INTEGRATION ANALYSIS OF PRODUCT ARCHITECTURE TO SUPPORT EFFECTIVE TEAM CO-LOCATION

by

CARLOS IÑAKI GUTIERREZ FERNANDEZ
B.S., Mechanical Engineering
Massachusetts Institute of Technology, 1996

Submitted to the DEPARTMENT OF MECHANICAL ENGINEERING in Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1998

Signature of Author.....
Master of Science Program
Department of Mechanical Engineering
May 15, 1998

Certified by.....
Professor Steven D. Eppinger
Associate Professor of Management Science, Sloan School of Management
Thesis Supervisor

Accepted by.....
Professor Ain A. Sonin
Chairman, Department Graduate Committee

INTEGRATION ANALYSIS OF PRODUCT ARCHITECTURE TO SUPPORT EFFECTIVE TEAM CO-LOCATION

by

CARLOS IÑAKI GUTIERREZ FERNANDEZ

Submitted to the Department of Mechanical
Engineering on May 15 in Partial Fulfillment
of the Requirements for the Degree of
Master of Science at the
Massachusetts Institute of Technology

Abstract

Successful product development efforts are greatly facilitated through the use of integration analysis. Teams working on a product development project need to be brought together into clusters to address interactions between the functions or product elements they represent. This thesis presents a stochastic clustering algorithm to find such clusters in an efficient manner. The algorithm can find clustering solutions to architecture and organization interaction problems modeled using the design structure matrix method. The algorithm can be controlled to favor solutions with certain characteristics such as level of overlap, number of clusters, maximum number of teams per cluster, and emphasis on the level of interactions addressed by the clusters.

The difficulty to co-locate teams is measured by a coordination cost, which varies according to the composition of clusters. A mathematical model that minimizes the coordination cost to find the optimal solution for a given number of clusters has been developed. It has been used to measure the performance of the algorithm through a series of comparison tests. When the algorithm is run several times, the best solutions are reasonably close to an optimal solution.

As a sample application, the algorithm is used to analyze the architecture of an automotive cockpit system according to six dimensions of integration. A set of solutions with different number of clusters was generated.

Thesis Supervisor: Steven D. Eppinger
Title: Associate Professor of Management Science, Sloan School of Management

Key Words

clustering algorithm, co-location, design structure matrix, integration analysis, integrative mechanisms, integer program, product architecture, product development, product development process, system teams

Acknowledgements

This thesis is incomplete without recognizing those who made it possible. I want to thank the Jet Propulsion Laboratory and MIT for funding this research. I especially thank my advisor, Professor Steven D. Eppinger, for his advice, support, and friendship. Other professors and students at MIT offered me valuable insight and suggestions. In particular, I want to thank Professor Thomas L. Magnanti, and students who took part in a team presenting material of this research for a Systems Optimization class. Their feedback during the early part of this research was very useful. I thank MIT and all the people who make this institution a magnificent place to learn. I also thank Ford Motor Company for the real-life data provided for this thesis.

On a personal note, I want to thank my parents for their love, wise guidance, and support throughout my life. I thank my brother and sister for their love and affection. My friends, both those in Mexico, and the ones I have met during my studies at MIT have always given me support and encouragement, thanks a lot. I thank my professors, godparents, ants and uncles, and friends of my family for sharing their knowledge and experiences with me. Above all, I thank God for giving me the opportunity to improve myself, learn, and share the MIT experience.

Biographical Note

The author has been research assistant for Professor Steven D. Eppinger from the Sloan School of Management through MIT's Center for Innovation in Product Development since 1996. The author is a native of Puebla, Mexico. He has a Bachelor of Science Degree in Mechanical Engineering conferred by the Massachusetts Institute of Technology in June, 1996. Mr. Gutierrez has work experience with McKinsey & Company, Universidad de las Americas, MIT Artificial Intelligence Lab, and MIT Solar Car Club.

Address correspondence to:

Carlos Iñaki Gutierrez Fernandez
21 Sur 2730 esq. 31 pte.
Puebla, Pue. 72410 Mexico

Phone: (52-22) 47-16-82
Email: inaki@alum.mit.edu

The Center for Innovation in Product Development (CIPD)

