of the temperature is initially high and finally low. This is because the high temperature leads to a high acceptance of bad solutions thus destroying good initial ones, and the final low temperature neglects worse solutions and accepts improved ones. The core of our work is to decrease the time wasted at the beginning and the end of the search and increase it in between. The cooling schedule will then vary and will use an updated value of T and β at each iteration.

The cooling schedule parameters are illustrated below as follows:

- *The initial and final temperature values* are to be set to the maximum $\boldsymbol{\Delta_{max}}$ and to the minimum $\boldsymbol{\Delta_{min}}$ differences in the objective function values respectively. The initial value is set to a small value in the event the initial solution is considered a good heuristic start to avoid waste of time in the early stages.

- *The decrement rule* applies after each iteration, *k* where the temperature and the parameter β_k are updated according to the following equation:

$$T_{k+1} = \frac{T_k}{(1+\beta_k \times T_k)}$$

As *k* increases, β_k decreases and the temperature is then decreased more slowly with ***k*.**

$$\beta_k = \frac{Ts - Tf}{(\alpha + \gamma \times \sqrt{k}) \times Ts \times Tf}$$

In the above equation, α and γ are constants and determined experimentally in terms of problem characteristics. If $\gamma = 0$, then β_k is a constant independent of **k,** and T and β will have the same form of the continuous temperature reduction schemes.

- *Occasional temperature increase* occurs whenever a cycle is determined. The increase should not be very high to escape from the local minimum and should not deviate much as when a total new random sequence is restarting.

The temperature value is modified as follows:

1. Initially: $T_{reset} = Ts$
2. If a cycle is detected:
   2.1 $\mathbf{T_{reset} = T_{reset}/2}$
   2.2 Test if: $\mathbf{T_{reset} > T_k}$**,**
   Yes: $\mathbf{T_{k+1} = T_{reset}}$
   No: $\mathbf{T_{k+1} = T_{found}}$

After this reset, the decrement and the occasional reset rules are used until the algorithm stops.

- *The stopping criterion* must be a controlled parameter to be a trade-off between the quality of solution and the computation time. The algorithm stops either when the algorithm runs for predefined number of iterations or after a predefined number of temperature resets ***R*** is reached without solution improvement.

The pseudocode of the new cooling schedule is shown in Table 3.1.

Table. 3.1. New cooling schedule pseudocode

```
Step 1: 1.1 Assign values for Tinitial, Tfinal, and the decrement value;  change flag set to
            true, # of reheat times = 2 and  # of max consecutive unchanged cycle = 2.
        1.2 A random solution of process is generated; each elements of the product and
        team domain is put in a separate cluster. This combination is named X.
        1.3 The OF value of the above initialized solution is calculated and saved as the
 best saved solution.

Step 2: While stopping criteria are not achieved, (T > Tfinal or # of reheat times ≠ max # of
        reheat times)
       do
        2.1 If change flag is false
          Yes:
             2.1.a #of consecutive unchanged cycle++
            2.1.b If #of consecutive unchanged cycle reached max # of consecutive
 unchanged cycle
                         Yes:
                            2.1.b.1 Reheat the value of temperature
                                        1. Initially: $T_{reset}$ = Ts
                                        2. If a cycle is detected:
                                           2.1 $T_{reset} = T_{reset}/2$
                                           2.2 Test if: $T_{reset} > T_k$,
                                               Yes: $T_{k+1} = T_{reset}$
                                               No:  $T_{k+1} = T_{found}$
                            2.1.b.2 Increase # of reheat times
                         No: Continue loop
          No: Continue loop
        2.1.1 Perform the following loop for each of the three domains and reset change
 flag.
             2.1.1.1 Applying the interchange swap to the three domains, three
        neighboring solutions (X') are executed for each iteration.
             2.1.1.2 Compute $\Delta = f(X') - f(X)$
             2.1.1.3 If $\Delta < 0$ or worse solution was acceptable, then the combination is
             saved as the best attribute and change flag = true. (X = X')
             2.1.1.4 Temperature decrement by factor of α between 0.5 and 1.
 2.1.2 Back to step 2

 Step 3: Return X.
```

After discussing the search techniques and heuristics in CHAPTER 3, we will handle in CHAPTER 4 below the optimization of the three domains simultaneously.

CHAPTER 4

OPTIMIZING THE THREE DOMAINS SIMULTANEOUSLY

4.1. Convention of DSM Annotation

During product development, managers are faced with the following problems/queries: What is the best way to organize individual resources into teams? What is the best way to combine product elements/components into modules? In which order the various development tasks should be executed?

Before tackling these questions, the notions adopted while representing each of the domains need to be agreed upon. In the team domain, capital letters are used to represent a member in the team. The analysis technique used to optimize this domain is the clustering algorithm. The clusters obtained represent teams in an organization. In the process domain, numbers are used to represent tasks to be executed. The analysis technique used to optimize this domain is the partitioning approach to minimize feedbacks which in turn reduces rework. In the product domain, small letters are used to represent physical elements or components. The analysis technique used to optimize this domain is also the clustering algorithm. The clusters obtained represent product modules.

The network of the three domains is defined in the beginning where DSM/MDM of the following is built. People/people DSM indicating the relations among people in a team. Who works with whom? Who addresses whom? Who sits next to whom? Task/task DSM indicates the relations of the tasks. What are the tasks available in order to execute a certain task? For which tasks is the output of a task an input? The last DSM is the component /component DSM where the physical relation among the components is entered. Which component is physically located next to other components? The DMM built are process/team to indicate who are the people that are responsible of which tasks, and the component/task structure is also built to identify the tasks that are executed in the design and development of the components.

4.2. Optimizing the Process Design Structure Matrix (DSM)

In the design structure matrix, feedbacks between tasks are presented by marks above the diagonal. If task n and task m are in feedback, then task m must wait for task n accomplishment to start its work. Because the feedback represents inefficiency in the PD process, the task dependencies must be arranged under the diagonal to reach an optimized solution.

To start with any process, DSM is analyzed as follows. The DSM in Fig. 4.1 shows the sequence of the process yet without any relations between the tasks. All the tasks can be processed in parallel. The marks shown in Fig. 4.2 indicate the relations dependency between the tasks. Task 1 cannot be accomplished without the completion of task 3 because the result of task 3 may result in a rework of task 1. Whereas the dependency marks between task 2 and task 1 are considered smooth given that task 2 starts when task 1 finishes without any possibility of rework.

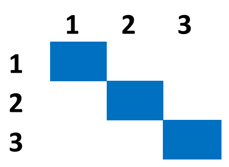


Fig. 4.1. Process DSM without marks

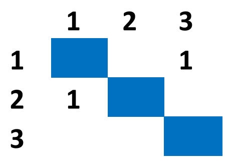


Fig. 4.2. Process DSM with marks

Therefore, the rule is to minimize the marks above the diagonal. A penalty factor is applied to differentiate the feedback marks according to their distance from the diagonal. The penalty equation is elaborated below applying the above DSM in Fig. 4.2.

Given the fact that this example is of a small size, enumerating all the possibilities exhaustively is appropriate; (DSM size)! = 3! = 6 possible distinct arrangements are thus available. The DSM matrix resulted from the above ordering are shown below in Fig. 4.3 where n demonstrates the number of marks above the diagonal and penalty is calculated using the below formula:

$$\textit{Process Penalty} = \sum_{n=0}^{n=\#\ of\ feedback\ marks} \textit{Column Index} - \textit{Row Index}$$

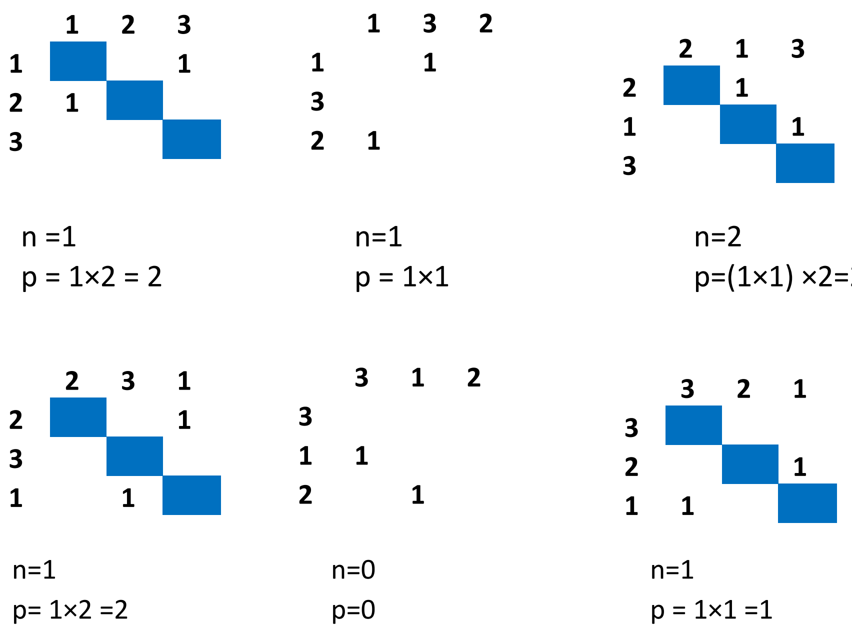


Fig. 4.3. Result of all possible solutions of a Process Matrix

The analysis of the six possible solutions shows that the sequence of the fifth case (3, 1, 2) is the best solution where no above diagonal marks exist. The task process goes smoothly. Comparing the second solution with the last solution and assuming that the time for each task accomplishment is the same, we find that the penalty value for both is equal because the time loss of 1 waiting 3 is the same as 2 waiting 1.

As the existence of feedback marks hold negative effect on the process domain, the aim of the objective function is to decrease the number of feedback marks. Hence, a simple number of the feedback marks will be considered instead of the penalty factor calculation.

4.3. Optimizing the Team and Product Design Structure Matrix (DSM)

The crucial work of managers is to find appropriate ways to organize people and assign them work over time by enabling communication and synchronization actions. Team coordination and formation is indeed a crucial activity for any organization. How to package products into modules is another issue the manager should think of. The aim is to find all combinations of clusters that could be generated and choose the best DSM arrangement.

Let us first calculate the number of possible combinations. How many clustering arrangements are possible? Consider a 3×3 DSM i.e. a matrix with 3 elements: A, B, and C. A listing of 8 distinct combinations is shown in Fig. 4.4.

Overlapping clusters in teams is justified since it is possible for a person to belong to multiple teams. This overlap, however, should not exceed a certain limit due to cognitive and time capacity limitations of a person. Yet, in the product domain, clusters are defined by the physical existence of their product parts; it is thus impossible to have products existing physically in two different cluster modules. Symmetrical information is justified and is a must in both the team and the product domains. If a member is in contact with another team member, this relation is shared and hence the relation is in vice versa. Similarly with the product domain, if a component ‘a’ is physically related to another component ‘b’, then the inverse also holds true.

	The options of ABC		
1	A	B	C
2	ABC		
3	A	BC	
4	B	AC	
5	C	AB	
6	AB	AC	
7	AB	BC	
8	AC	BC	
8 distinct cases			

Fig. 4.4. Distinct cluster combinations of a 3×3 DSM

The different numbers of combinations can be obtained by using the Stirling number of the second kind denoted by S(n, k), which is the number of ways to partition n distinct objects into k nonempty subsets (Mohr, 2009). The numbers presented here were calculated using the following well-known recurrence: $S(n, k) = S(n - 1, k - 1) + k * S(n - 1, k)$. Applying the formula below, the following possibilities are shown without allowing for overlapping.

$$S(n,k) = \frac{1}{k!}\sum_{i=0}^{k}(-1)^i\binom{k}{i}(k-i)^n$$

DSM size	3	4	5	6	7	8	....	30
# of all possible combinations	5	15	52	203	877	4140	...	~ 10^30 (cannot be calculated)

Fig. 4.5. Number of possible combinations

Given that it is common in an enterprise to have more than 30 employees or products, the calculation made in Fig.4.5 proves the impossible use of the exhaustive method and the need for a meta-heuristic search method instead. With overlapping, the number of possible combinations will increase.

The objective is to find the cluster combination with the lowest cost. In prior research, Thebeau (2001) considered that the cost of interactions for outside clusters is higher than the cost of interactions inside the clusters. In this thesis, the interactions of outside clusters and the missing interactions in a structured team are both penalized. Having two persons in different teams who must communicate together is as worse as having members of the same team not communicating together. This means that we shall not have scattered marks, and, as a second level, we shall not have teams or modules with elements not connected or related.

The cost is then divided into outside cost and inside cost. Since we have assumed a symmetrical DSM, the upper part of the DSM is just treated in the calculations. The total cost is the sum of the number of scattered interfaces multiplied by the DSM size and the number of marks multiplied by the cluster size. The outside cost equals the outside number of feedback marks multiplied by DSM size; the inside cost equals the unavailable marks in a cluster multiplied by its size.

Symmetrical matrices are used for the team and product domains where overlapping elements are discarded for the sake of simplicity.

4.4. Relational Rules among the Three Domains

After having elaborated on the meaning of the cost of each domain in addition to the way we used to calculate such cost, and in view of our studying the three domains simultaneously, we present the model below to illustrate the rules connecting the three domains together. The relational rules deduced from the domains will in turn improve the objective function and thus provide more precise optimal solutions. In the process domain, the persons in charge of each of the processes along with the tasks performed on each of the products are entered. The relations between the three domains are illustrated in Fig. 4.6 hereunder as an anticlockwise navigation.

We note that the navigation among the domains can take different forms as declared by Tyson Browning (2001) who indicates the dual direction among the three domains. The below arguments defend our choice in our thesis. The structure of the organization is related to the structure of the development process where process with coupled activities requires integrated executable team. Furthermore, the product architecture can influence the team structure given the fact that the team is in charge of building the organizational products. In addition, a relation exists between the process and the product domain where the legacy development process overly constrains the design of unprecedented products.

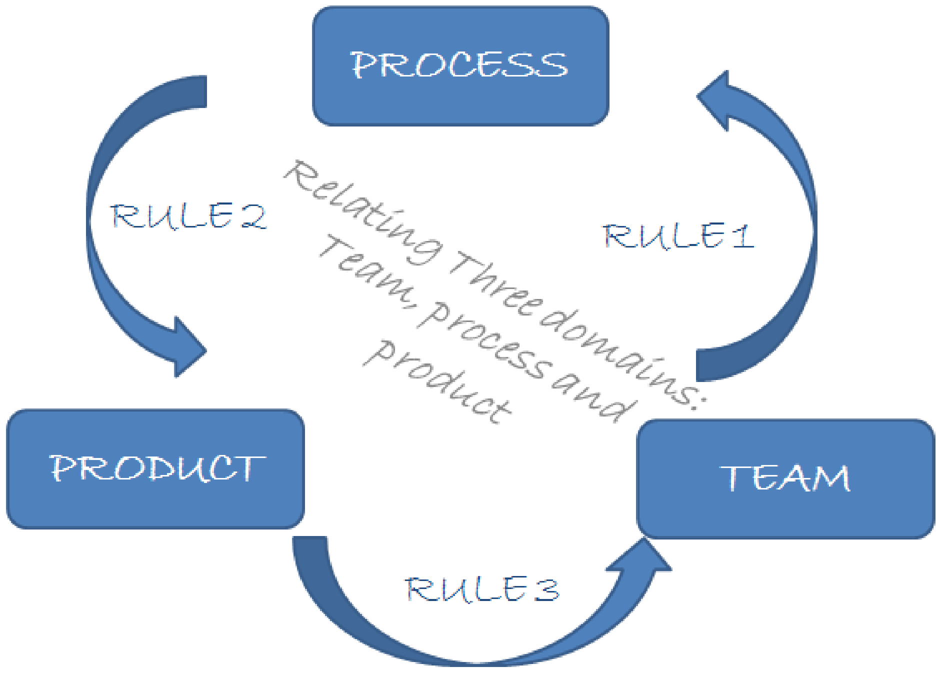


Fig. 4.6. Rules relating the three domains

4.4.1. The Effect of the Team Domain on the Process Domain (Rule 1)

If a feedback exists between two different tasks where the latter is performed by the same resource or by resources that belong to the same team, then the feedback penalty must be reduced or removed completely. This penalty is reduced as the rework is done with less time because the resource (being the same or being in the same team) will have ease of communication and understanding of the problem faced. The feedback loop will be also done with less time. Nevertheless, if the persons involved are in different teams, then the rework due to feedback mark will consume further time to be noticed, understood and executed. This means that problem solving is easily done when individuals have a face-to-face or a direct contact (Braha, 2002).

Such feedback mark will be therefore maintained in the calculation to determine its negative effect on the system. This rule is defended in the literature. Tyson Browning (2001) states that "[w]hen a process contains coupled activities, the organizational teams with responsibility for executing those activities require integration." In addition, Braha (2002) mentions that tasks that are strongly related must be assigned to the same team for an effective overall cycle. People linked together must work on the same activities. (Lindemann, et al., 2009)

4.4.2. The Effect of the Process Domain on the Product Domain (Rule 2)

If an interface (i.e. a dependency between two components from different modules) exists between two components, and the tasks corresponding to these components are sequential (i.e. not involved in feedback), then the module interface penalty must be reduced or removed completely. The components on which feedback tasks are executed must be put together to blockade the rework in one module. Tang et al., (2009) consider that if a component is changed, the effect would be propagated to other modules. Hence the aim of this rule is to reduce this effect and limit it to one module. For example, if an interface joins component 'a' and component 'b' (each being in different modules), and the process corresponding to each of the components is sequential, then this interface will be diluted because the said components are not related by the task executed on them.

4.4.3. ***The Effect of the Product Domain on the Team Domain (Rule 3)***

If two or more persons work on common product clusters or work on one or more modules in common, then the interface penalty between these persons/teams is maintained and its existence is justified. For example, if person A communicates with person B without being in the same team, we test whether these two persons work on same modules. If they do, then the interface is maintained and justified because persons working on the same products need to communicate and thus be in the same team. If, however, they work on different modules, the interface mark must be diluted given that there will be no reason for these two persons to communicate as long as they are not working on the same products. Lindemann, et al., (2009) defend this rule stating that people must cooperate in the same team if they are working on the same interrelated components.

Traditionally, the three domains were treated separately and individually where a local optimization exists for each domain alone regardless of the others. Our aim is to formulate a global optimal solution for the product development organizational problem.

4.5. Overall Objective Function Calculation

The objective function of the Product development organizational product is to minimize the sum of the team, product and process cost. The three relational rules relating the domains together are used in the OF calculation.

Overall Objective Function = $\sum$ Cost of domains……………………….. (eq.4.1)

The following notations are used

$N \times N$ = DSM size
D = # of DSM marks of the process domain excluding the diagonal marks
F = total # of feedback marks
F' = Adjusted number of feedback marks due to the dilution effect of rule #1. The dilution factor depends on a ratio between 0 and 1. This ratio is multiplied by the adjusted feedback marks. So, a ratio of 0 represents a complete dilution of the mark where as a ratio of 1 represents no impact of the rule.
Ci = Cluster size of cluster i where $1 \leq i \leq C$

di = # of marks in cluster i. Note that this number represents half of the matrix because the product and team matrix considered contains symmetrical data.
I = # of interface marks between clusters (outside cluster boundaries)
I' = Adjusted interface marks according to Rule 2(for product matrices) or Rule 3(for team matrices). Note that the dilution ratio discussed above is applied in the above mentioned two cases.

The cost value of the process domain is an interpretation of the existing feedback marks. As mentioned earlier, optimizing the process domain is a simple minimization of the number of feedbacks between the tasks. This is because the feedback between tasks leads to rework, and an extended time to accomplish the overall task is required accordingly. Yet, if a feedback exists between two members in the same team, then a dilution of the feedback mark is considered because the repetitive work is done easily among the same team members. The process cost is illustrated in equation (4.2) below:

$$\textbf{Process Cost} = \frac{F'}{D} \qquad \text{(eq.4.2)}$$

If a feedback mark exists between two tasks executed by members in the same team, then this feedback mark is diluted (Rule 1). This means that the value of justified feedback marks $F'$is equal to the number of feedback marks decreased by the number of marks diluted. In other words, if all the existing feedback marks are justified, i.e. executed by members in different teams, then the process domain cost will be 100%.

In optimizing the product domain cost, we aim to gather all the components dependencies in cluster modules and decrease the interface scattered marks existing among clusters. Each component must be assigned to exactly one module cluster. Less scattered marks lead to better results as long as the modules formulated contain related components. The interrelation between the process and the product affects the product cost in whether or not the interface between components should be diluted (Rule 2). If an interface exists between two components where the tasks performed on these two components are in feedback, then the existing interface is justified. In the event the tasks are in sequence, then the interface mark should be diluted. This applies similarly to the team domain. Yet, in this case, the impact of the product on the team domain is considered (Rule 3). If an interface exists between two members (not in the same team) working on different module components, then the interface is diluted. In the event the unrelated team members work on the same module, then the interface mark should be justified.

Therefore, the target is to minimize scattered marks and maximize the inside connections. This is illustrated best in the equation 4.3 below. The maximum possible cost existing in a certain combination is formulated by equation 4.4 and the current cost is illustrated in equation 4.5. This calculation targets half the matrix since the product and the team domains should be symmetrical matrices.

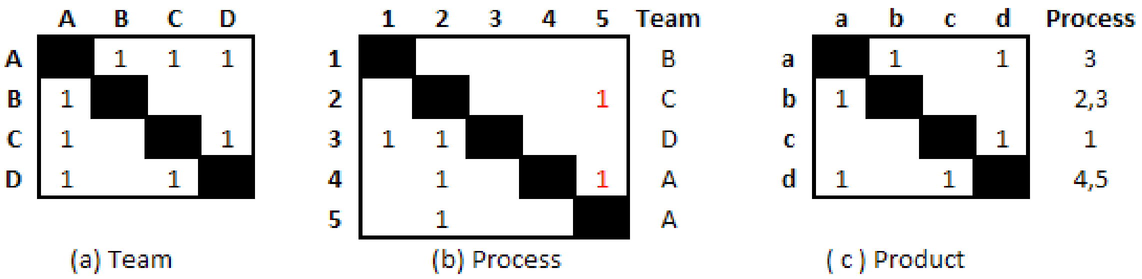
Product Cost or Team Cost $= \frac{current\ cost}{maximum\ cost}$ ………………….. (eq. 4.3)

maximum cost = $\sum_{i=1}^{c} \frac{Ci\,(Ci-1)}{2}\ Ci + \left(\frac{N(N-1)}{2} - \sum_{i=1}^{c} \frac{Ci\,(Ci-1)}{2}\right) \times N$… (eq. 4.4)

current cost = $\sum_{i=1}^{c} \{ \frac{Ci\,(Ci-1)}{2} - di\} Ci + I'N$……………………………. (eq. 4.5)

The number of justified interfaces I' equals the number of interfaces that excludes the interfaces diluted due to the effect of the process on the product domain or the effect of the product on the team domain. Given that the cost in each domain is illustrated as a percentage value, the objective function cost ranges between 0 and 300. The importance of normalizing each of the domain cost using a percentage representation for each domain is justified since it is the sum of the process, product and team cost of the three domains together. Hence, we cannot add the cost of the three different domains that are generated from different meaning and calculation. A cost of value 50 in the process domain has a different meaning in the product or process domain. Moreover, if the size of one of the domains was larger compared to the other domains, then the calculation of the other domains will be negligible. Hence, the cost of the larger size domain will be leading the calculation diminishing the cost effect of the other two domains.

The below three domains of Fig. 4.7 illustrate the team, process and product domain respectively. The "1" mark reflects the connection among the elements of the above DSM. The calculation of the OF for the following combination; team [A][B[C][D]; process [1 2 3 4 5]; and product [a][b][c][d], is shown below.



	A	B	C	D
A	■	1	1	1
B	1	■		
C	1		■	1
D	1		1	■

(a) Team

	1	2	3	4	5	Team
1	■					B
2		■			1	C
3	1	1	■			D
4		1		■	1	A
5		1			■	A

(b) Process

	a	b	c	d	Process
a	■	1		1	3
b	1	■			2,3
c			■	1	1
d	1		1	■	4,5

(c) Product

Fig. 4.7. Three domain example

As per equation 4.2, the process cost equals 1/6, i.e. 16.667%. The number of justified feedback is equal to the number of feedback decreased by one (2 - 1). The feedback mark existing between task 4 and task 5 is diluted since it is executed by the same resource A, but the feedback mark existing between task 2 and task 5 is counted since the resources in charge of these tasks are not in the same team.

As per equation 4.4 on the team domain, the maximum cost is 24 = 0 + 6×4. The current cost as per equation 4.5 is calculated by referring to the team/product DMM in Fig.4.8. The interfaces existing between "A" and "B, C, D" are diluted because they do not work on the same component modules. As an example, Fig.4.8 shows that person “A” works on component “d” and person “B” works on component “c” where components “c” and “d” are not in the same module. Therefore, the interface between “A” and “B” shown in Fig. 4.7.a is diluted according to Rule 3.

$$\begin{array}{c|cccc|}
 & a & b & c & d \\
\hline
A & & & & 1 \\
B & & & 1 & \\
C & & 1 & & \\
D & 1 & 1 & & \\
\hline
\end{array}$$

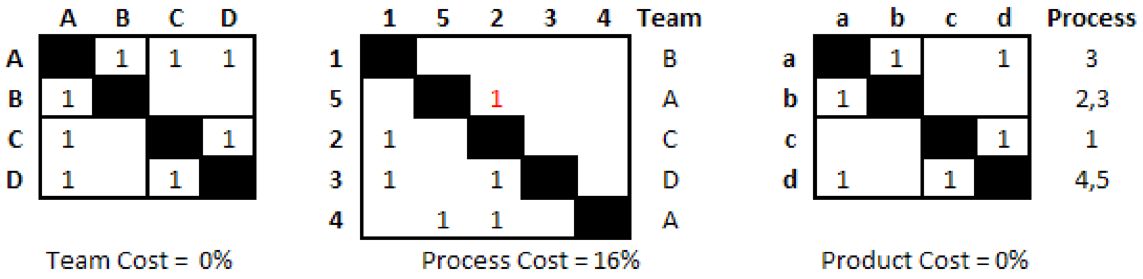
Fig. 4.8. Team/product DMM

However, the interface existing between “C” and “D” is not diluted since they both work on same component “b” as evident in Fig.4.8. The current cost is equal to 0 + 1×4. The team cost as per equation 4.5 is 4/24 = 16.667%.

Similar calculations are applied for the product cost calculation. As per equation 4.4, the maximum cost is 0 + 6 ×4 = 24. The current cost reflects the existing of scattered marks and nonrelated elements in a cluster. In this case, there exist just scattered interface marks. I' is subjected to Rule#2. The interface between “a” and “b” is diluted since the tasks performed on these components are not involved in feedback. Dilution also exists between “a” and “d” since task 3 does not involve a feedback with tasks 4 and/ or 5.Similarly for the interface between “c” and “d”. The product domain equals 0 /24, i.e. 0%

As per equation 4.1, the overall cost equals 16.667 + 16.667 + 0, i.e. 33.334%

The problem is then solved either by optimization in isolation or by simultaneous optimization of the three domains together. The solution of optimizing in isolation is shown in Fig.4.9.



	A	B	C	D
A	■	1	1	1
B	1	■		
C	1		■	1
D	1		1	■

	1	5	2	3	4	Team
1	■					B
5		■	1			A
2	1		■			C
3	1		1	■		D
4		1	1		■	A

	a	b	c	d	Process
a	■	1		1	3
b	1	■			2,3
c			■	1	1
d	1		1	■	4,5

Team Cost = 0%
Process Cost = 16%
Product Cost = 0%

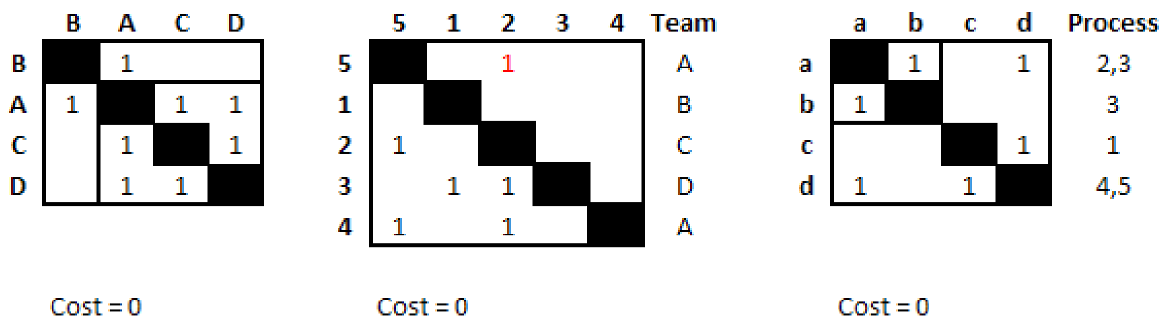
Fig. 4.9. Solution of optimization in isolation

In this solution, it can be shown that in the team domain, member "A" and "B" are in the same team, similarly for "C" and "D". However, it is true that there are two interfaces marks existing in the team domain but their negative effect is diluted the fact that "A" and "C" as well as "A" and "D" work on different modules. This is shown on Fig 4.8. Hence, the importance of these members to be in the same team is not recommended anymore due to the effect of the product structure. This result with a team cost of 0.

In the process domain, the two feedback marks were decreased to one feedback mark with the following order of sequence: 1, 5, 2, 3, 4. This feedback mark is not diluted the fact that "A" and "C" are different teams. This means that the rework requires more time to be realized and the cost is 16%.

In the product domain, component "a" and "b" are in the same module; similarly, for components "c" and "d". The interface mark existing between "a" and "d" is diluted the fact that the processes in charge are sequential hence, there is no rework on the module and these two parts could be in separate modules. This results with a product cost of 0.

Is there another organizational structure that might lead to a lower cost? The answer is positive and it is the solution of the simultaneous optimization shown in Fig. 4.10.



	B	A	C	D
B	■	1		
A	1	■	1	1
C		1	■	1
D		1	1	■

Cost = 0

	5	1	2	3	4	Team
5	■		1			A
1		■				B
2	1		■			C
3		1	1	■		D
4	1		1		■	A

Cost = 0

	a	b	c	d	Process
a	■	1		1	2,3
b	1	■			3
c			■	1	1
d	1		1	■	4,5

Cost = 0

Fig. 4.10. Solution of simultaneous optimization

In simultaneous optimization, a zero cost structure of the three domains is obtained. In the team domain all members are in the same team except for member “B”. The exiting interface between “B” and “A” is diluted the fact that these two members work on totally different tasks and modules. Thus, there is no need to have them in the same team, knowing that it is preferable to have team members working on related tasks or same modules.

In the process domain, one feedback mark exists but the rework effect is diluted the fact that the rework occurs within the same team members (“A” and “C”). However, the fact that there is a coupled dependency between task 2 and task 5, this means that there is no possible order with zero feedback.

In the product domain, a different distribution of components is structured where “c” and “d” are preferred to be in separate modules. Where the effect of the interfaces existing is diluted the fact that the processes in charge are sequential. This means that there is no need to have these components in the same module.

After handling in CHAPTER 4 the formulation of optimizing the three domains simultaneously, we will display in CHAPTER 5 the implementation of the various search techniques.

CHAPTER 5

IMPLEMENTING THE SEARCH TECHNIQUES

5.1. Search Technique Used in Simultaneously Optimizing the Three Domains

Optimizing the product, process, and team domains simultaneously is not a trivial matter, as all possible combinations must be enumerated and evaluated (i.e. exhaustive search) to guarantee getting the optimal solution. Computing all possible combinations is computationally prohibitive because it is restricted by time and computer memory. In this chapter, we propose a hybrid algorithm composed of heuristic and meta-heuristic techniques will improve the search technique and achieve, in less time, good solutions, if not optimal ones.

The best improvement and first improvement of the descent approach are used as a heuristic technique for one or for all of the three domains simultaneously. Moreover, two different cooling schedules of the meta-heuristic SA technique are used for all the three domains simultaneously. Note however, that SA is used in this paper instead of other meta-heuristic techniques such as GA or Tabu search which did well in some areas, since this thesis includes a sequencing problem that SA showed successful results for many sequencing problems. Sample tests will be performed with and without simulated annealing in order to test the importance of using SA in retrieving the optimal solution without being trapped in local optimum. The cooling schedules of the simulated annealing adopted in our testing are discussed in detail. The first cooling schedule is that described by Osman and Christofides (1989) and known as the "*New Cooling Schedule*" (Osman, 1994).

The second suggested schedule is an extension for the *New Cooling Schedule* where the idea of increasing the temperature is based on the fact that there are no benefits out of decreasing temperature further as a cycle occurs. The model shown in Fig 5.1 is a suggested modified model of Osman and Christofides where the temperature is updated at the end of each cycle and not at every iteration. The term used to describe the new suggested model is "*Modified Sequence Chain Length of the New Cooling Schedule.*" The factor of update of temperature will stay the same as implemented in the *New Cooling Schedule.* As a cycle is done without improvement, the temperature value is reheated according to the following rule. If the temperature value, as the cycle finishes, is greater than half the initial temperature, the temperature is reset to a value where the latest improvement is recorded. However, if the temperature value at the end of the cycle is less than half of the temperature, the temperature is reheated to half the initial temperature. The temperature reheating technique will ameliorate the search process and will stop the unnecessary temperature decrease when there is no improvement in a certain cycle. Reheating is allowed twice, and as two consecutive cycles hold no improvement the search is stopped.

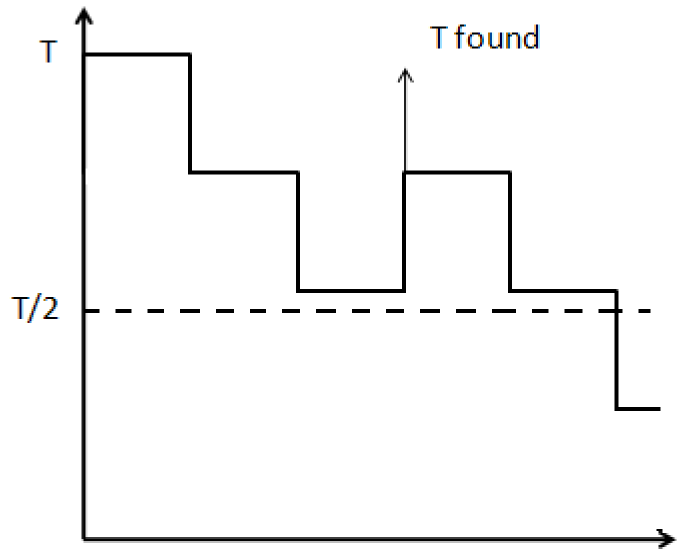


Fig. 5.1. Modified sequence of the Osman and Christofides cooling schedule

The pseudocode of the “Modified New New Cooling Schedule” (Table 5.1) is similar to that of the New Cooling Schedule with the difference on the occurrence of the update temperature.

Table. 5.1. Modified New Cooling Schedule Model

```
Step 1: 1.1 Assign values for T*initial*, T*final*, decrement value, Change flag set to true, # of reheat
            times = 2 and # of max consecutive unchanged cycle =2.
        1.2 A random solution of process is generated, each elements of the product and team
  domain is put in a separate cluster. This combination is named X.
        1.3 The OF value of the above initialized solution is calculated and saved as the best saved
  solution.
Step 2: While stopping criteria are not achieved, (T > Tfinal or # of no change consecutive cycle ≠ #
        of max allowable no change consecutive cycle)
        Do
        2.1 If no improvement is recorded in a full cycle
          Yes:
            2.1.a If maximum number of reheating times is reached, stop step2.
                  Else: Increment the number of consecutive unchanged cycle attribute
                        Reheat the value of temperature
                              1. Initially: $T_{reset} = Ts$
                              2. A cycle is detected:
                                    $\mathbf{T_{reset} = T_{reset}/2}$
                                    Test if: $\mathbf{T_{reset} > T_k}$,
                                      Yes: $\mathbf{T_{k+1} = T_{reset}}$
                                      No: $\mathbf{T_{k+1} = T_{found}}$
                        Increment number of reheat time occurrence
                  No: Go to step 2.1.1
          No: Go to step 2.1.1
        2.1.1 Perform the loop for each of the three domains and set Change flag to false initially
              2.1.1.1 Using the interchange swap on the three domains, three neighboring
        solutions (X') is executed for each iteration
              2.1.1.2 Compute $\Delta = f(\mathbf{X}') - f(\mathbf{X})$
              2.1.1.3 If $\Delta < 0$ or worse solution was accepted, then the combination is saved as
              the best attribute and change flag = true. (X = X')
              2.1.1.4 Back to 2.1.1 till the loop finishes.
  2.1.2 Temperature decrement by factor of $\alpha = 0.85$.
        Back to step 2
  Step 3: Return X.
```

While applying one of the SA cooling schedules, the parameters of the SA are calculated in the following way:

- The variables of the probability function $P = 1 - \Delta / T$ are assigned in the following way. Δ is achieved by getting the average of the OF value by running a separate run of the problem for a 20,000 number of iterations. The separate run uses the best improvement descent approach of all the three domains simultaneously. With an assumption of probability 0.85 and the value of Δ calculated, the T value is then obtained. Given the fact that Δ is in range of 0 to 300 for all size problems, the temperature is then multiplied by the sizes of all three domains. In this way, the temperature shows the problems characteristics.

- Number of reheating times = 2
- Number of unchanged consecutive cycles = 2
- Final temperature must be greater than 1
- The neighborhood generation is executed using the swapping method as shown below:

Neighboring Swap: The cycle size of (n)(n-1)/2 ways where n is the size of the domain.

1) In the process domain: Select a neighboring $S_q' \in N(S_q) = \{ S_q'$ by exchanging i and j positions and looping on the indexes i and j where i < j and i: 1 till size -1 and j = i +1 till size$\}$.

   Let us consider 1, 2, 3 are the jobs to be executed. The swapping sequences cycle will be:

   (2 1 3), (3 2 1) and (1 3 2).

2) In the team/product domain: consider each element in a cluster by itself then the cycle starts by swapping each element in the other set. For instance, if a team of 4 members A, B, C, D is considered, every element is initially set in a separate cluster [A][B][C][D]. The neighboring generator proceeds then as follows: [AB][C][D], [AC][B][D], [AD][B][C], [A][BC][D], [A][BD][C] and [A][B][CD]

   If [AC][B][D] is found to have the lowest OF, then the two elements are merged as if they were one element and the cycle restarts again.

Ten different promising heuristic and/or meta-heuristic approaches are defined for optimizing simultaneously the three domains. The performance of each approach is tested by comparing its results to the optimal result obtained by the exhaustive search method described below.

1. **BIC:** Applying the best improvement descent method on the process domain while constructing the team and the product domain.
2. **FIC:** Applying the first improvement descent method on the process domain while constructing the team and the product domain.
3. **BICSAEC:** Applying the best improvement descent method on the process domain while constructing the team and the product domain implementing SA modified new cooling schedule.
4. **FICSAEC:** Applying the first improvement descent method on the process domain while constructing the team and the product domain implementing SA modified new cooling schedule.
5. **3DBI:** Applying the best improvement descent method on the three domains.
6. **3DFI:** Applying the first improvement descent method on the three domains.
7. **3DBISAEC:** Applying the best improvement descent method on the three domains using SA new modified cooling schedule.
8. **3DFISAEC:** Applying the first improvement descent method on the three domains using SA new modified cooling schedule.
9. **3DBISAEI:** Applying the best improvement descent method on the three domains using SA new cooling schedule.
10. **3DFISAEI:** Applying the first improvement descent method on the three domains using SA new cooling schedule.

The characteristic of each of the 10 approaches is shown clearly in Table 5.1.