"The Center for Innovation in Product Development (CIPD) is an interdisciplinary program devoted to the creation and deployment of breakthrough product development science, processes and tools. The Center has organized its research activities into four mutually supporting thrusts. Thrusts 1 and 2 (*Designing Successful Products* and *Information Based Product Development*) focus on understanding and improving major activities within the product development process. Thrusts 3 and 4 (*Enterprise Strategy* and *Accelerating Capabilities Improvement*) represent fundamental concerns of companies seeking to increase their product development performance. Research is validated in industry and is integrated into new educational offerings. The educational activities of the Center focus on curriculum integration of the product development process into both management and the engineering schools, new degree programs, short courses and seminars, and educational outreach. Our objective is to establish instruction in product development as a standard component of both management and engineering education throughout MIT and across the United States. In both research and education activities, we intensively involve our industrial collaborators as active partners. We will consider the Center successful if its students, faculty, and industrial partners, working together, generate significant progress in the development and diffusion of product development processes used throughout the United States."[1]

Information Based Product Development (Thrust 2) concentrates on developing a "better understanding of the information-intensive product development process, and will create more effective tools and methods to support product development activities. As a result of success in this research thrust, practitioners will be able to develop high-quality products faster and with less effort than is possible today."[2]

[1] CIPD, "Projects, Fall 1997", CIPD Internal Publication, Fall, 1997. p. 1
[2] CIPD, p. 21

Table of Contents

Chapter 1. Introduction

Overview of Thesis

In this thesis, we present a clustering algorithm that groups together elements or teams into clusters to facilitate the exchange of parameters or information in a product development effort. Refer to Figure 1-1 for a flowchart of this thesis.

Chapter 1 gives an overview for the need to integrate separate elements or teams to facilitate interactions in large product development projects. The Design Structure Matrix (DSM), a useful tool to map interactions is presented in this chapter. Chapter 2 presents the clustering algorithm that we have developed, and discusses previous algorithms that served as the basis for our algorithm. Chapter 3 analyzes the clustering problem formally in a mathematical form, and presents an integer program model that can find the optimal solution to this problem. Chapter 4 measures the performance of the algorithm through a series of comparison tests between the clustering algorithm and the integer program. Chapter 5 shows the use of the clustering algorithm in an industrial application, where different components are clustered together according to different metrics of the architecture of the product. Chapter 6 contains concluding remarks. It discusses the limitations of the clustering algorithm and possible future improvements.

After the conclusions, we have included a section for bibliography followed by Appendix A, which contains the code and instructions on how to use the clustering algorithm. Appendix B contains the comparison tests summarized in Chapter 4.

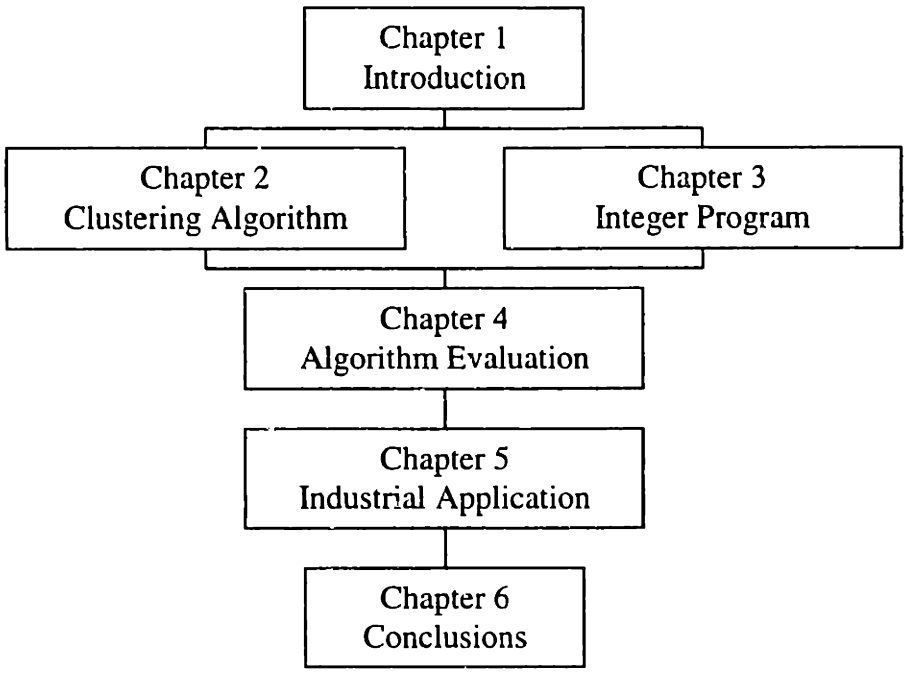


Figure 1-1. Thesis flowchart.

The Need for Integration Analysis[3]

The Product Development Process consists of five major activities: requirements development, design and integration, analysis, trades, and verification. Within each of these major activities are several sub-activities. One of the common threads of design consideration in many of these sub-activities is the optimization of interfaces or interactions between the various system elements. These interfaces occur in both the product itself, as defined in its conceptual stage or current state of development, as well as between the various teams or system teams working on the development of the product.