Table. 5.2. Various Characteristic of the 10 different approaches used in the analysis

	First Improvement	Best Improvement	SA (Osman & Christofide)	SA modified	Varying 3 domains simultaneously	Varying one domain and building other
3DFISAEI	✓		✓		✓	
3DBISAEI		✓	✓		✓	
3DFISAEC	✓			✓	✓	
3DBISAEC		✓		✓	✓	
3DFI	✓				✓	
3DBI		✓			✓	
BI		✓				✓
FI	✓					✓
BISAEC		✓		✓		✓
FISAEC	✓			✓		✓

The approaches adopted are discussed in detail where the example of Fig.5.2 is referred to through defining the search methods:

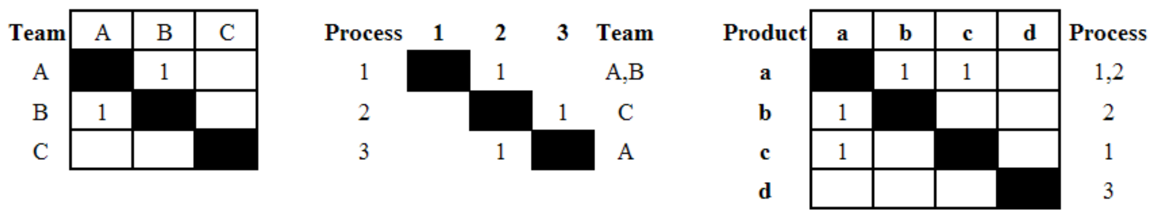


Fig. 5.2. Three domain example used to describe ten approaches

5.1.1. Exhaustively Evaluating all Possible Combinations of the Three Domains Simultaneously

Three lists of all possible combinations of each domain are enumerated. The process list contains DSM size! Ways. In the java code a combinatorial library is used to enumerate them all. The process in Fig. 5.2 is of size 3; there are 3! = 6 different ways for the tasks to be executed. The product and the team domain list size are computed using the Stirling equation. If Team T is of size 3, then there exist 5 distinct partitions; if product PDT is of size 4, then there exist 15 distinct partitions. The partitions of the combinations of the team and product domain are obtained in Fig. 5.3 in the last horizontal line for a domain of n = 3 elements.

The partitioning approach of set S = {T1, T2, T3..Tk} must satisfy simultaneously the following three conditions (Nijenhuis, 1978):

1) $Ti \cap Tj = \emptyset$ $(i \neq j)$

2) $\bigcup_{i=1}^{k} Ti = S$

3) $Ti \neq \emptyset$ (i=1,…..,k)

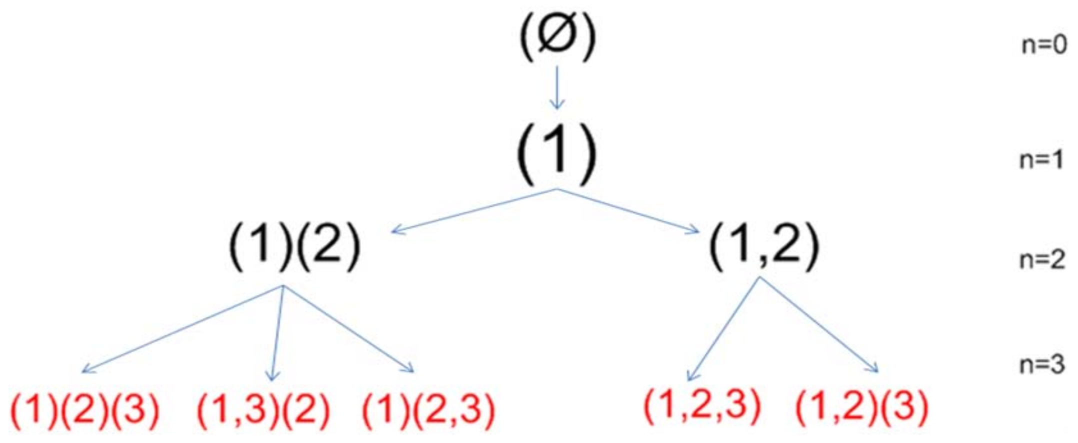


Fig. 5.3. All possible partitions of a set of size 3

The above diagram shows how the partitions are formulated for a given n element example. The tree is first divided into n+1 levels where the first level is empty by default. Each level is made of k sets and each set is made of p partitions. In each level from each set, a p+1 descendants are formed. The descendant is built by adding the n+1 element value to each of the sets and to the already existing partitions and to a separate partition each one at a time, i.e. at level n=2, there are two sets (1)(2) and (1,2) which are made of 2 and 1 partitions respectively. The descendants of the set (1)(2), which is made of 2 partitions, are 3 sets where the value n=3 is added to each of the sets. The sets of the nth level hold all the possible combinations of the domain.

The exhaustive search tackles the problem using the search approach shown in Fig. 5.4 where the number of computations done is equal to:

$$Exhaustive\ Search\ Size = Team\ List\ Size \times Product\ List\ Size \times Process\ List\ Size$$

If domains T, S and PDT of example 5.2 are considered, then the size of all possible computations is 450. For each combination, the objective function is evaluated and a cost value is assigned. Combinations with the lowest cost are saved as being the optimal and are referred to in evaluating the effectiveness of the approaches assumed.

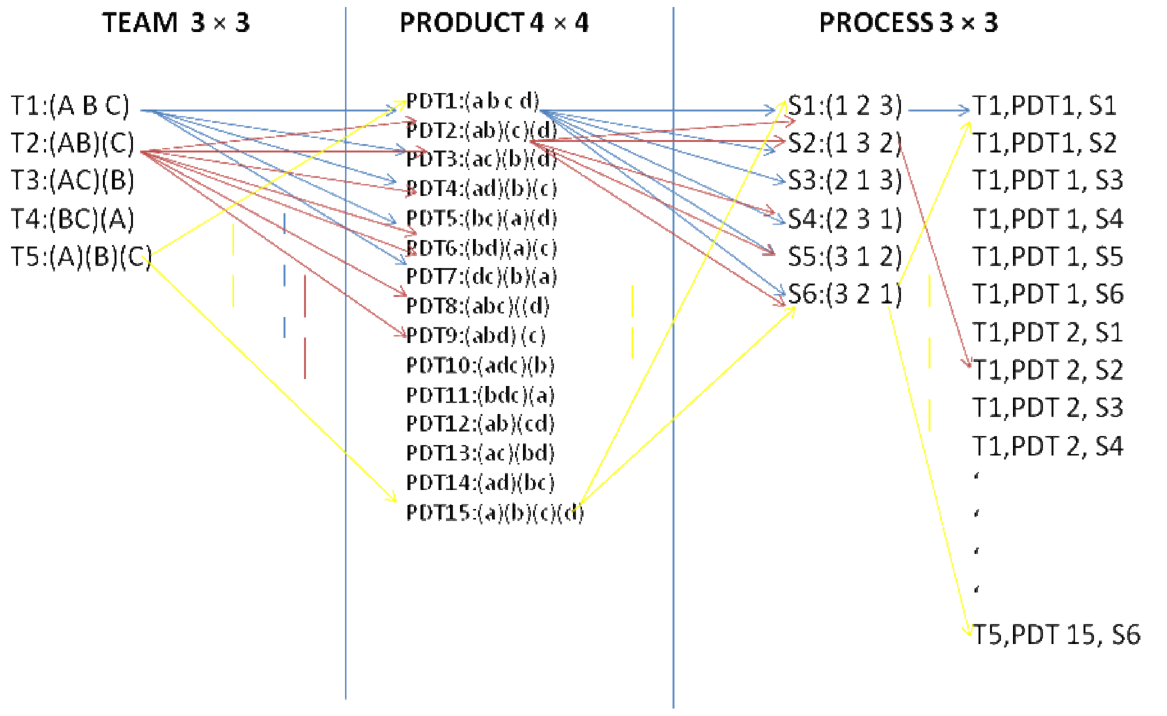
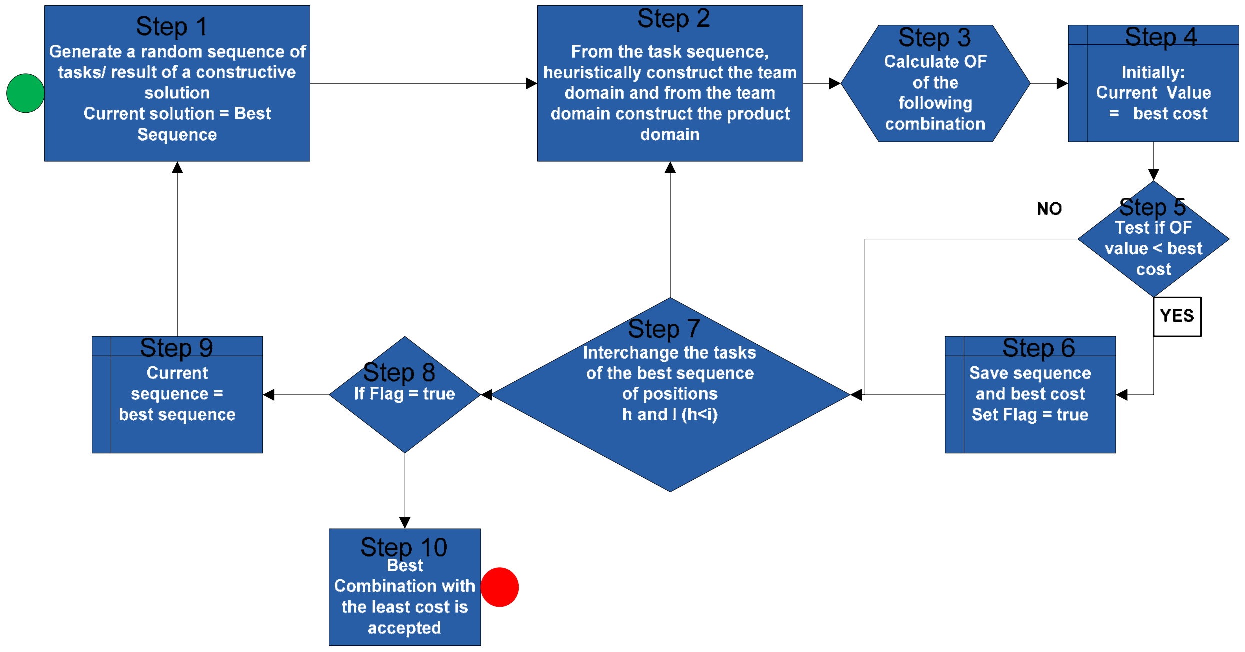


Fig. 5.4. Exhaustive search of all possibilities

5.1.2. Applying the Best Improvement Descent Method on the Process Domain while Constructing the Team and the Product Domain (BIC)

The descent approach is used to improve the solution heuristically as illustrated in Fig. 5.5.



Step 1
Generate a random sequence of tasks/ result of a constructive solution
Current solution = Best Sequence
Step 2
From the task sequence, heuristically construct the team domain and from the team domain construct the product domain
Step 3
Calculate OF of the following combination
Step 4
Initially: Current Value = best cost
Step 5
Test if OF value < best cost
NO
YES
Step 6
Save sequence and best cost
Set Flag = true
Step 7
Interchange the tasks of the best sequence of positions h and l (h<i)
Step 8
If Flag = true
Step 9
Current sequence = best sequence
Step 10
Best Combination with the least cost is accepted

Fig. 5.5. BIC flowchart

The descent approach is applied to the process domain where the neighboring generator has used the swapping technique already defined. Initially, the following should be performed:

Step 1: Generate a random sequence (S_q) of the process. (Ex. 1 3 2)
Step 2: Construct an associated solution of the Team domain Ta (S_q) and Product domain Pa ($_{Ta\ ,Sq}$).

The constructive heuristic of the cluster of the team domain is done as follows:

1. Consider every element of the team a cluster by itself.
2. Check for feedback marks in the upper triangle of the process matrix.
   2.a When a feedback mark is detected, check the team members corresponding to this task.
   2.a.1 If number of team members > 1, then assign team members in the same cluster.
   2.a.2 Repeat step 2 till no more feedbacks are detected.
3. Delete common clusters.
4. The cluster matrix obtained should be made of a maximum of T size clusters.

After constructing the team domain according to a specific sequence of processes, the product domain is constructed.
DMM team/product domain is represented from the beginning as shown below:

TEAM/PRODUCT	A	B	C	D
A	1		1	1
B	1		1	
C	1	1		

1. Analyze the team cluster combination and group the products accordingly. If A and B are in the same cluster, then the products corresponding to A and B must be in the same module. This is implemented by ORing the row of A by row of B. The result of (1 0 1 1) or (1 0 1 0) is (1 0 1 1). This means that the first cluster of the product contains components a, c and d altogether.

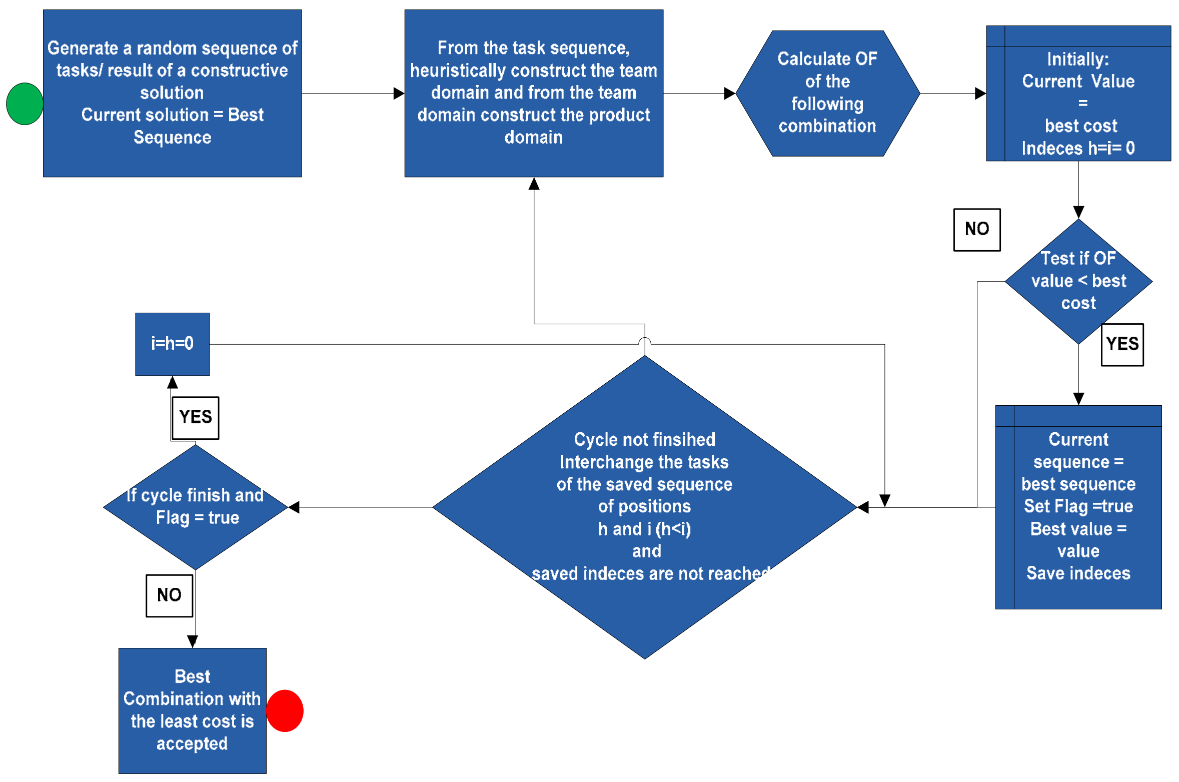
2. Repeat step 1 until all the clusters of team domain are tested.

3. Delete common module clusters.

Step 3: Compute the total current cost f(S) of the overall objective function of
(S_q U T_a U P_a);
Step 4: Save the current solution as the best solution obtained; S_c = S. (executed initially)
Step 5: Generate neighboring tasks by using the interchange improvement method until all the cycle is executed.
Select a neighboring S_q’ $\in$ N(S_q) = { S_q’ by exchanging i and j positions and looping on the indexes i and j where i < j and i: 1 till size -1 and j = i +1 till size}.
Let us consider 1, 2, 3 are the jobs to be executed. The swapping sequences will be: (2 1 3), (3 2 1) and (1 3 2). These are three ways to be considered.
Step 6: Construct T_r (Sq’) and P_r (T_a ,S_q).
Step 7: Compute f (S’ = S_q’ U P_r U T_r).
Step 8: If f (Sq’) < f (Sc),
Yes: Accept S_q’, set Sc = Sq’, update the best solution, save the best sequence accordingly, set the change flag, and go back to **step 5**.
No: Back to **step 5**
Step 9: Test if an improvement change occurred in a specific cycle:
Yes: Step 10 Update current sequence with the best sequence
Change flag = false, back to **step 4** to reinitialize a new cycle with the combination of the best improvement.
No: Step 11 The best value combination is achieved

This constructing method started by applying the descent improvement on the process domain while constructing the team and the product domain accordingly. However, through computations, if the descent improvement method started by the team or the product domain and constructed the other accordingly, the results obtained were similar. Thus, the choice of the initial domain holds no difference on the quality of solution. Moreover, the initial solution was generated randomly abiding with the literature review findings of Osman (1989) where the random search shown to give surprisingly good results.

5.1.3. Applying the First Improvement Descent Method on the Process Domain while Constructing the Team and the Product Domain (FIC)



Generate a random sequence of tasks/ result of a constructive solution Current solution = Best Sequence
From the task sequence, heuristically construct the team domain and from the team domain construct the product domain
Calculate OF of the following combination
Initially: Current Value = best cost Indeces h=i= 0
NO
Test if OF value < best cost
YES
Current sequence = best sequence Set Flag =true Best value = value Save indeces
Cycle not finsihed Interchange the tasks of the saved sequence of positions h and i (h<i) and saved indeces are not reached
i=h=0
YES
If cycle finish and Flag = true
NO
Best Combination with the least cost is accepted

Fig. 5.6. FIC flowchart

The difference between the FIC and the BIC is that the first improvement descent approach is considered a greedy method that accepts directly the first best solution obtained. The flowchart in Fig. 5.6 illustrates the steps of this method. There is no need to continue a specific cycle when an improvement is detected. This approach is considered cheaper where less iterations are made.

An example of a process of size 4 is considered. If the random sequence is (1 3 2 4) with an OF cost value = 61, then the indexing will first start with i=0 and j =1 i.e. by swapping 1 and 3. The new arrangement will be (3 1 2 4) with an OF cost value = 41 which improves the result as 41<61. The new arrangement is accordingly saved with the indexes in order to continue the swapping from this arrangement without repeating already testing indexes. The new swapping will occur with i=0 and j=2 on the improved arrangement which is (3 1 2 4) and not (1 3 2 4). Therefore, the next arrangement will be (2 1 3 4). The steps of this method will go similarly as the previous one except for the indexing and the generation of sequences.

5.1.4. Applying the Best/ First Improvement Descent Method on the Process Domain while Constructing the Team and the Product Domain Implementing SA Modified New Cooling Schedule (BISAEC & FISAEC)

The modified schedule of the New Cooling Schedule suggested by Osman and Christofides is adopted in this approach. Fig. 5.7 below illustrates in detail the flowchart of the modified new cooling schedule.

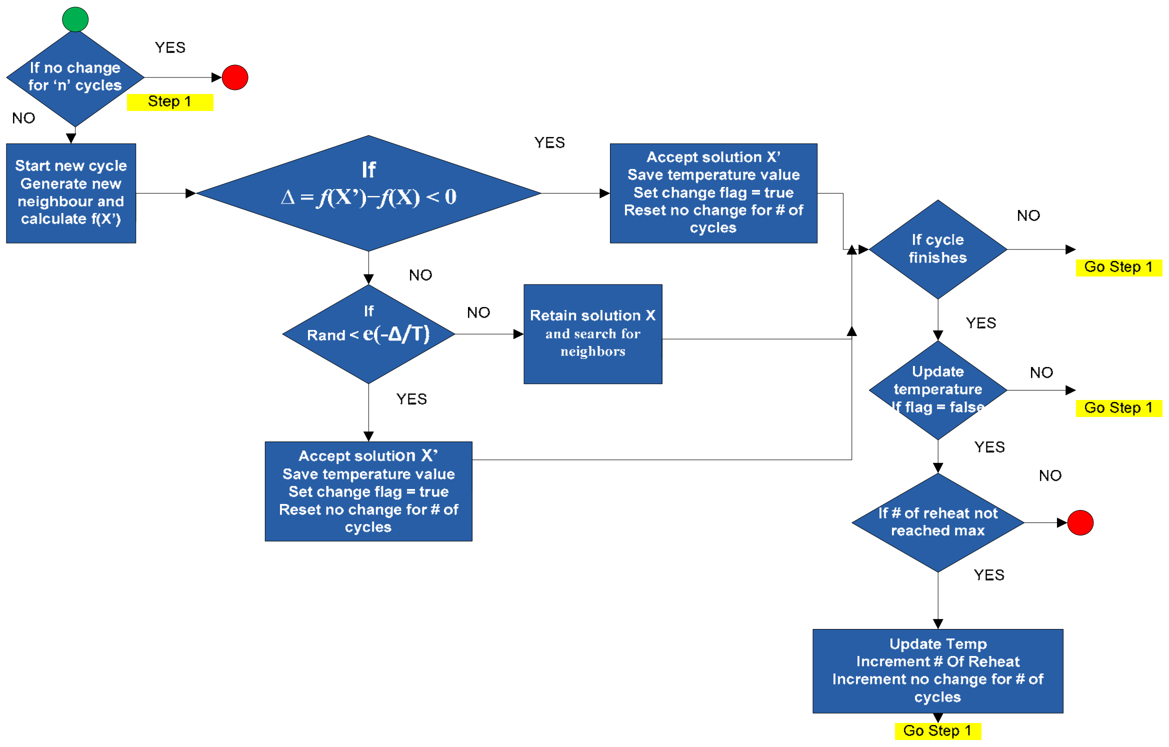

If no change for 'n' cycles
YES
Step 1
NO
Start new cycle Generate new neighbour and calculate f(X')
If $\Delta = f(X') - f(X) < 0$
YES
Accept solution X' Save temperature value Set change flag = true Reset no change for # of cycles
NO
If Rand < $e^{(-\Delta/T)}$
NO
Retain solution X and search for neighbors
YES
Accept solution X' Save temperature value Set change flag = true Reset no change for # of cycles
If cycle finishes
NO
Go Step 1
YES
Update temperature If flag = false
NO
Go Step 1
YES
If # of reheat not reached max
NO
YES
Update Temp Increment # Of Reheat Increment no change for # of cycles
Go Step 1

Fig. 5.7. Simulated Annealing flowchart of the modified cooling schedule

Recalculation of the new neighboring candidates of the three domains **X**' is done and the result is compared to the best solution saved. The three neighboring candidate is obtained through applying the first/best improvement heuristic descent method on the process domain and heuristically building the team and product domain. If the move to the three generated neighborhood gives a better objective function value (less than the best saved value) where $\Delta = f(\mathbf{X}') - f(\mathbf{X})$ is < 0, then the move is always acceptable and the neighborhood generated is saved as the best combination with a new best cost value. Yet, if an increase in the objective function ($\Delta > 0$) is found and the new cost is greater than the best cost value, then the solution will be accepted with a probability function $P = 1 - \Delta / T$ allowing the move to avoid a trap in a local optimum. T is a temperature parameter that varies from a relatively large value to a small value close to zero with occasional temperature increase when a cycle with no improvement is determined. The number of iterations k, to be performed at each temperature, represents the number of swaps performed on the process domain and differs with the process size and data.

The reheating temperature increase should not be that high in order not to escape from the local minimum and not to deviate much as if a total new random sequence is restarting. Note that the number of iterations in this case is high given the fact that the temperature is updated whenever a cycle finishes. Still, two stopping scenarios are taken into account as a trade-off between the quality of the solution and the computation time: (i) when no improvement occurs for two consecutive cycles of k iterations or (ii) when two reheatings of the temperature occur.

5.1.5. Applying the Best /First Improvement Descent Method on the Three Domains (3DBI & 3DFI)

What differs in this approach is that the descent approach is applied to the three domains simultaneously. Hence, there exists three cycles simultaneously for the three domains as shown in Fig. 5.8.Initially, the OF of the following combination is calculated where the sequence process is executed randomly and each element of the product and team domain is in a separate cluster. The value calculated is retained as the temporary best answer.

Then, the loop of generating neighborhood in the three domains starts by the process domain then team and product domains respectively. The flowchart of Fig.5.8 shows the steps of the loop in detail. The three cycles function simultaneously, as the cycle of the process runs the cycle of the team, and then the cycle of the product proceeds. As the cycle of the product finishes without any improvement, the team cycle proceeds. And then if the team cycle finishes without any improvement, the process cycle continues. However, at any stage, if an improvement occurs in a cycle, then the cycle restarts from the best improvement combination recorded. At each iteration, the OF is calculated and compared to the best solution attained. As the cycle comes to an end, the best combination is saved and will hold, if not the optimal, the near optimal solution.

The above-mentioned steps are applied in the best improvement descent method. The first improvement method, however, follows the same steps with the difference that as an improvement is caught, the cycle continues on the latest improved solution.

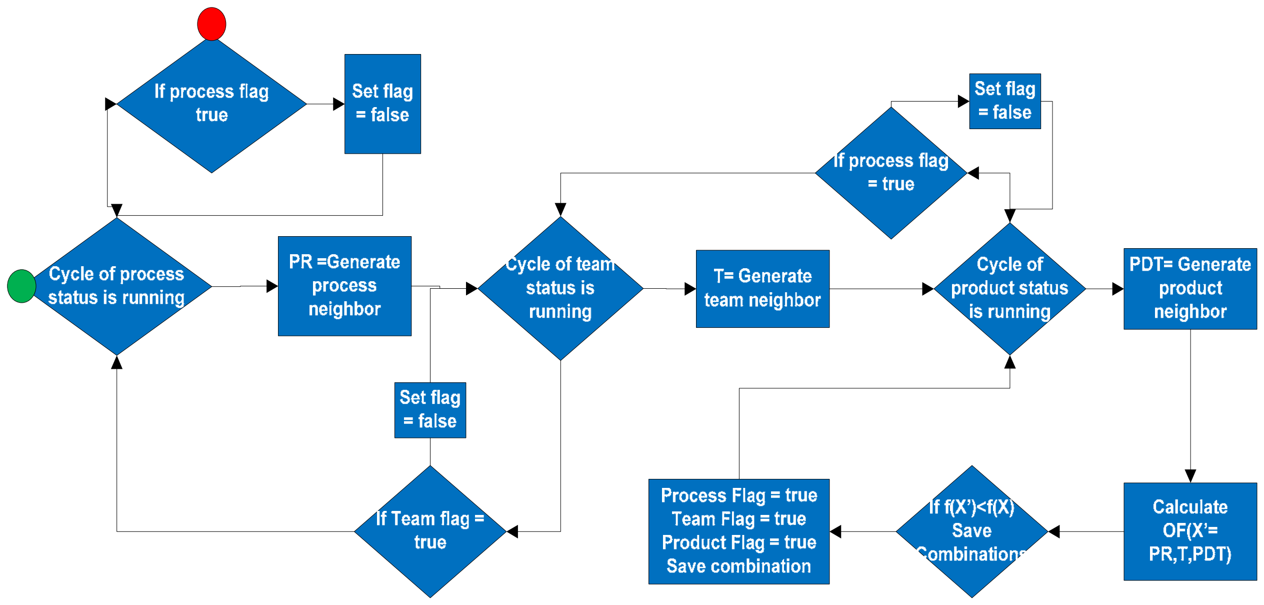

If process flag true
Set flag = false
Cycle of process status is running
PR =Generate process neighbor
Cycle of team status is running
T= Generate team neighbor
If process flag = true
Set flag = false
Cycle of product status is running
PDT= Generate product neighbor
Set flag = false
If Team flag = true
Process Flag = true
Team Flag = true
Product Flag = true
Save combination
If f(X')<f(X)
Save Combinations
Calculate OF(X'= PR,T,PDT)

Fig. 5.8. Flowchart descent approach on the three domains simultaneously

Applying the best/ first improvement descent method on the three domains using SA new modified cooling schedule. **(3DBISAEC & 3DFISAEC)** or using SA new cooling schedule **(3DBISAEI & 3DFISAEI)** follows the above detailed procedure with the temperature being updated at every iteration for the new cooling schedule and after each whole cycle for the modified new cooling schedule.

After handling in CHAPTER 5 the details of the ten implementation techniques used in the optimization of the three domains simultaneously, CHAPTER 6 shows the analysis of six hundred random test instances using these ten methods in order to recommend a single approach.

CHAPTER 6

COMPUTATIONAL RESULTS

6.1. Overview

Chapter 6 assesses the effectiveness of the developed approaches and sheds light on the performance of each approach. We start by describing the data set used for testing the ten approaches. Then an analysis of performance is presented in terms of the percent deviation of each approach compared to the best known solution and to the CPU time needed to run each approach. Other analyses are also presented to help choose the best method for solving this NP hard problem.

6.2. Data Generation

Table 6.1 shows the 15 different size problems used in the testing process where 4 different instances of each size is generated and each instance is run 10 times. This results in 40 runs for a specific size example and a total of 600 (40 ×15) different runs.

The DSM data (i.e. size and density) of these examples were generated randomly where the size of each was generated between 2 and 20. The examples were thus classified into small (sum < 15), medium (sum < 30) and large (sum ≥ 30) types according to the sum of the sizes of the three domains. The density of each DSM domain (team, process and product) was also randomly generated as shown in Table 6.1. The overall percentage shown is the average of the domain density percentages: team, process, and product. The value of this percentage is used to classify the examples into low, average and high density types according to the following percentage intervals: [0, 50[, [50, 75[and [75, 100].

The percentage density is the ratio of the number of existing marks to the maximum possible marks. Moreover, the resource allocation in the process domain is the average of the number of teams allocated on the four instances of a specific size. Hence, if a 3×3×4 example (team size × process size × product size) is considered, the average of the number of teams on the process domain of the four instances {3×3×4 (1), 3×3×4 (2), 3×3×4 (4), and 3×3×4 (4)} corresponds to the value displayed in the column of resource allocation. Similar calculation is used for the process allocation on components.

Table. 6.1. Input Data

	Team Size × Process Size × Product Size	Type	Team % Density	Process % Density	Product % Density	Overall % Density	Density Type		Resource Allocation numbers Average	Process Allocation numbers Average
1	3×3×4	Small	74.75	66.50	58.00	66.42	Low		4.00	4.00
2	4×5×4	Small	100.00	88.88	83.25	90.71	High		7.00	6.00
3	4×5×5	Small	66.25	76.33	82.50	75.03	High		7.00	8.00
4	5×4×6	Medium	40.00	72.25	74.50	62.25	Average		6.00	9.00
5	5×5×6	Medium	40.00	69.80	74.50	61.43	Average		6.00	9.00
6	6×5×5	Medium	63.00	68.17	45.00	58.72	Average		6.00	6.00
7	6×7×6	Medium	80.00	63.57	38.00	60.52	Average		11.00	9.00
8	6×8×6	Medium	78.25	59.94	42.75	60.31	Average		11.00	10.00
9	7×8×6	Medium	72.00	57.67	41.25	56.97	Average		14.00	9.00
10	9×10×8	Medium	66.50	55.00	55.50	59.00	Average		14.00	16.00
11	11×6×8	Medium	59.75	54.98	53.00	55.91	Average		12.00	13.00
12	10×10×10	Large	26.00	53.17	70.25	49.81	Low		15.00	32.00
13	12×10×9	Large	36.00	51.90	72.00	53.30	Average		13.00	17.00
14	13×8×9	Large	33.00	50.73	50.75	44.83	Low		9.00	17.00
15	13×17×8	Large	24.75	48.73	57.50	43.66	Low		18.00	40.00

6.3. Optimization of Domain in Isolation versus Simultaneous Optimization

Before delving into the analysis of the selection of the best approach, it is necessary to visualize the range of the effectiveness of the calculation derived. Is the solution retrieved by optimizing each domain alone worse or better than optimizing the three domains simultaneously?

The 60 different examples of 15 different sizes were run using two methodologies. First, the optimization in isolation procedure included the partitioning and sorting method for optimizing the process domain; Thebeau (2001) approach was used in optimizing the team and the product domain. However, the overlapping criterion was not adopted in this case given that overlapping is not justified in our calculations. Second, the 10 approaches developed were simulated for the 60 examples, and the best answer achieved was recorded and compared to the solution of optimization in isolation.

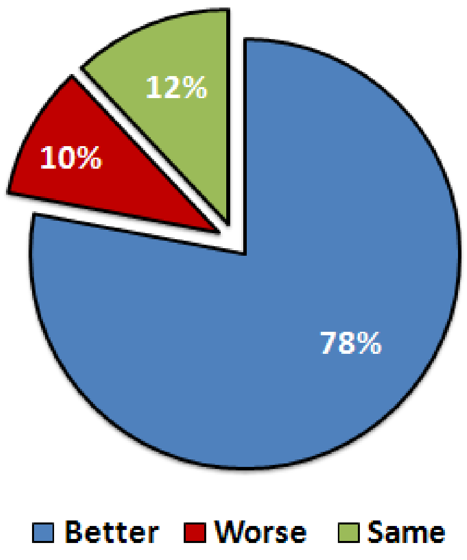


Fig. 6.1. Performance percentage of simultaneous optimization versus optimization in isolation

As shown in Fig.6.1, the simultaneous optimization performs better than the optimization in isolation for 78% of the time, while the other 10% gave equal results, and just for 12% the optimization in isolation recorded better outcomes. This signifies the importance of the method developed, but the question remains: which of the 10 approaches should be selected?

6.4. Factors of Analysis

The effectiveness of the approaches developed is tested by the following factors as discussed below:

(i) Relative deviation ratio of the best, worst and average solution;

(ii) Required CPU time;

(iii) Size of the problem (team, process and product domain); and

(iv) Density of data generated.

6.4.1. Relative Deviation Ratio (RD)

To measure the effectiveness of each method and its performance the RD ratio is used. The formula applied to calculate this ratio is:

$$\mathrm{RD} = \begin{cases} \frac{value-optimal}{optimal} & (optimal \neq 0) \\ value - optimal & (optimal = 0) \end{cases}$$

The optimal value is the reference to which the best, worst or average solution is compared to. This value is equal to the best solution found throughout the simulation of the 10 approaches.

If the value is equal to the optimal, then the RD = 0, and this is the best result obtained. However, as the RD increases, the quality of solution decreases, reflecting the high deviation from the optimal solution. This ratio, nevertheless, could be greater than 1 since for zero value optimal the deviation of the result is considered instead of the ratio. Appendix-B1 shows the average RD of the best, worst and average solutions of 15 sets, made of 4 different instances each and run for 40 times on 10 different approaches. A graphical solution of the RD is shown in Fig.6.2.

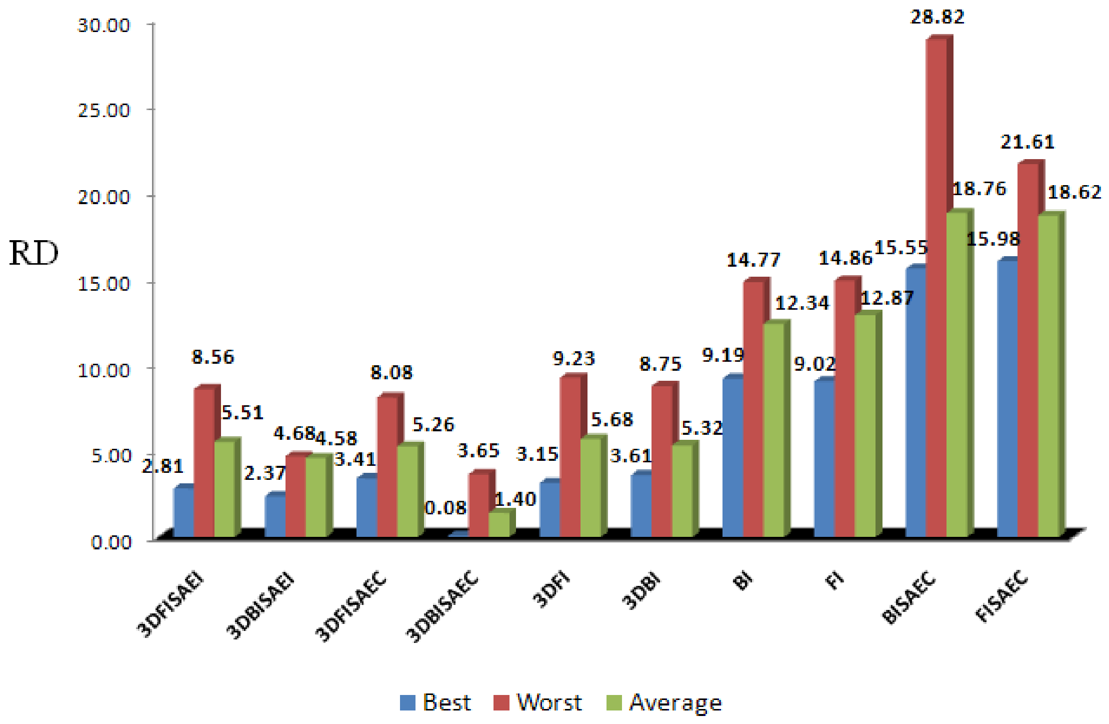


Fig. 6.2. RD ratio of the 10 approaches

As a first glance on this graph, it is clear from the data shown that the "3DBISAEC" holds the most promising solution. This is because it has the lowest RD values for the best (0.08), worst (3.65) and average (1.4) solutions.

Yet, the descent approach gives further promising results than the constructive methods. This is because the RD values of the first six methods upon using the improvement heuristic are less than the RD values of constructive approaches. This was not surprising given that the constructive method is hard to formulate although a cheap and quick approach.

Simulated annealing enhanced the result of the improvement methods approaches while worsening the solution of constructive method approaches. This is shown by comparing the RD of the 3DBI or 3DFI approaches to the other four methods using either the SA cooling schedule of Osman and Christofides or the Suggested Cooling Schedule in this thesis.

Table 6.2 shows the ranking of the approaches according to their RD. The lowest RD values are recorded first by “3DBISAEC” and second by “3DBISAEI”. This indicates the high quality solution of these approaches.

Table. 6.2. Ranking approaches according to RD

	BEST	WORST	AVERAGE
3DFISAEI	3	4	5
3DBISAEI	2	2	2
3DFISAEC	5	3	3
3DBISAEC	1	1	1
3DFI	4	6	6
3DBI	6	5	4
BI	8	7	7
FI	7	8	8
BISAEC	9	10	10
FISAEC	10	9	9

6.4.2. Required CPU Time

The target of our work is not just a quality of the solution but rather a tradeoff between the quality and the CPU time required. The CPU time is measured by the number of iterations needed for each approach. The average of the 40 runs of each set of the 15 examples is shown for each approach in the rows of the table in Appendix-C1. The average of all the 600 runs is displayed in Fig. 6.3.

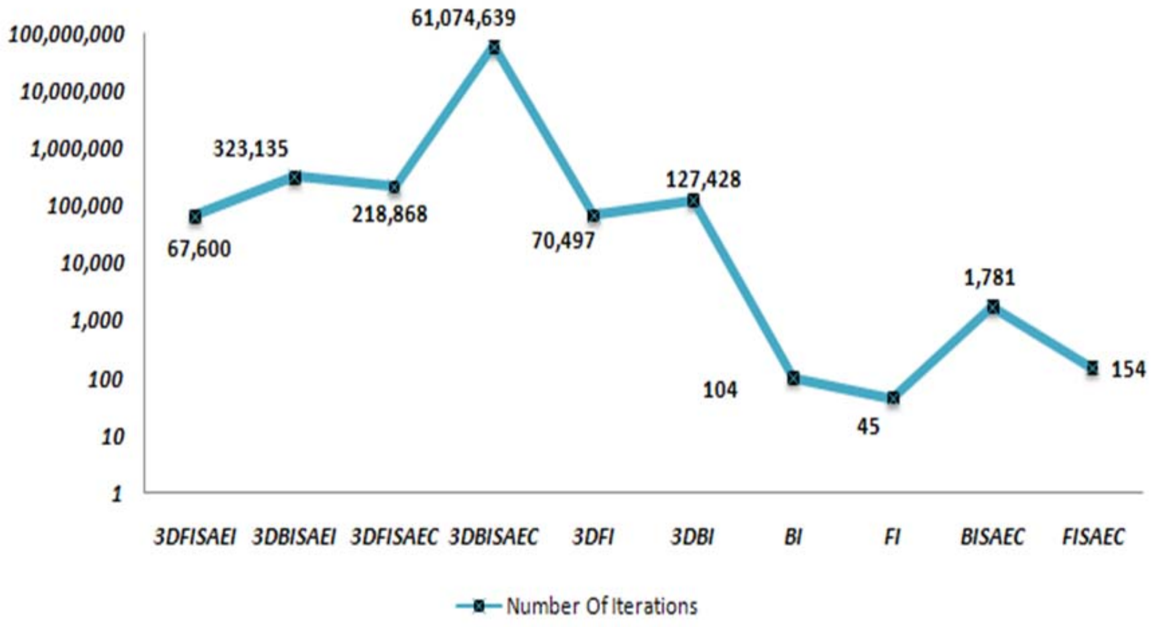


Fig. 6.3. Average of the number of iterations of 600 runs

The highest number of iterations is recorded by the “3DBISAEC” which showed the best approach for the previous criterion. This high value, which is higher from the second higher iteration by about 99.5% limits the choice of this approach.