[3] Some material presented in this section was originally written for a Systems Optimization class at MIT on April 15, 1997. The team was composed by Paul Adamsen, Alison Davis, Carlos Iñaki Gutierrez, Burt LaFountain, Nestor Macias, Shawn Ritchie, Mike Rodeffer, and Nader Sabbaghian

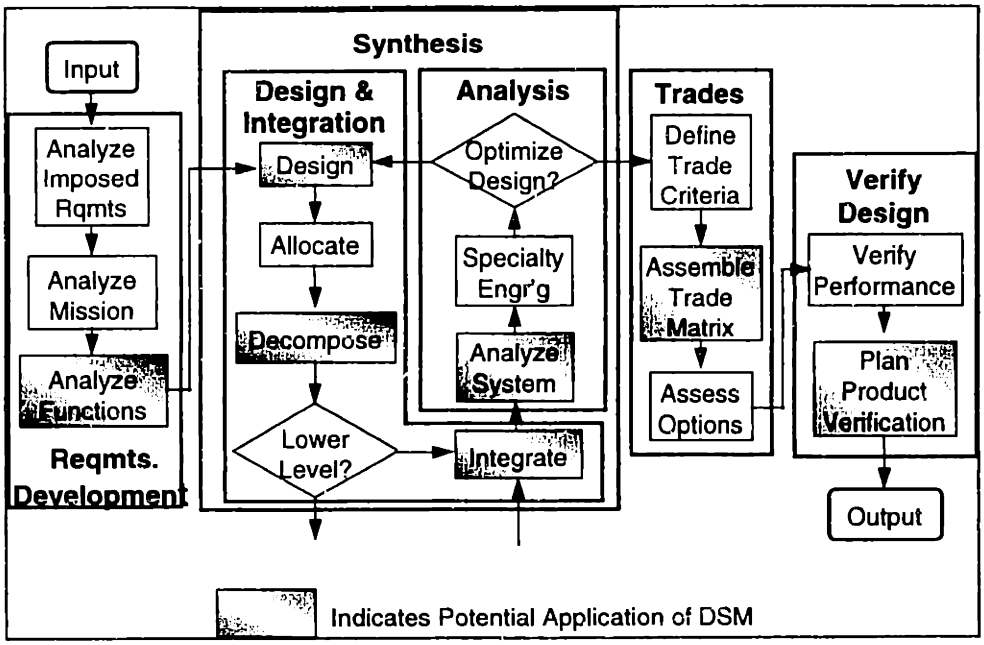


Figure 1-2. A generalized product development process.[4]

It is often the case that when problems occur in a system design, they occur at interfaces between system elements. Design teams in general do an excellent job of developing the components of the system. However, each component team is focused on its own set of tasks and is thus less sensitive or even unaware of the design issues of the other component teams or system elements. As a consequence, significant time and resources are spent on iteration to go back and correct problems that were not previously or formally addressed. A properly integrated development effort can mitigate these problems. When this integration task is optimized, considerable time and resources are saved, and better products are brought earlier to market.

Integration analysis involves the use of analytic tools to identify elements or teams of a product development effort, and integrate them into modules, sub-systems or system teams to address interactions between the parts involved. Integration is done by creating higher level entities that group together elements into clusters according to the interactions between the elements.

The architecture of a product is determined both by the elements that compose the product, and the way in which they interact with each other. Complex products become

[4] The figure includes potential uses of the Design Structure Matrix (DSM), presented later in this chapter.

valuable from the synergy of interactions between the elements that compose them.[5] By identifying such elements and rearranging them to facilitate the interactions between the elements, the product can be designed faster, and its performance enhanced. The different stages of the design process can be formally addressed by explicitly identifying and analyzing the elements taking part at that stage. The product architecture has a direct impact on the elements of a product, and is reflected in the organization of teams responsible for a particular task in the development of such product. Therefore, the elements might be physical or functional entities, and they can be the teams behind the design of particular parts, tasks, or functions.

The identification of elements and interactions between such elements is facilitated through the use of various tools such as the Design Structure Matrix. The grouping of elements into clusters is performed manually or by means of various optimization algorithms (two of which we present in this thesis). The objective when grouping elements together into clusters varies depending on the application at hand, but in general, the main goal is to address interactions between elements within clusters, minimizing the inter-cluster interactions. When dealing with teams, it is desirable to group them together into system teams, such that the number of interactions between different system teams is minimized. Likewise, with respect to technical activities such as functional analysis and design tasks, it is desirable to minimize interactions between different clusters of elements (modules) of the system, while addressing most inter-element interactions within the modules.