6.4.3. Size of the Problem

Does the size of the problem affect the solution and thus the choice of the approach to be selected? The problems handled are divided into three categories: small, medium, and large. The analysis of the solution quality from the RD view and the CPU time is repeated taking the size of the problem into consideration.

The ranking of the approaches is repeated for the three different categories and the result is shown in the Table 6.3. The results match perfectly with the overall solution. The fact that the first two methods selected are still "3DBISAEC"and "3DBISAEI" for the three categories, indicates that the approach selected is irrespective of the size of the problem.

Table. 6.3. RD ranking approaches according to problem size

	Small	Medium	Large
3DFISAEI	3 or 4	4	4
3DBISAEI	**2**	**2**	**2**
3DFISAEC	3 or 4	3	3
3DBISAEC	**1**	**1**	**1**
3DFI	5 or 6	6	5 or 6
3DBI	5 or 6	5	8
BI	7	8	7
FI	8	7	5 or 6
BISAEC	10	10	10
FISAEC	9	9	9

*The "or" indicates the same ranking for approaches

Does the CPU time vary with the size of the problem? As per Appendix-C1, the number of iterations varies with the size of the problem.

Table 6.4 shows that as the size increases, the number of iterations increases with approximately the same percentage distribution. Hence, the "3DBISAEC" and "3DBISAEI" ranks the first and second place respectively with a high difference between them.

Table. 6.4. Percentage average of iterations according to size type

Size Type	3DFISAEI	3DBISAEI	3DFISAEC	3DBISAEC	3DFI	3DBI	BI	FI	BISAEC	FISAEC
Small	0.58	3.22	1.28	92.65	0.64	1.19	0.01	0.01	0.21	0.21
Medium	0.28	1.19	0.62	97.19	0.26	0.44	0.00	0.00	0.02	0.00
Large	0.10	0.49	0.34	98.78	0.11	0.19	0.00	0.00	0.00	0.00

6.4.4. Density of Data Generated

According to Table 6.1, the categorization of the 15 examples into high, low or average density (as shown in the last column) is taken into account to check whether the density of problems affects the approach selected.

The average of RD of each category is shown in Appendix-B3. However, a similar ranking of the approaches is calculated, and the result is shown in Table 6.5.

Table. 6.5. RD ranking approaches according to density type

	High	Low	Average
3DFISAEI	4	3	3
3DBISAEI	**2**	**2**	**2**
3DFISAEC	3	6	6
3DBISAEC	**1**	**1**	**1**
3DFI	6	5	5
3DBI	5	4	3
BI	7	7	7
FI	8	8	7
BISAEC	10	9	9
FISAEC	9	10	10

From the information at hand, it can be deduced that the data density of the problem holds the same result and analysis as the size criterion, thus does not affect the approach selection of the best two methods.

It can be shown, moreover, from Table 6.6 that the number of iterations varies in the same percentage from one approach to the other. The ranking of the approaches remained the same. “3DBISAEC” has the highest average percentage of iterations followed by "3DBISAEI”.

Therefore, it can be deduced that the density of data is not related to the number of iterations performed on each approach.

Table. 6.6. Percentage average of iterations according to density data type

Density Type	3DFISAEI	3DBISAEI	3DFISAEC	3DBISAEC	3DFI	3DBI	BI	FI	BISAEC	FISAEC
High	0.57	3.16	1.29	92.80	0.63	1.17	0.01	0.01	0.20	0.16
Low	0.09	0.46	0.34	98.82	0.10	0.19	0.00	0.00	0.00	0.00
Average	0.15	0.70	0.39	98.31	0.17	0.26	0.00	0.00	0.01	0.00

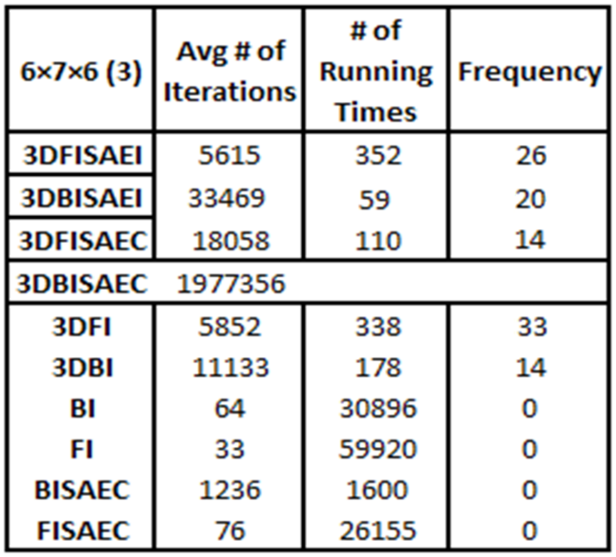
6.5. Selected Approach

Is the benefit of the value obtained by "3DBISAEC" at the cost of the time and, accordingly, the number of iterations?

To answer this question, various tests are done. Set 6×7×6 of the third instance is used as n example. This choice is justified because the "3DBISAEC" approach scored in that case the best solution that neither of the other methods could reach. The number of iterations of each approach is shown in the second column. The average iterations of 10 runs of the 6×7×6 set of the third instance is as displayed in Appendix-C2.

Equal time for all approaches is assigned, and a normalization of the other 9 approaches with respect to "3DBISAEC" is made where the running time for each is set forth in the second column of Table 6.7.

Table. 6.7. Normalizing the time factor of the 10 approaches for the 6×7×6 (3) example



6×7×6 (3)	Avg # of Iterations	# of Running Times	Frequency
3DFISAEI	5615	352	26
3DBISAEI	33469	59	20
3DFISAEC	18058	110	14
3DBISAEC	1977356		
3DFI	5852	338	33
3DBI	11133	178	14
BI	64	30896	0
FI	33	59920	0
BISAEC	1236	1600	0
FISAEC	76	26155	0

The results show that the constructive methods were never able to achieve the best found result. This is another indicator proving that the constructive approach is not efficient in optimizing the three domains simultaneously. It is thus considered a dirty and cheap method.

However, improvement approaches, whether with or without simulated annealing, are able to achieve the best found result for more several times as shown in the frequency column in Table 6.7. The highest frequency was nevertheless recorded by the "3DFI" approach.

In addition, it is necessary to visualize the effect of the number of iterations on the quality of the solution.

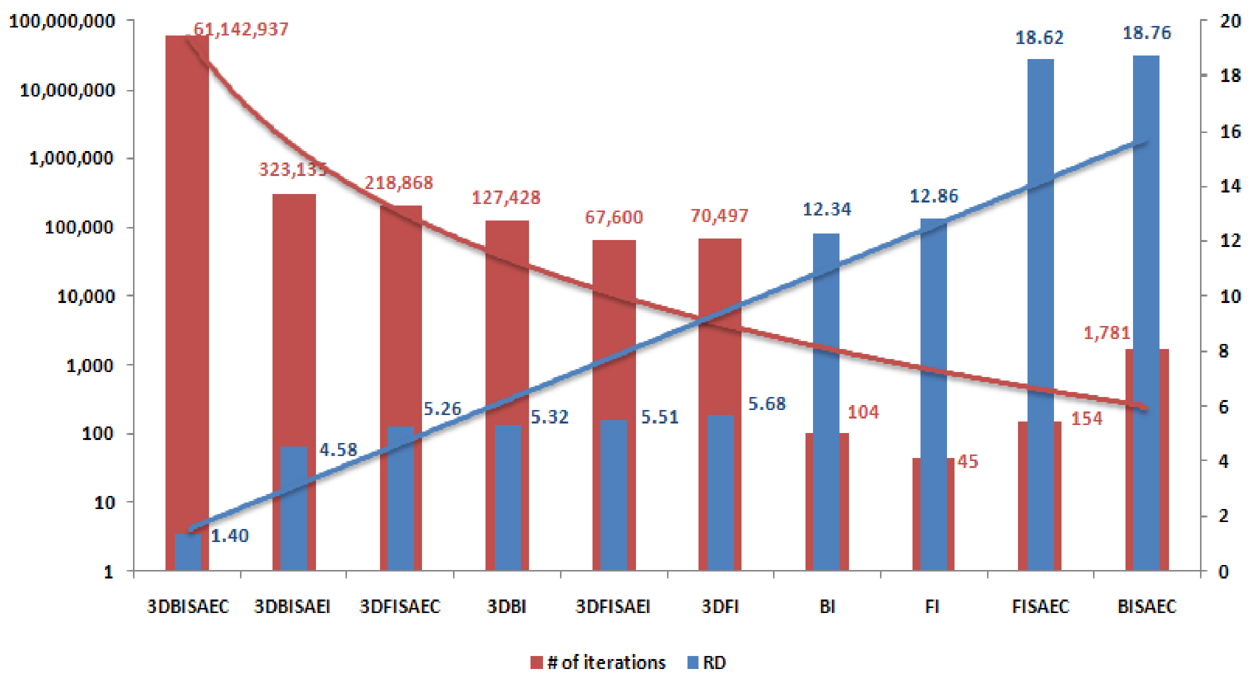


Fig. 6.8.The effect of number of iterations on the quality of solution

Fig. 6.8 shows that the approach with the highest number of iterations leads to better results except for the “BISAEC” and “FISAEC” approaches where the simulated annealing applied did not enhance the quality of the solution. This was due to the low number of iterations of the constructive method where worse solutions were accepted highly in the early stages hence in most of the iterations.

Fig. 6.9 shows the overlapping of the exponential trend line of both figures. The exponential trend line of the number of iterations shows that the improvement approaches record higher than the constructive approaches. The “3DBISAEC” records the highest value down till “FI”. The exponential trend line of the relative deviations shows that the best solution achieved is the most expensive approach, i.e. “3DBISAEC” in this case. The approaches are displayed from left to right showing the decrease in the quality of solution.

The intersection point demonstrates that the “3DFI” approach was chosen to be a tradeoff between cheap and dirty approaches versus more accurate but expensive approaches.

The quality of solution achieved by one“3DFI”run is less than “3DBISAEC”, “3DBISAEI” and others. Still, if this approach were run for equal time as in the most expensive method, it would find the solution of 33 times. This indicates that approximately 10% (338/33) of the time run by “3DBISAEC”would guarantee the best found solution to be achieved by the “3DFI” approach and with less number of iterations and less time.

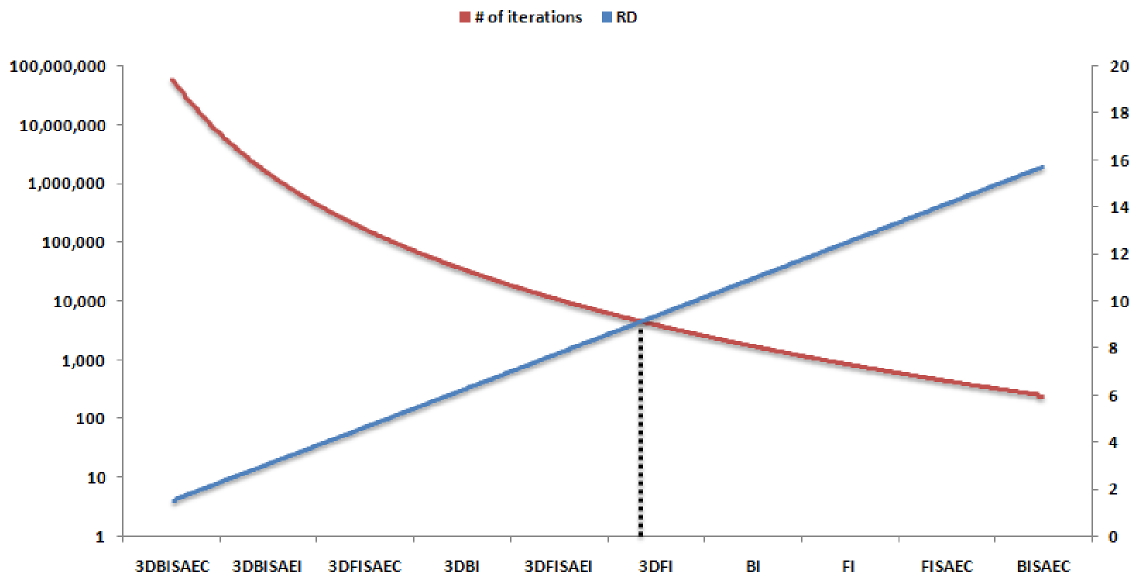


Fig. 6.9.The recommended trade off approach

Finally, simulated annealing showed benefits in improvement heuristic approaches but still did not do well. This might be due to a miss of the best solution when worse solutions were accepted and due to the absence of parallel memory savings. Thus, the combination of simulated annealing with other artificial intelligence heuristics (such as tabu search or genetic algorithm which are beneficial in individual optimization [Osman, 1994]) might as well enhance the solution quality of a global optimization.

CHAPTER 7

CONCLUSION

7.1. Summary

Organizations are defined by their team structure and their ability to execute product development processes in a certain sequence to produce salable products to meet the market needs and growth. Up to this point of time, organizations seek an individual and isolated optimization of each of the product, process and team domains. The main goal of this thesis, however, as shown in CHAPTER 1, is to formulate a global optimal solution for the product development organization problem. CHAPTER 2 includes a literature review of the design structure matrix and the existing optimization techniques used. Moreover, CHAPTER 3 describes the heuristic and meta-heuristic search methods and models used in the literature as well.

CHAPTER 4 illustrates the work to formulate the simultaneous optimization of the three domains using a multi-domain DSM model and to build three relational rules of inter and intra dependencies within and between the domains. Moreover, CHAPTER 5 describes the ten hybrid methods combining heuristics and meta-heuristic techniques to solve large size problems. The performance of these ten approaches is tested in CHAPTER 6 which includes the analysis of six hundred random test instances with the most recommended method being the “3DFI”. This method shows a tradeoff between the quality of the solution and the required computational CPU time. Constructive approaches are shown to be less performing than the improvement descent ones where the latter results are enhanced by the use of simulated annealing. Finally, a software code using JAVA is designed in a user friendly interface to compute the results achieved by the ten different approaches in an excel format file.

7.2. Limitations and Weaknesses

Several limitations might be faced while implementing the global optimal solution. These are, among others: The required data for the three domains may be unreachable or inaccurately collected;

1. The relational rules among the three domains are not detailed; and
2. The dilution of interface and feedback marks is over estimated.

The weaknesses faced while simultaneously optimizing the three domains include, without limitation:

1. The analysis is performed over a limited size range examples (0, 20).
2. The recommended approach is based on the result of one testing instance example due to time limitation.
3. Data are randomly generated and do not reflect real case situations.

7.3. Recommendations for Future Work

The following steps are recommended:

1. An added value option in the software should be developed to retrieve the information directly from the organizational database to minimize the error margin in collecting data and decrease data time collection.

2. The amount of dilution of the relational rules is entered as a number between 0 and 1. Hence, this amount should be tested and assigned in later research.
3. The relational rules should consider penalty on the size and number of the clusters formed as well as on the number of resources and processes mapped to the process and product domains respectively.
4. More rules should be formulated through extensive analysis of the inter and intra dependencies among the three domains.
5. More examples of larger sizes (> 20) should be tested to guarantee the correctness of the selected approach.
6. Real case example should be tested to acquire the real work benefit.
7. Further testing should be done on the way the initial solution of the three domains is generated; this is because few examples are tested with an initial solution equal to the optimal solution retrieved by optimizing each domain separately and resulted in placing all elements in one cluster. Due to the fact that the initial optimal solution is mainly consisted of high cluster formation and the neighbourhood generation method used in this thesis tries to build clusters rather than to destroy. Refer to Appendix-E for details.
8. Parallel implementation of several meta-heuristics approaches should be implemented, such as combining both simulated annealing and tabu search.
9. Improving and/or promoting the appearance of the software interface to show the result in matrix rather than in English form.

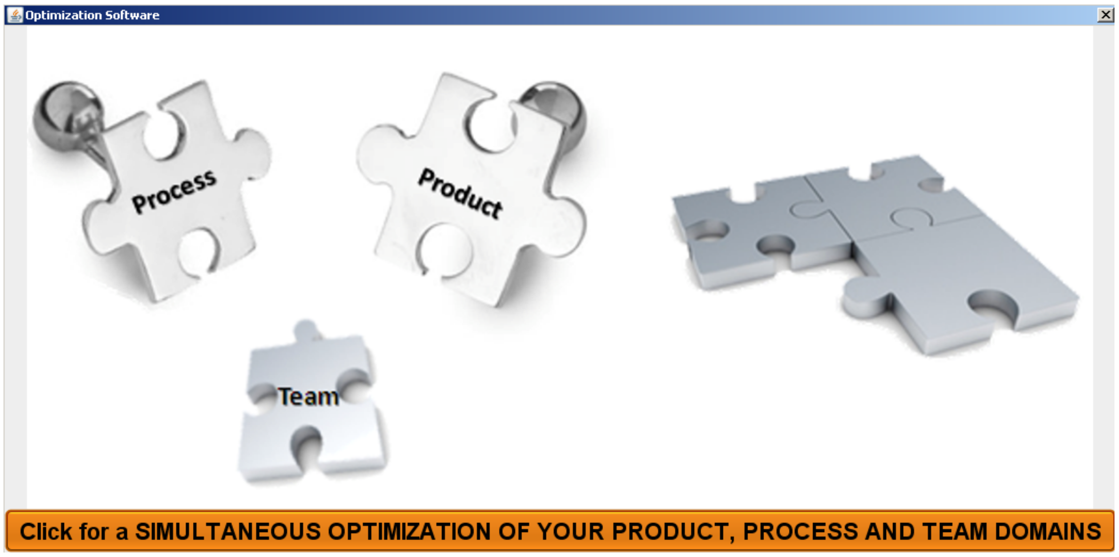
APPENDIX

APPENDIX - A

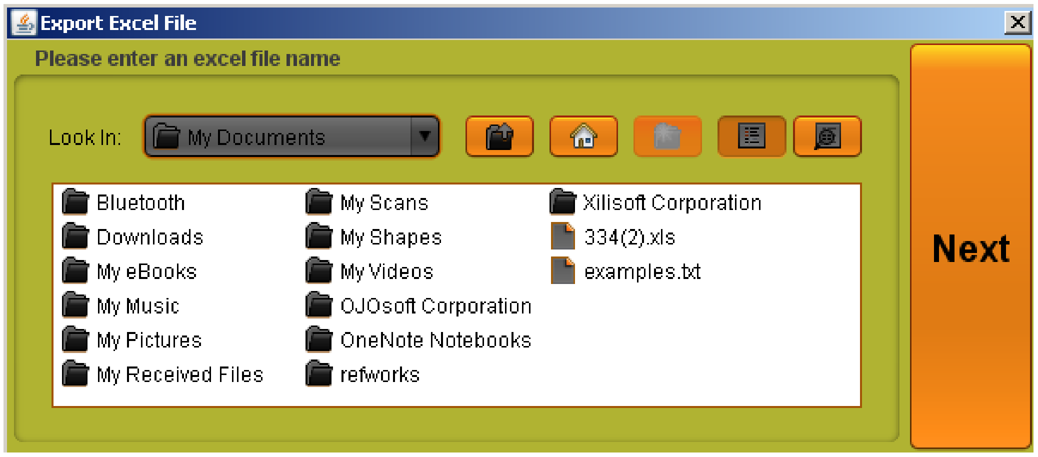
SOFTWARE MANUAL

1. Run the .bat file.

2. The below window is shown. Click for a simulatneous optimization of your organization structure.


Optimization Software
Process
Product
Team
Click for a SIMULTANEOUS OPTIMIZATION OF YOUR PRODUCT, PROCESS AND TEAM DOMAINS

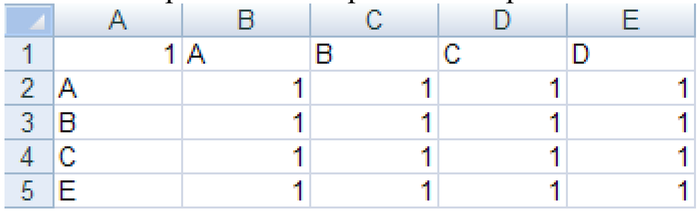
3. Browse your excel file and click on next.



- **Note that just excel file is accepted, the next button is frozen in case other file extension is chosen.**

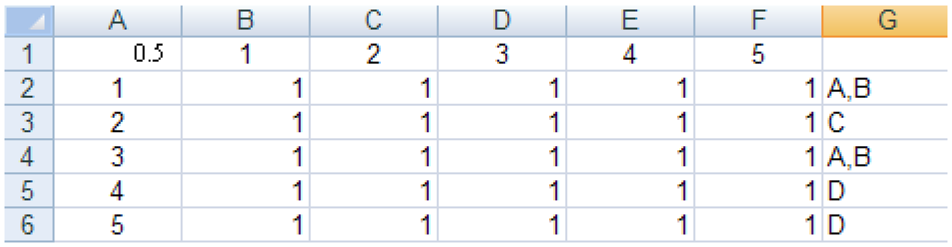
The format of the excel file should have 3 sheets in the following **ORDER:** Team, Process and Product respectively. The file sheet should start entering the data from the first column using the first cell A1 to weight the domian between 0 and 1. The titles should be added horizantally and vertically. The additional information in the process (specifiying the team member for each task)and the product domain (specifying the task to be executed on this component) should be added accurately where the data entered must be an existing element in the problem. No extra information should be added in the sheet else an error occurs.

Here is a snapshot of team process and product excel sheets:



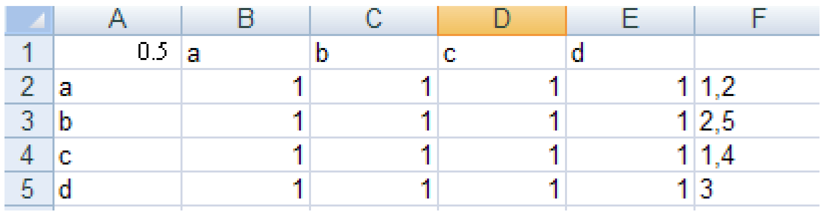
	A	B	C	D	E
1	1	A	B	C	D
2	A	1	1	1	1
3	B	1	1	1	1
4	C	1	1	1	1
5	E	1	1	1	1

Fig. A.1: Team Excel Sheet



	A	B	C	D	E	F	G
1	0.5	1	2	3	4	5	
2	1	1	1	1	1	1	A,B
3	2	1	1	1	1	1	C
4	3	1	1	1	1	1	A,B
5	4	1	1	1	1	1	D
6	5	1	1	1	1	1	D

Fig. A.2 Process Excel Sheet

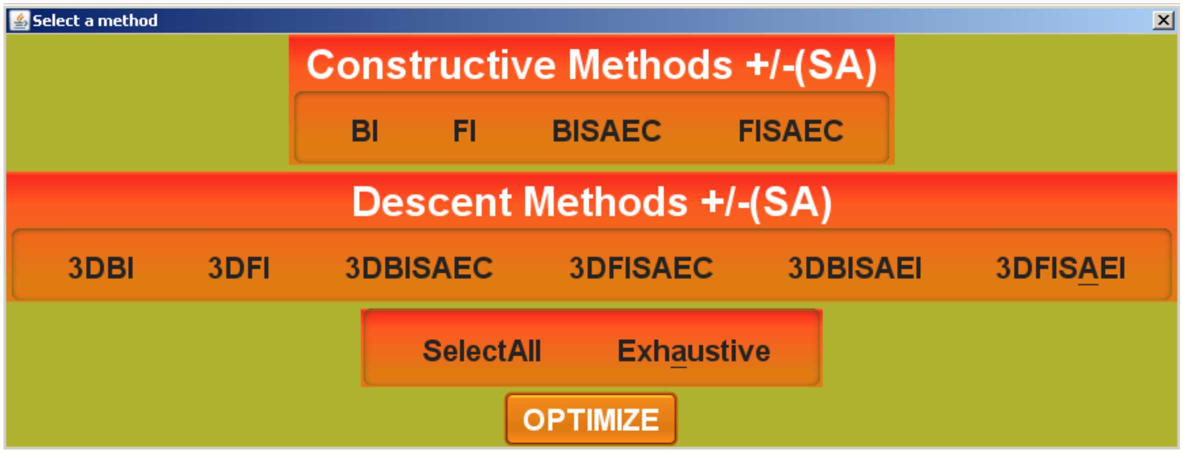


	A	B	C	D	E	F
1	0.5	a	b	c	d	
2	a	1	1	1	1	1,2
3	b	1	1	1	1	2,5
4	c	1	1	1	1	1,4
5	d	1	1	1	1	3

Fig. A.3: Product Excel Sheet

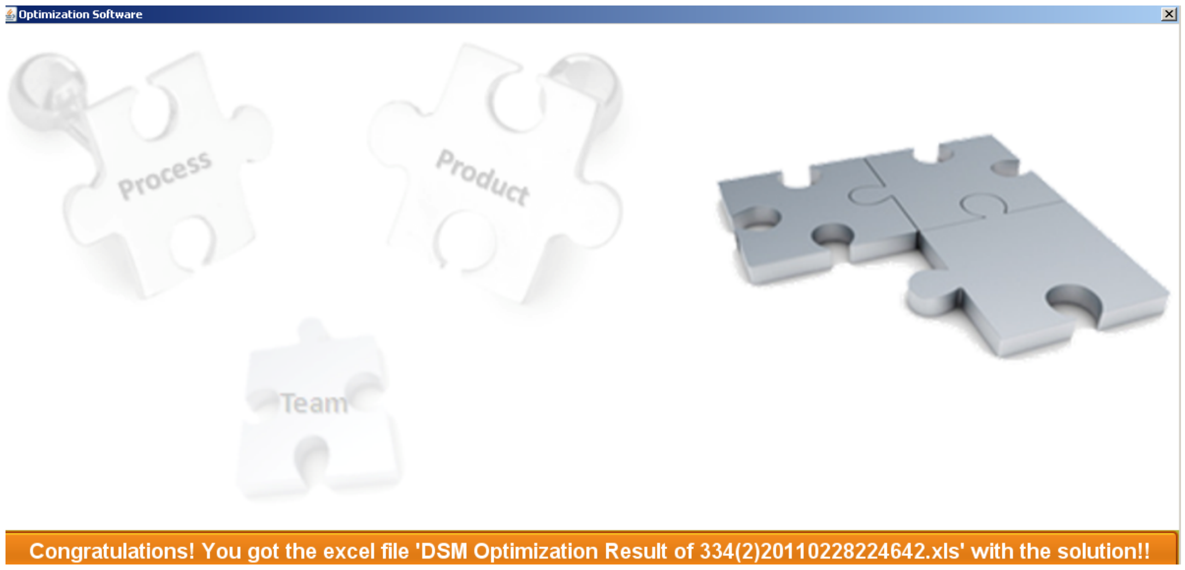
Note that the higher weight of the team (1) than the weight of process and product (0.5) indicates that the effect of the rules on the team domain is reduced compleletey while other rules are reduced by half.

4. Select one or more of the approaches described and click on optimize button.



- **Select all applies all the methods including the exhaustive method**

5. If the information was entered correctly a window is shown indicating that an excel file with the naming:DSM Optimization Result of "**fileNameExported**"+time in millisecomds.xls is created in the location where the software is found.



Congratulations! You got the excel file 'DSM Optimization Result of 334(2)20110228224642.xls' with the solution!!

This file includes the input data of the process, product and team domains with sheet for every method selected. This sheet contains the optimal solution cost and combination achieved for every selected approach. The result is shown in an english plain text instead of a matrix model. The ";" enumerates the elements of a cluster where as ":" announces a new cluster formation. For example, if the optimal solution for the team domain was

[A;B;E:C], this means there are two cluster teams. The first includes A, B and E and the second has just member “C”.

APPENDIX- B

AVERAGE RELATIVE DEVIATION

APPENDIX- B1

High Level Data

			3DFISAEI			3DBISAEI			3DFISAEC			3DBISAEC			3DFI			3DBI			BI			FI			BISAEC		FISAEC	
		B	W	A	B	W	A	B	W	A	B	W	A	B	W	A	B	W	A	B	W	B	B	W	A	B	W	B	W	A
1	3×3×4	0.00	8.47	4.99	0.00	0.00	0.00	8.25	9.12	8.82	0.00	8.25	2.15	6.25	6.25	6.25	6.25	6.25	6.25	9.47	28.25	20.74	9.47	28.25	24.48	28.24	28.25	28.24	28.25	28.25
2	4×5×4	33.00	37.00	34.80	23.00	36.50	31.35	11.00	16.50	12.08	0.00	31.00	12.35	29.50	61.50	41.00	35.00	39.00	38.20	30.75	45.75	40.75	30.75	45.75	40.75	45.50	47.00	45.75	45.75	45.75
3	4×5×5	4.50	37.75	18.40	4.50	0.00	17.95	26.50	45.75	33.50	0.00	3.50	1.40	2.25	26.00	13.55	4.50	41.75	13.85	14.25	38.75	28.45	14.25	36.50	29.10	42.50	177.75	44.75	77.25	58.75
4	5×4×6	0.08	3.92	2.76	2.92	3.04	2.97	0.13	13.93	11.70	0.33	3.03	1.08	3.17	3.75	3.34	2.92	3.99	3.22	5.42	6.83	6.00	5.83	6.83	6.23	8.45	9.06	7.75	11.96	9.46
5	5×5×6	0.00	4.72	1.88	0.11	0.75	0.33	0.14	5.03	0.90	0.18	0.80	0.40	0.34	3.55	1.19	0.11	2.58	0.79	3.80	4.19	4.11	3.91	4.19	4.13	4.77	5.30	4.19	6.88	5.18
6	6×5×5	0.11	2.95	1.27	0.20	2.41	0.97	0.18	1.46	0.78	0.09	0.79	0.29	0.20	2.81	1.60	0.20	2.24	0.73	4.70	4.87	4.78	4.34	4.79	4.69	5.16	5.39	5.10	6.55	5.62
7	6×7×6	0.23	5.56	2.71	0.13	5.08	2.30	0.85	6.51	1.96	0.03	0.59	0.18	0.08	4.05	1.84	0.13	7.18	1.84	11.42	23.74	17.26	13.15	24.00	20.54	24.90	26.78	22.09	34.31	27.78
8	6×8×6	0.77	9.26	5.59	0.69	6.01	3.00	1.39	6.28	2.48	0.16	1.64	0.91	1.06	8.54	5.17	0.63	8.92	3.90	21.52	24.36	22.11	18.26	23.48	21.27	25.90	27.17	29.77	34.21	32.61
9	7×8×6	0.66	5.47	2.68	1.42	5.07	2.91	0.66	5.23	2.09	0.10	1.09	0.39	1.76	4.20	3.10	0.92	3.82	2.43	8.13	10.01	9.01	6.64	9.79	8.54	10.81	47.82	13.11	14.53	13.75
10	9×10×8	0.50	2.77	1.61	0.50	2.66	1.31	0.55	2.85	1.17	0.00	0.75	0.33	0.75	2.64	1.38	0.92	2.78	1.70	4.05	4.66	4.37	4.17	4.66	4.43	3.26	5.37	5.07	6.45	5.65
11	11×6×8	0.12	1.29	0.76	0.43	0.92	0.66	0.13	1.05	0.67	0.28	0.43	0.35	0.17	1.70	0.94	0.42	1.50	0.95	1.64	1.98	1.78	1.69	2.20	1.89	2.34	14.14	2.15	3.20	2.68
12	10×10×10	0.48	2.12	1.36	0.41	1.82	1.09	0.47	2.14	0.74	0.07	0.52	0.36	0.20	4.73	1.81	0.47	3.55	1.63	1.32	2.85	1.96	1.29	2.20	1.81	2.74	3.88	3.29	7.19	5.05
13	12×10×9	0.50	2.12	1.17	0.47	1.66	0.97	0.27	1.97	0.54	0.00	0.58	0.30	0.63	1.74	1.17	0.71	3.02	1.78	1.51	1.94	1.68	1.55	2.01	1.74	2.29	2.72	2.33	4.08	3.12
14	13×8×9	0.30	2.31	1.14	0.33	1.80	1.26	0.33	1.80	0.91	0.00	1.40	0.44	0.16	1.68	0.75	0.16	2.06	0.94	4.85	6.74	5.89	4.89	6.48	6.08	7.18	9.00	8.04	11.22	9.48
15	13×17×8	0.91	2.73	1.57	0.49	2.47	1.62	0.34	1.65	0.62	0.00	0.37	0.12	0.65	5.37	2.12	0.77	2.59	1.57	14.99	16.69	16.29	15.06	21.79	17.34	19.19	22.71	18.09	32.31	26.19
	Average	2.81	8.56	5.51	2.37	4.68	4.58	3.41	8.08	5.26	0.08	3.65	1.40	3.15	9.23	5.68	3.61	8.75	5.32	9.19	14.77	12.34	9.02	14.86	12.87	15.55	28.82	15.98	21.61	18.62

*Average Relative Deviation of the best, worse and average solutions of 15 examples

*Each example is the average of four instances which is simulated 10 times.

APPENDIX- B2

Detailed Level Data

			3DFISAEI			3DBISAE			3DFISAE			3DBISAE			3DFI			3DBI			BI			FI			BISA			FISA	
		B	W	A	B	W	A	B	W	A	B	W	A	B	W	A	B	W	A	B	W	B	B	W	A	B	W	A	B	W	A
	1	0	0.9	0.2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2.9	3	3	2.9	3	2.9	3	3	3	3	3	3
334	2	0	33	20	0	0	0	33	33	33	0	33	8.6	0	0	0	0	0	0	33	108	78	33	108	93	108	108	108	108	108	108
	3	0	0	0	0	0	0	0	3.5	2.3	0	0	0	0	0	0	0	0	0	2	2	2	2	2	2	2	2	2	2	2	2
	4	0	0	0	0	0	0	0	0	0	0	0	0	25	25	25	25	25	25	0	0	0	0	0	0	0	0	0	0	0	0
	1	45	45	45	40	45	44	0	0	0	0	35	13	45	45	45	45	45	45	0	30	16	0	30	22	29	31	30	30	30	30
454	2	28	42	34	7	42	31	28	50	32	0	28	5.6	28	50	44	50	50	50	50	50	50	50	50	50	50	51	51	50	50	50
	3	14	16	16	0	14	5.6	16	16	16	0	16	13	0	106	30	0	16	13	73	73	73	73	73	73	73	73	73	73	73	73
	4	45	45	45	45	45	45	0	0	0	0	45	18	45	45	45	45	45	45	0	30	24	0	30	18	30	33	31	30	30	30
	1	18	54	34	18	0	43	0	0	0	0	0	0	9	36	25	18	81	36	0	18	7.2	0	9	5.4	19	21	20	18	36	23
455	2	0	0	0	0	0	0	0	0	0	0	14	5.6	0	13	2.6	0	9	1.8	34	58	39	34	58	43	60	83	73	71	125	88
	3	0	88	37	0	0	29	0	77	28	0	0	0	0	55	26	0	77	18	14	25	23	14	25	23	32	541	136	36	69	49
	4	0	9	1.8	0	0	0	106	106	106	0	0	0	0	0	0	0	0	0	9	54	45	9	54	45	59	66	63	54	79	74
	1	0	1.7	0.3	0	0	0	0	0	0	0	7.7	1.6	0	1.7	0.3	0	0	0	2.9	4.9	3.7	2.9	4.9	4.1	7	7.9	7.3	8.6	12	10
546	2	0.3	1	0.5	0.2	0.2	0.2	0	0.9	0.1	0.3	0.4	0.4	0.2	0.3	0.2	0.2	1	0.4	0.6	0.6	0.6	0.6	0.6	0.6	0.8	0.9	0.8	0.6	1.3	1
	3	0	0.5	0.2	0	0.5	0.2	0.5	18	9.7	0	0	0	0	0.5	0.3	0	0.5	0.4	6.2	7.8	6.5	7.8	7.8	7.8	7.5	7.5	7.5	7.8	7.8	7.8
	4	0	13	10	12	12	12	0	37	37	1	4	2.3	13	13	13	12	15	12	12	14	13	12	14	12	19	20	19	14	27	19
	1	0	1.4	0.3	0	0.8	0.3	0	2.9	1.2	0	0	0	0	1.4	0.6	0	0	-0	3.2	3.2	3.2	3.2	3.2	3.2	3.9	4.7	4.3	3.2	6.3	4.5
556	2	0	4.4	1.6	0	1.1	0.2	0	5.1	0.4	0	0.4	0.2	0	3.4	0.7	0	0	0	2.9	4.4	4.1	3.9	4.4	4.3	4.9	5.2	5	4.4	6.4	5
	3	0	4.6	2.3	0.5	1.1	0.8	0	4.6	0.5	0.2	0.5	0.3	1.4	3.1	2.1	0.5	4.6	1.7	4.4	4.5	4.4	3.8	4.5	4.3	5	5.5	5.2	4.5	7	5.3
	4	0	8.4	3.4	0	0	0	0.556	7.4	1.6	0.6	2.3	1.1	0	6.2	1.4	0	5.7	1.6	4.7	4.7	4.7	4.7	4.7	4.7	5.3	5.8	5.5	4.7	7.8	5.9
	1	0	6.6	2.5	0	5.9	1.3	0.308	1.5	0.5	0	1.5	0.3	0	6.5	3.5	0	5.9	1.3	6.5	6.8	6.5	5.2	6.5	6.2	7.5	8.1	7.7	7.3	11	8.8
655	2	0	1.3	0.4	0	0.1	0.1	0	0.1	0.1	0	0.4	0.2	0	0.9	0.2	0	0.1	0.1	1.6	1.6	1.6	1.6	1.6	1.6	2	2.1	2.1	2	3.5	2.3
	3	0.2	1.8	1.2	0.8	1.5	1	0.182	1.8	0.3	0	0.3	0.1	0.8	1.8	1.2	0.8	0.8	0.8	3.9	4.3	4.2	4.2	4.3	4.2	4.5	4.5	4.5	4.3	4.9	4.5
	4	0.3	2.1	1	0	2.1	1.5	0.25	2.4	2.2	0.4	1	0.6	0	2.1	1.4	0	2.1	0.7	6.8	6.8	6.8	6.4	6.8	6.7	6.8	6.8	6.8	6.8	6.8	6.8
	1	0	12	6.9	0	15	6.9	0	12	3.9	0	0	0	0	11	4.9	0	18	3.6	28	47	38	35	47	41	48	51	50	47	69	56
676	2	0	7	2	0	2.5	0.5	2.5	11	2.7	0	1.5	0.3	0	2.5	1	0	8	2.1	15	44	27	15	44	36	46	49	48	36	61	47
	3	0.9	1.2	1.1	0.4	1.3	0.8	0.909	2.6	1.1	0	0.5	0.2	0.2	1.5	0.8	0.4	1.3	0.9	0.9	1.4	1.2	1.4	1.8	1.7	2.3	2.7	2.5	2.1	4	3.5
	4	0	2	0.9	0.1	2.1	1	0	0.4	0.2	0.1	0.4	0.3	0.1	1.7	0.6	0.1	1.4	0.8	2.3	3.6	3.3	1.2	4.2	3.4	3.8	4.4	4	4.2	4.2	4.2
	1	2.5	16	11	0	15	7.3	2.5	16	4.8	0	3.5	2	2.5	13	8.4	0	19	7.1	41	47	42	41	43	41	49	51	49	61	71	67
686	2	0.6	2.4	1.5	0.3	2	0.8	0.563	2.4	0.8	0.3	0.9	0.6	0.3	4.8	2.1	0	2.3	0.7	2.4	2.4	2.4	2.4	2.4	2.4	3.5	3.9	3.8	4.4	6.1	5.3
	3	0	2.6	1.2	0	0.6	0.1	0	0.7	0.6	0.4	0.7	0.5	0	1.4	0.9	0	2.4	0.5	2.7	3.1	2.9	2.7	3.1	2.8	3.6	3.8	3.7	3.7	4.7	4.1
	4	0	17	8.9	2.5	6.5	3.8	2.5	6.5	3.7	0	1.5	0.5	1.5	16	9.3	2.5	13	7.3	41	46	42	28	46	39	48	51	49	51	56	55
	1	0.2	0.3	0.3	0.2	1.2	0.5	0.15	1.2	0.5	0.3	0.5	0.4	0.2	0.8	0.4	0	1.8	0.5	1.3	2.2	1.7	1.3	2.2	1.6	2.3	2.9	2.5	3.3	4.1	3.6
786	2	2.5	11	5.3	2.5	13	6.8	2.5	11	3.8	0	1	0.4	2.5	9	6.4	0.8	6	4.3	12	16	14	13	16	14	18	20	18	23	25	24
	3	0	5.6	2.5	0.6	2.9	1.6	0	3.5	1.1	0.2	1.5	0.5	1	3.6	2.2	0.5	2.1	1.5	3.2	3.7	3.3	3.2	3.7	3.3	4.1	148	33	4.2	5.2	4.6
	4	0	5.2	2.7	2.4	3.4	2.8	0	5.2	3	0	1.4	0.3	3.4	3.4	3.4	2.4	5.4	3.5	16	18	17	9.6	18	15	19	21	20	22	24	23
	1	0.1	2	0.9	0	2.8	1.1	0.053	4	1.3	0	0.3	0.1	0.1	2.9	1.3	1	2.8	1.7	3.5	3.9	3.7	3.5	3.9	3.8	4	4.5	4.3	4.2	6	4.9
9 10 8	2	1.5	4.6	3	1.6	4.3	2.7	1.5	4	2.1	0	1.6	0.8	2.5	4.1	3.1	2.2	4.6	3.3	3	3.6	3.3	3	3.6	3.3	3.9	4.6	4.2	4.3	6.4	5
	3	0	3.2	1.6	0	2.5	0.7	0	2.3	0.3	0	0.2	0.1	0	1.8	0.4	0	2.1	0.7	3.7	4	3.9	3.7	4	3.9	4.3	4.6	4.5	4.6	4.9	4.8
	4	0.4	1.3	0.9	0.4	1.1	0.6	0.643	1.1	1	0	0.9	0.4	0.4	1.7	0.8	0.4	1.6	1.2	5.9	7.1	6.6	6.4	7.1	6.7	0.8	7.7	6.1	7.1	8.5	8
	1	0.2	0.8	0.4	0	0.4	0.3	0.205	0.9	0.4	0.2	0.2	0.2	0.2	1.4	0.6	0	0.6	0.2	1.3	1.7	1.5	1.4	1.7	1.6	1.9	2.1	2	1.4	2.4	1.9
11 6 8	2	0	1.4	0.4	0	1.1	0.4	0.04	1.1	0.2	0.4	0.9	0.7	0	1.7	0.8	0	2.5	1.5	2.8	3.1	2.9	2.8	3.4	3.1	3.4	34	9.6	3.4	4	3.6
	3	0.2	0.5	0.4	0.2	0.2	0.2	0.184	0.3	0.2	0	0.1	0	0.4	1.2	0.8	0.2	0.5	0.3	0.9	0.9	0.9	0.9	0.9	0.9	1.2	17	4.5	1.4	1.9	1.7
	4	0.1	2.4	1.8	1.4	1.9	1.7	0.074	1.9	1.9	0.5	0.6	0.5	0	2.4	1.6	1.4	2.4	1.9	1.6	2.2	1.8	1.7	2.8	2	2.9	3.3	3.1	2.4	4.5	3.6

* Average Relative Deviation (RD) of the best, worse and average solutions of 60 examples

* Each example is run 10 times

			3DFISAEI			3DBISAE			3DFISAE			3DBISAE			3DFI			3DBI			BI			FI			BISA			FISA	
		B	W	A	B	W	A	B	W	A	B	W	A	B	W	A	B	W	A	B	W	B	B	W	A	B	W	A	B	W	A
	1	0.8	3.3	2.5	0.5	2.5	1.7	0.773	3.5	1.2	0	0	0	0.1	2.6	1.2	0.4	3.2	1.6	1.1	1.4	1.3	1.1	1.4	1.4	1.4	1.9	1.7	2	2.7	2.2
10 10 10	2	0.5	1.8	0.9	0.6	2.1	1.2	0.455	1.1	0.7	0	1.2	0.7	0.1	1.1	0.5	0.5	2.7	1.7	1.3	1.5	1.5	1.3	1.5	1.4	1.8	2.5	2.1	2.4	3	2.7
	3	0.5	1.6	1	0.5	2.1	1.2	0.375	1.6	0.7	0	0.5	0.4	0	14	4.1	0.5	6	2.4	1.8	7	3.9	1.8	3.9	3	6.1	9.5	7.8	7.4	20	13
	4	0.2	1.8	1	0	0.5	0.4	0.267	2.3	0.4	0.3	0.4	0.3	0.6	1.7	1.4	0.5	2.3	0.9	1.1	1.5	1.2	1	2	1.5	1.6	1.7	1.6	1.4	2.8	2.3
	1	0.3	2.9	0.9	0	1.6	0.6	0.375	2.9	0.7	0	0.4	0.1	0.4	1.1	0.7	0.4	5.4	2.3	1.8	2.2	2	1.8	2.2	2	2.6	3.2	2.9	2.2	4.4	2.9
12 10 9	2	0.2	0.6	0.3	0.1	0.5	0.3	0.194	1.3	0.5	0	0.3	0.2	0.1	0.8	0.3	0.1	1.2	0.7	0.4	0.6	0.5	0.4	0.5	0.4	1.2	1.3	1.3	1.4	2.7	1.9
	3	1	2.3	1.5	0.9	1.7	1.4	0	2.8	0.2	0	1.2	0.6	1	2	1.6	1.6	2.7	2.1	1.5	2.2	1.8	1.7	2.6	2	2.4	3	2.6	2.6	4.8	3.7
	4	0.5	2.7	2	0.9	2.8	1.6	0.5	0.9	0.7	0	0.5	0.3	1.1	3	2.1	0.8	2.8	1.9	2.3	2.9	2.4	2.3	2.7	2.5	3	3.5	3.1	3.1	4.5	3.9
	1	0	1	0.8	0	1.5	1	0	1.5	1	0	1	0.3	0	1.6	0.9	0	1	0.7	3.7	6.6	5.3	3.7	5.7	5.2	5.9	7.2	6.5	6.7	9.8	7.6
13 8 9	2	0.6	1.2	0.8	0.4	1	0.6	0.385	1.2	0.5	0	0.4	0.2	0.4	1	0.7	0.4	0.8	0.6	1.2	1.8	1.4	1.4	2	1.8	1.8	2.3	2.1	2	2.8	2.4
	3	0.6	2	1.4	0.7	1	0.8	0.696	1	0.7	0	0.7	0.1	0	1	0.6	0	2.1	0.7	1.5	2	1.7	1.5	1.7	1.5	2	2.7	2.3	2.7	4.3	3.5
	4	0	5	1.6	0.3	3.8	2.7	0.25	3.5	1.3	0	3.5	1.2	0.3	3	0.8	0.3	4.3	1.8	13	17	15	13	17	16	19	24	22	21	28	24
	1	0.1	1.2	0.7	0.2	2.4	1	0	1	0.2	0	0.1	0	0.1	1.5	0.8	0.3	2.1	1	1.5	1.8	1.7	1.8	1.8	1.8	2	2.3	2.1	2.4	3	2.7
13 17 8	2	0	2	0.4	0	0	0	0	0	0	0	0	0	0	11	2.2	0	0	0	57	63	62	57	83	66	71	83	75	63	117	94
	3	2.6	5.3	3.5	1.3	5.1	3.7	0.846	3.5	1.5	0	0.8	0.3	0.6	6.3	3.3	2.8	5.8	3.7	0.8	1.3	1	0.8	1.6	1.2	2.5	4	3.3	5.4	6.9	6.2
	4	0.9	2.5	1.6	0.5	2.4	1.8	0.5	2.1	0.8	0	0.5	0.2	1.9	2.6	2.2	0	2.4	1.6	0.6	0.7	0.6	0.6	0.7	0.6	1.2	1.5	1.4	1.5	2.3	1.9

* Average Relative Deviation of the best, worse and average solutions of 60 examples

* Each example is run 10 times

APPENDIX- B3

RD Classified by Size and Density

			3DFISAEI			3DBISAEI			3DFISAEC			3DBISAEC			3DFI	
		B	W	A	B	W	A	B	W	A	B	W	A	B	W	A
	small	12.5	27.7	19.4	9.17	12.2	16.4	15.3	23.8	18.1	0	14.3	5.3	12.7	31.3	20.3
SIZE	medium	0.30695	4.49	2.41	0.8	3.24	1.81	0.5	5.29	2.72	0.15	1.14	0.49	0.94	3.9	2.32
	large	0.54856	2.32	1.31	0.43	1.94	1.23	0.35	1.89	0.7	0.02	0.72	0.31	0.41	3.38	1.46
	high	18.75	37.4	26.6	13.8	18.3	24.7	18.8	31.1	22.8	0	17.3	6.88	15.9	43.8	27.3
DENSITY	low	0.42312	3.91	2.26	0.31	1.52	0.99	2.35	3.68	2.77	0.02	2.63	0.77	1.82	4.51	2.73
	average	0.32859	4.23	2.27	0.76	3.07	1.71	0.48	4.92	2.48	0.13	1.08	0.47	0.91	3.66	2.19

			3DBI			BI			FI			BISA EC		FISAEC		
		B	W	A	B	W	B	B	W	A	B	W	B	W	A	
	small	15.25	29	19.4	18.2	37.6	30	18.2	36.8	31.4	38.7	84.3	39.6	50.4	44.2	44.2
SIZE	medium	0.78225	4.12	1.95	7.58	10.1	8.68	7.25	9.99	8.97	10.7	17.6	11.2	14.8	12.8	12.8
	large	0.52455	2.8	1.48	5.67	7.06	6.45	5.7	8.12	6.74	7.85	9.58	7.94	13.7	11	11
	high	19.75	40.4	26	22.5	42.3	34.6	22.5	41.1	34.9	44	112	45.3	61.5	52.3	52.3
DENSITY	low	1.9107	3.61	2.6	7.66	13.6	11.2	7.68	14.7	12.4	14.3	16	14.4	19.7	17.2	17.2
	average	0.77371	4	1.93	6.91	9.18	7.9	6.62	9.11	8.16	9.76	16	10.2	13.6	11.8	11.8

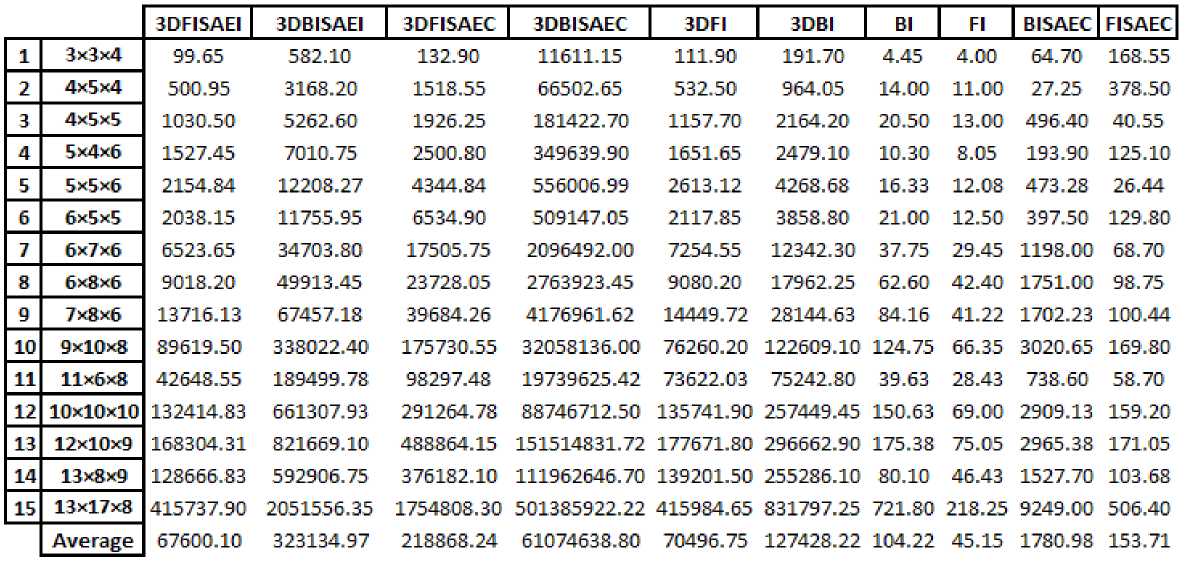
* Average RD classified by size and density of 60 examples.

APPENDIX- C

AVERAGE ITERATIONS

APPENDIX- C1

High Level Data



		3DFISAEI	3DBISAEI	3DFISAEC	3DBISAEC	3DFI	3DBI	BI	FI	BISAEC	FISAEC
1	3×3×4	99.65	582.10	132.90	11611.15	111.90	191.70	4.45	4.00	64.70	168.55
2	4×5×4	500.95	3168.20	1518.55	66502.65	532.50	964.05	14.00	11.00	27.25	378.50
3	4×5×5	1030.50	5262.60	1926.25	181422.70	1157.70	2164.20	20.50	13.00	496.40	40.55
4	5×4×6	1527.45	7010.75	2500.80	349639.90	1651.65	2479.10	10.30	8.05	193.90	125.10
5	5×5×6	2154.84	12208.27	4344.84	556006.99	2613.12	4268.68	16.33	12.08	473.28	26.44
6	6×5×5	2038.15	11755.95	6534.90	509147.05	2117.85	3858.80	21.00	12.50	397.50	129.80
7	6×7×6	6523.65	34703.80	17505.75	2096492.00	7254.55	12342.30	37.75	29.45	1198.00	68.70
8	6×8×6	9018.20	49913.45	23728.05	2763923.45	9080.20	17962.25	62.60	42.40	1751.00	98.75
9	7×8×6	13716.13	67457.18	39684.26	4176961.62	14449.72	28144.63	84.16	41.22	1702.23	100.44
10	9×10×8	89619.50	338022.40	175730.55	32058136.00	76260.20	122609.10	124.75	66.35	3020.65	169.80
11	11×6×8	42648.55	189499.78	98297.48	19739625.42	73622.03	75242.80	39.63	28.43	738.60	58.70
12	10×10×10	132414.83	661307.93	291264.78	88746712.50	135741.90	257449.45	150.63	69.00	2909.13	159.20
13	12×10×9	168304.31	821669.10	488864.15	151514831.72	177671.80	296662.90	175.38	75.05	2965.38	171.05
14	13×8×9	128666.83	592906.75	376182.10	111962646.70	139201.50	255286.10	80.10	46.43	1527.70	103.68
15	13×17×8	415737.90	2051556.35	1754808.30	501385922.22	415984.65	831797.25	721.80	218.25	9249.00	506.40
	Average	67600.10	323134.97	218868.24	61074638.80	70496.75	127428.22	104.22	45.15	1780.98	153.71

* Average of iterations of 15 different examples on 10 different approaches

APPENDIX- C2

Detailed Level Data

		3DFISAEI	3DBISAEI	3DFISAEC	3DBISAEC	3DFI	3DBI	BI	FI	BISAEC	FISAEC
SIZE		Cpu time	Cpu time	Cpu time	Cpu time	Cpu time	Cpu time	Cpu time	Cpu time	Cpu time	Cpu time
	1	119.6	604.8	152	12699.8	111.4	186.6	4	4	237.4	163.6
3×3×4	2	83.8	574.8	121	10333.2	86.4	185.6	4	4	6.8	109.6
	3	116.2	571	137.6	13301.4	112	186.2	5.8	4	7.6	145
	4	79	577.8	121	10110.2	137.8	208.4	4	4	7	256
	1	500.8	3058.6	1504.8	62247	530	1027	13	11	30	173
4×5×4	2	501.6	3256	1504.4	69365.2	552.2	667	17	11	29	323
	3	501.6	3299.6	1559.6	74169	517.8	1135.2	11	11	21	845
	4	499.8	3058.6	1505.4	60229.4	530	1027	15	11	29	173
	1	879.8	5013.6	1875.6	144796.2	980.2	1640.8	15	12.2	637	22.8
4×5×5	2	1343	5541	1908.8	192023.6	1410.2	2684.4	23	15.4	501	32
	3	933.4	5496	2017.6	171738.4	1224.8	2683.8	27	12.8	424.6	83
	4	965.8	4999.8	1903	217132.6	1015.6	1647.8	17	11.6	423	24.4
	1	1400	7326.8	2496	368035.8	1447.8	2382.4	7	7.8	235	12.4
5×4×6	2	1449.8	7532.2	2308	295268.8	2015.8	2402.8	16.6	10	331	17.4
	3	1453.8	7013	2860.8	385155.4	1256	2450.2	7	7	19	457
	4	1806.2	6171	2338.4	350099.6	1887	2681	10.6	7.4	190.6	13.6
	1	2109.8	11414.2	4295.2	550406.6	2014	3782.2	21	11	501	27.6
5×5×6	2	2616.6	13196.2	4318.6	555562.4	3109	5039.4	13	12.2	473	26
	3	1934.556	12337.67	4465.556	564978.8	2386.889	4412.333	14.33333	12.33333	432.1111	27.77778
	4	1958.4	11885	4300	553080.2	2942.6	3840.8	17	12.8	487	24.4

* Average of iterations of 60 different examples (for 10 runs) on 10 different approaches

		3DFISAEI	3DBISAEI	3DFISAEC	3DBISAEC	3DFI	3DBI	BI	FI	BISAEC	FISAEC
SIZE		Cpu time	Cpu time	Cpu time	Cpu time	Cpu time	Cpu time	Cpu time	Cpu time	Cpu time	Cpu time
	1	2301.6	11759.4	6245.4	476158.4	2278.2	3912	25	13.6	585	34.8
6×5×5	2	1950.8	12181	6212.6	565038.4	1985.4	3893	25	14	525	33.6
	3	1939.6	11326.4	6606.4	515709.4	1963.2	3783	15	11.4	449	29.8
	4	1960.6	11757	7075.2	479682	2244.6	3847.2	19	11	31	421
	1	7610.6	36997.8	17280.8	2165517	8214.4	13545.4	26.2	30.6	1256.8	78
6×7×6	2	7144.4	33292.4	17418.4	2224898	7548	12260.8	30.4	28.6	1156	68.6
	3	5615	33469.4	18057.8	1977356	5851.8	11133.2	64	33	1235.8	75.6
	4	5724.6	35055.6	17266	2018197	7404	12429.8	30.4	25.6	1143.4	52.6
	1	10503.4	50147.6	23421.6	2911942	9707.6	16492.8	45.8	43.6	1653	93.6
6×8×6	2	9122.8	45840.6	23682	2489628	9756.2	19568.8	79.4	43.8	1809.8	106
	3	8526	54814.4	24325.8	2701476	8117	20989.4	57	37.2	1837.8	99.2
	4	7920.6	48851.2	23482.8	2952647	8740	14798	68.2	45	1703.4	96.2
	1	10399.6	61366.4	47077.6	4093935	11125.2	28181.8	111.4	44.4	1882.6	103.4
7×8×6	2	15871.33	68512.33	37424.44	4264832	16555.89	27266.33	83.22222	31.66667	1830.333	100.5556
	3	13169.4	64728.6	36410.6	3799947	12995.4	26227.4	79.4	43.6	1459.8	96.4
	4	15424.2	75221.4	37824.4	4549132	17122.4	30903	62.6	45.2	1636.2	101.4
	1	69674.6	356840.4	176423.8	33670426	70329.8	128633.2	145	58.2	2818	167.2
9 ×10 ×8	2	74593.4	345558.6	171374.2	32688083	79959.8	147521.8	136	69.8	3115	171
	3	58037.4	308362.6	173439.6	33069371	88644	84828.8	145	77.6	3016.6	175
	4	156172.6	341328	181684.6	28804664	66107.2	129452.6	73	59.8	3133	166
	1	38069	180979.7	100865.6	20673492	172333	60946.7	29.5	21.2	751.3	46.3
11× 6× 8	2	43549.6	181232.2	98436.4	19888839	46446.8	72684.2	40	23	716.2	54.6
	3	47849	200766.6	96424	19086741	34136	93941.2	46	41.8	707.4	52.4
	4	41126.6	195020.6	97463.9	19309431	41572.3	73399.1	43	27.7	779.5	81.5
	1	114776.7	579032.9	240536.9	83164829	135174.4	193557.8	122.5	58.6	2921.5	160.8
10× 10× 10	2	132870.8	618698	307473.6	79808631	153740	193183.8	190	85	3070	152
	3	140108.2	665004	317068.2	93006545	141655.4	351968.4	118	72.8	2692	155.2
	4	141903.6	782496.8	299980.4	99006845	112397.8	291087.8	172	59.6	2953	168.8
	1	149213.8	795892.8	471591.4	1.63E+08	170751	226825.2	127	69	2935	162.6
12× 10× 9	2	198485.6	818354	483167.4	1.32E+08	204509.8	387901.8	217	78.4	2953	174.6
	3	178255.4	891607.4	520768.8	1.53E+08	186374.6	296385.2	190	77	2989	174
	4	147262.4	780822.2	479929	1.58E+08	149051.8	275539.4	167.5	75.8	2984.5	173
	1	143640.7	587564.2	366118.8	1.07E+08	141726.1	237171.2	37.4	43.7	1303	104.3
13× 8× 9	2	135626.4	660605.8	401377	1.18E+08	138757.6	322406.6	96.2	43.8	1619.4	104.6
	3	108721.6	608797.6	360440.8	98360777	151627.7	285723.6	90.6	51.6	1714.6	106.4
	4	126678.6	514659.4	376791.8	1.24E+08	124694.6	175843	96.2	46.6	1473.8	99.4
	1	465219.6	2193578	1911503	5.22E+08	413234.8	1128806	681	209.2	8759.4	524.4
13× 17× 8	2	477697.8	2198386	1792275	4.89E+08	506001.2	883810.2	463.4	174.6	8106.6	446
	3	354899.4	1875073	1739656	4.75E+08	430884.2	625861.6	898.6	253.8	10119.4	531.6
	4	365134.8	1939188	1575799	5.2E+08	313818.4	688711.6	844.2	235.4	10010.6	523.6

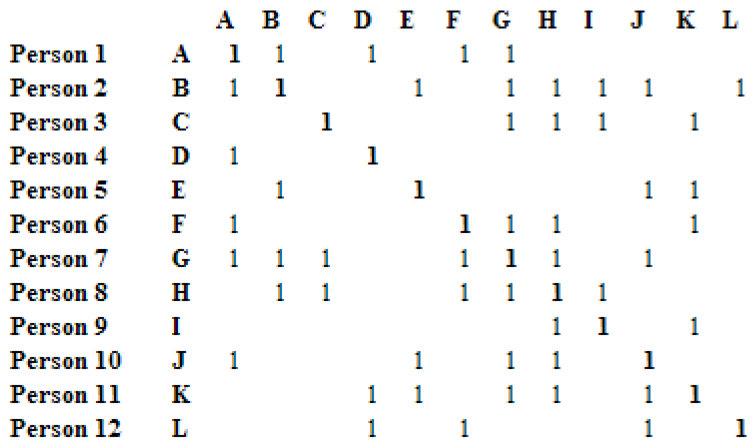
* Average of iterations of 60 different examples (for 10 runs) on 10 different approaches

APPENDIX- D

VARIOUS SOLUTIONS OF A ROBOT CODE

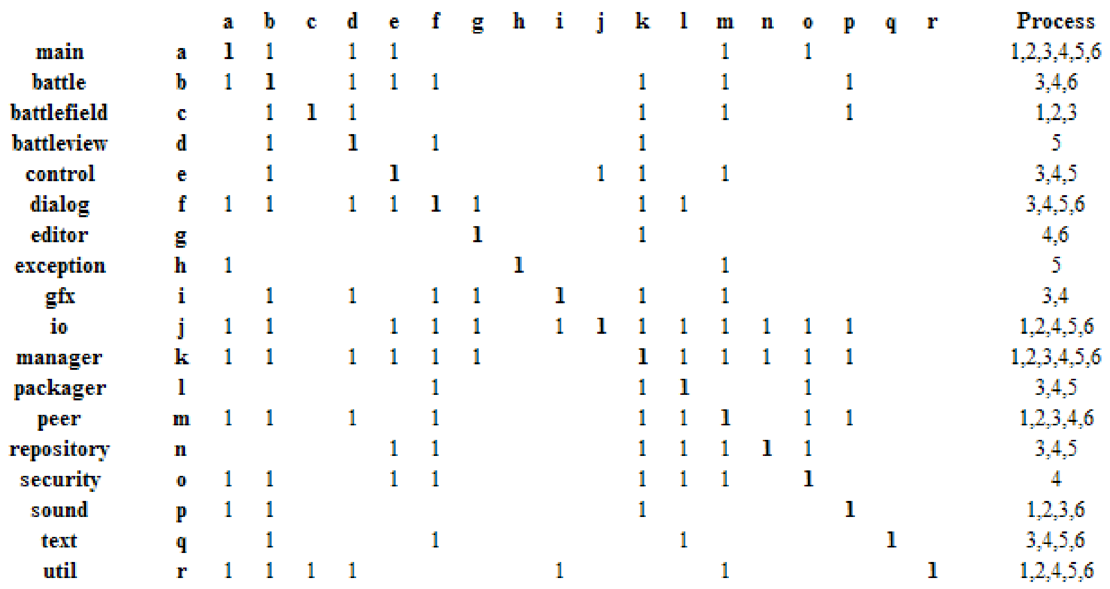
APPENDIX- D1

DSM Domains Structure



		A	B	C	D	E	F	G	H	I	J	K	L
Person 1	A	1	1		1		1	1					
Person 2	B	1	1			1		1	1	1	1		1
Person 3	C			1				1	1	1		1	
Person 4	D	1			1								
Person 5	E		1			1					1	1	
Person 6	F	1					1	1	1			1	
Person 7	G	1	1	1			1	1	1		1		
Person 8	H		1	1			1	1	1	1			
Person 9	I								1	1		1	
Person 10	J	1				1		1	1		1		
Person 11	K				1	1		1	1		1	1	
Person 12	L				1		1				1		1

Fig.D1.1: Organization structure DSM



		a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	Process
main	a	1	1		1	1								1		1				1,2,3,4,5,6
battle	b	1	1		1	1	1					1		1			1			3,4,6
battlefield	c		1	1	1							1		1			1			1,2,3
battleview	d		1		1		1					1								5
control	e		1			1					1	1		1						3,4,5
dialog	f	1	1		1	1	1	1				1	1							3,4,5,6
editor	g							1				1								4,6
exception	h	1							1					1						5
gfx	i		1		1		1	1		1		1		1						3,4
io	j	1	1			1	1	1		1	1	1	1	1	1	1	1			1,2,4,5,6
manager	k	1	1		1	1	1	1				1	1	1	1	1	1			1,2,3,4,5,6
packager	l						1					1	1			1				3,4,5
peer	m	1	1		1		1					1	1	1		1	1			1,2,3,4,6
repository	n					1	1					1	1	1	1	1				3,4,5
security	o	1	1			1	1					1	1	1		1				4
sound	p	1	1									1					1			1,2,3,6
text	q		1				1						1					1		3,4,5,6
util	r	1	1	1	1					1				1					1	1,2,4,5,6

Fig.D1.2: Product structure DSM with the corresponding processes

		1	2	3	4	5	6	Teams
Idea Generation	**1**	1	1	1				E,H,K
Requirement Analysis	**2**	1	1	1	1	1	1	E,H,K
Design	**3**	1	1	1	1		1	A,B,D,E,H,J,L
Programming	**4**			1	1	1		A,B,D,F,G,I,K,L
Testing	**5**	1			1	1	1	B,C,D,J,K
Implementation	**6**					1	1	A,D,E,F,H,K

Fig.D1.3: Process structure DSM with the corresponding teams

APPENDIX- D2

Starting from an Optimal Initial Solution

		A	B	C	D	E	F	G	H	I	J	K	L
Person 1	**A**	1	1		1		1	1					
Person 2	**B**	1	1			1		1	1	1	1		1
Person 3	**C**			1				1	1	1		1	
Person 4	**D**	1			1								
Person 5	**E**		1			1					1	1	
Person 6	**F**	1					1	1	1			1	
Person 7	**G**	1	1	1			1	1	1		1		
Person 8	**H**		1	1			1	1	1	1			
Person 9	**I**								1	1		1	
Person 10	**J**	1				1		1	1		1		
Person 11	**K**				1	1		1	1		1	1	
Person 12	**L**				1		1				1		1

Fig.D2.a: Optimal team domain (Optimal initial solution)

		a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	Process
main	a	1	1		1	1								1		1				1,2,3,4,5,6
battle	b	1	1		1	1	1					1		1			1			3,4,6
battlefield	c		1	1	1							1		1			1			1,2,3
battleview	d		1		1		1					1								5
control	e		1			1					1	1		1						3,4,5
dialog	f	1	1		1	1	1	1				1	1							3,4,5,6
editor	g							1				1								4,6
exception	h	1							1					1						5
gfx	i		1		1		1	1		1		1		1						3,4
io	j	1	1			1	1	1		1	1	1	1	1	1	1	1			1,2,4,5,6
manager	k	1	1		1	1	1	1				1	1	1	1	1	1			1,2,3,4,5,6
packager	l						1					1	1			1				3,4,5
peer	m	1	1		1		1					1	1	1		1	1			1,2,3,4,6
repository	n					1	1					1	1	1	1	1				3,4,5
security	o	1	1			1	1					1	1	1		1				4
sound	p	1	1									1					1			1,2,3,6
text	q		1				1						1					1		3,4,5,6
util	r	1	1	1	1					1				1					1	1,2,4,5,6

Fig.D2.b: Optimal product domain (Optimal initial solution)

		3	6	1	5	4	2	Teams
Design	**3**	**1**	1	1		1	1	A,B,D,E,H,J,L
Implementation	**6**		**1**		1			A,D,E,F,H,K
Idea Generation	**1**	1		**1**			1	E,H,K
Testing	**5**		1	1	**1**			B,C,D,J,K
Programming	**4**	1			1	**1**		A,B,D,F,G,I,K,L
Requirement Analysis	**2**	1		1			**1**	E,H,K

Fig.D2.c: Optimal process domain (Optimal initial solution)

APPENDIX- D3

Starting from a Random Initial Solution

		A	B	C	D	E	F	G	H	I	J	K	L
Person 1	A	1	1		1		1	1					
Person 2	B	1	1			1		1	1	1	1		1
Person 3	C			1				1	1	1		1	
Person 4	D	1			1								
Person 5	E		1			1					1	1	
Person 6	F	1					1	1	1			1	
Person 7	G	1	1	1			1	1	1		1		
Person 8	H		1	1			1	1	1	1			
Person 9	I								1	1		1	
Person 10	J	1				1		1	1		1		
Person 11	K				1	1		1	1		1	1	
Person 12	L				1		1				1		1

Fig.D3.a: Optimal team domain (random initial solution)

		a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	Process
main	a	1	1		1	1								1		1				1,2,3,4,5,6
battle	b	1	1		1	1	1					1		1			1			3,4,6
battlefield	c		1	1	1							1		1			1			1,2,3
battleview	d		1		1		1					1								5
control	e		1			1					1	1		1						3,4,5
dialog	f	1	1		1	1	1	1				1	1							3,4,5,6
editor	g							1				1								4,6
exception	h	1							1					1						5
gfx	i		1		1		1	1		1		1		1						3,4
io	j	1	1			1	1	1		1	1	1	1	1	1	1	1			1,2,4,5,6
manager	k	1	1		1	1	1	1				1	1	1	1	1	1			1,2,3,4,5,6
packager	l						1					1	1			1				3,4,5
peer	m	1	1		1		1					1	1	1		1	1			1,2,3,4,6
repository	n					1	1					1	1	1	1	1				3,4,5
security	o	1	1			1	1					1	1	1		1				4
sound	p	1	1									1					1			1,2,3,6
text	q		1				1						1					1		3,4,5,6
util	r	1	1	1	1					1				1					1	1,2,4,5,6

Fig.D3.b: Optimal product domain (random initial solution)

		6	4	1	2	5	3	Teams
Implementation	6	1				1		A,D,E,F,H,K
Programming	4		1			1	1	A,B,D,F,G,I,K,L
Idea Generation	1			1	1		1	E,H,K
Requirement Analysis	2			1	1		1	E,H,K
Testing	5	1		1		1		B,C,D,J,K
Design	3	1	1	1	1		1	A,B,D,E,H,J,L

Fig.D3.c: Optimal process domain (random initial solution)

APPENDIX- D4

Optimization in Isolation

		D	I	J	A	C	G	K	B	E	L	F	H
Person 4	D	1			1								
Person 9	I		1					1					1
Person 10	J			1	1		1			1			1
Person 1	A	1			1		1		1			1	
Person 3	C		1			1	1	1					1
Person 7	G			1	1	1	1		1			1	1
Person 11	K	1		1			1	1		1			1
Person 2	B		1	1	1		1		1	1	1		1
Person 5	E			1				1	1	1			
Person 12	L	1		1							1	1	
Person 6	F				1		1	1				1	1
Person 8	H		1			1	1		1			1	1

Fig.D4.a: Optimal team domain (Isolated optimization solution)

		b	e	h	k	a	c	f	j	l	m	n	o	i	q	d	g	p	r	Process
battle	b	1	1		1	1		1			1					1		1		3,4,6
control	e	1	1		1				1		1									3,4,5
exception	h			1		1					1									5
manager	k	1	1		1	1		1		1	1	1	1			1	1	1		1,2,3,4,5,6
main	a	1	1			1					1		1			1				1,2,3,4,5,6
battlefield	c	1			1		1				1					1		1		1,2,3
dialog	f	1	1		1	1		1		1						1	1			3,4,5,6
io	j	1	1		1	1		1	1	1	1	1	1	1			1	1		1,2,4,5,6
packager	l				1			1		1			1							3,4,5
peer	m	1			1	1		1		1	1		1			1		1		1,2,3,4,6
repository	n		1		1			1		1	1	1	1							3,4,5
security	o	1	1		1	1		1		1	1		1							4
gfx	i	1			1			1			1			1		1	1			3,4
text	q	1						1		1					1					3,4,5,6
battleview	d	1			1			1								1				5
editor	g				1												1			4,6
sound	p	1			1	1												1		1,2,3,6
util	r	1				1	1				1			1		1			1	1,2,4,5,6

Fig.D4.b: Optimal product domain (Isolated optimization solution)

		6	5	4	3	2	1	Teams
Implementation	**6**	**1**	1					A,D,E,F,H,K
Testing	**5**	1	**1**				1	B,C,D,J,K
Programming	**4**		1	**1**	1			A,B,D,F,G,I,K,L
Design	**3**	1		1	**1**	1	1	A,B,D,E,H,J,L
Requirement Analysis	**2**				1	**1**	1	E,H,K
Idea Generation	**1**				1	1	**1**	E,H,K

Fig.D4.c: Optimal process domain (Isolated optimization solution)

REFERENCES

Abdelsalam, H., Bao, H. (2006). A simulation-Based Optimization Framework for Product Development Cycle Time Reduction. *IEEE Transactions on Engineering Management,* 53, pp.69-85.

Bertsimas, D., & Tsitsiklis, J. (1993). Simulated annealing. *Statistical Science, 8*(1), 10-15.

Braha, D. (2002). Partitioning tasks to product development teams. Paper presented at the *Proceedings of ICAD.*

Browning, T. R. (2001). Applying the design structure matrix to system decomposition and integration problems: A review and new directions. *Engineering Management, IEEE Transactions on, 48*(3), 292-306.

Browning, T. R., & Eppinger, S. D. (2002). Modeling impacts of process architecture on cost and schedule risk in product development. *Engineering Management, IEEE Transactions on, 49*(4), 428-442.

Chinneck, J. W. (2006). Chapter 14: Heuristics for discrete search: Genetic algorithms and simulated annealing. *Practical optimization: A gentle introduction* (pp. 1-10). Carleston University, Otawa, Ontario K1S 5B6 Canada:

Clark, K. B., & Fujimoto, T. (1991). *Product development performance : Strategy, organization, and management in the world auto industry*. Boston, Mass.: Harvard Business School Press.

Clausen, J. (1999). Branch and bound algorithms-principles and examples. Dept. Comput. Sci., Univ. Copenhagen). , 1-30. (http://www.imm.dtu.dk/~jha/)

Danilovic, M., & Sandkull, B. (2005). The use of dependence structure matrix and domain mapping matrix in managing uncertainty in multiple project situations. *International Journal of Project Management, 23*(3), 193-203.

Danilovic, M., & Browning, T. R. (2007). Managing complex product development projects with design structure matrices and domain mapping matrices. *International Journal of Project Management, 25*(3), 300-314.

Eppinger, S. D. (1997). A planning method for integration of large-scale engineering systems. *International Conference on Engineering Design,* 199-204.

Glover, F., Taillard, E., & Taillard, E. (1993). A user's guide to tabu search. *Annals of Operations Research, 41*(1), 3-28.

Harmel, G., Bonjour, E., & Dulmet, M. (2006).Product, Processes and Organization Architectures Modeling: From Strategic Expectations to Strategic Competencies.

Johnson, D. S., Aragon, C. R., McGeoch, L. A., & Schevon, C. (Nov.- Dec., 1989). Optimization by simulated annealing: An experimental evaluation; part I, graph partitioning. *Operations Research, 37*(6), 865-892.

Lindemann, U., Maurer, M., & Braun, T. (2009). *The challenge of complexity* Springer.

Mohr, A., (2009) Applications of Stirling Numbers Involving Chromatic Polynomials (with T. D. Porter). *Journal of Combinatorial Mathematics and Combinatorial Computing,* 70,57 -64.

Meier, C., Yassine, A. A., & Browning, T. R. (2007). Design process sequencing with competent genetic algorithms. *Journal of Mechanical Design, 129*, 566. Retrieved from http://www.asme.org/terms/Terms_Use.cfm

Nijenhuis, A., & Wilf, H. S. (1978). Next partition of an n-set (NEXEQU). In W. Rheinboldt (Ed.), *Combinatorial algorithms for computers and calculators* (Second Edition ed., pp. 88-92). London: Academic Press, Inc. (London) LTD.

Osman, I. H. (1993). Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research, 41*(4), 421-451.

Osman, I. H., & Christofides, N. (1994). Capacitated clustering problems by hybrid simulated annealing and tabu search. *International Transactions in Operational Research, 1*(3), 317-336.

Osman, I., & Potts, C. (1989). Simulated annealing for permutation flow-shop scheduling. *Omega, 17*(6), 551-557.

Sharman, D. M., Yassine, A. A., & Carlile, P. (2002). Architectural optimisation using real options theory and dependency structure matrices.

Sridhar, N., Agrawal, R., & Kinzel, G. L. (1993). Active occurrence-matrix-based approach to design decomposition. *Computer-Aided Design, 25*(8), 500-512.

Suman, B., & Kumar, P. (2005). A survey of simulated annealing as a tool for single and multiobjective optimization. *Journal of the Operational Research Society, 57*(10), 1143-1160.

Tang, D., Zhang, G., & Dai, S. (2009). Design as integration of axiomatic design and design structure matrix. *Robotics and Computer-Integrated Manufacturing, 25*(3), 610-619.

Tang, D., Zhu, R., Tang, J., Xu, R., & He, R. (2010). Product design knowledge management based on design structure matrix. *Advanced Engineering Informatics, 24*(2), 159-166.

Thebeau, R. E. (February 2001). Knowledge management of system interfaces and interactions from product development processes. (Master of Science in Engineering and Management, Massachusetts Institute of Technology).

Yassine, A., & Braha, D. (2003). Complex concurrent engineering and the design structure matrix method. *Concurrent Engineering, 11*(3), 165.

Yassine, A., Falkenburg, D., & Chelst, K. (1999). Engineering design management: An information structure approach. *International Journal of Production Research, 37*(13), 2957-2975.

Yassine, A. A.(2004).An introduction to modeling and analyzing complex product development processes using the design structure matrix (DSM) method. *Urbana, 51.*