The Design Structure Matrix

Overview of the DSM

The Design Structure Matrix (DSM) is a useful tool for optimizing the composition of system teams or product development elements in terms of minimizing interfaces and extra-team interactions. As shown earlier in Figure 1-2, the DSM has potential applications in different stages of the design process: to analyze functions, facilitate the tracking of elements in a new design, decompose a system into elements,

[5] Source: Rechtin (1991), p. 29

integrate elements into modules or sub systems, and so forth. The DSM is primarily used to optimize the grouping of teams into system teams, or product elements into modules. It can also be used to find an optimal sequencing of tasks that need to be performed as part of a product development effort.[6]

The DSM is a square matrix where rows and columns list the same elements. The entries in the matrix record interactions between elements. Elements can represent components, tasks, or teams.

Figure 1-3 demonstrates the use of a DSM. Suppose we are developing a new product that involves four teams: A, B, C, and D. We list A, B, C, and D across the columns and down the rows. An "x" is placed in each entry to indicate an interaction between two teams. Reading across a row we can see from which other teams information must be passed to the team in that row. For instance, the third row in the figure shows that team C depends on both teams B, and D. Next, reading down the columns we understand which teams depend on the team in that column. From the fourth column we can see that both teams A and C depend on team D.

	A	B	C	D
A	A			x
B		B	x	
C		x	C	x
D	x			D

Figure 1-3. Sample DSM.

A powerful characteristic of the DSM is that it can be used to model complex development projects in a compact form. It can capture not only whether an interaction exists (by using an "x"), but also the strength of the interaction (by using numbers or distinct symbols). It can be used to visualize coupled or interdependent tasks (unlike the PERT tool). The DSM's unique capabilities make it an excellent tool to map interactions and improve the product development processes.[7] DSMs are used in many different system analysis problems such as systems engineering, project scheduling, and

[6] Steward (1981) pioneered the use of the DSM to analyze product development projects.
[7] Smith and Eppinger (1992) use the DSM as a tool to model the engineering design iteration. Morelli (1993) compares information needs in a project with actual communication patterns in an organization.

organizational planning. In general, the DSM tool can be used in three important types of situations encountered in the product development process[8]:

1. *Parameter-based*: Modeling system architecture based on parameter interrelationships.
2. *Task-based*: Modeling project schedule based on inter-task information flow.
3. *People based*: Modeling organizational structure based on information flow between groups.

Parameter-based Use of the DSM

DSM optimization methodologies can be effectively applied to aid the product development process during the system-level design phase. Once a concept has been identified, the functional elements of a product need to be arranged into modules or sub-systems, which in turn are grouped to form a system that meets the objective of the product. Using a DSM, the interrelationships between these functional elements can be recorded according to different parameters. Then, the elements can be rearranged to show sub-system or module interactions, while minimizing interactions among sub-systems.

To form the matrix, a level, or set of levels measuring different parameters has to be assigned to each interaction. Pimmler and Eppinger use four different parameters to analyze a climate control system, such parameters are shown in Table 1-1.

Table 1-1. Parameters used to analyze a climate control system.[9]

PARAMETER	DESCRIPTION
Spatial	A spatial-type interaction identifies needs for adjacency or orientation between two elements.
Energy	An energy-type interaction identifies needs for energy transfer between two elements.
Information	An information-type interaction identifies needs for information or signal exchange between two elements
Material	A material-type interaction identifies needs for material exchange between two elements.

[8] Source: Browning (1997).
[9] Source: Pimmler and Eppinger (1994).

Any of these parameters can be divided into levels, quantified and entered as numerical values in each entry of the DSM. Pimmler and Eppinger propose the five different levels for a spatial interaction shown in Table 1-2.

Table 1-2. Parameters used to analyze a climate control system.[10]

LEVEL	DESCRIPTION
Required (+2)	Physical adjacency is necessary for functionality
Desired (+1)	Physical adjacency is beneficial, but not absolutely necessary for functionality
Indifferent (0)	Physical adjacency does not affect functionality
Undesired (-1)	Physical adjacency causes negative effects but does not prevent functionality
Detrimental (-2)	Physical adjacency must be prevented to achieve functionality

These are a few of the parameters and levels that can be used to model the interactions across elements. Once one or different parameters have been identified, a DSM provides a system designer with the means to model a complex set of interrelationships, and a method to evaluate the effect of changing the architecture of that system by grouping elements into modules or sub-systems. Effectively finding an optimal configuration of elements involves the following steps:

1. Decomposing the system into elements.
2. Identifying interactions between elements and recording interactions in the matrix.
3. Reordering rows and columns in the matrix to group elements into sub-systems according to an objective function, such as minimizing the interactions between sub-systems.

A simple example of an optimized DSM matrix for a climate control system using material interactions alone is shown in Figure 1-4. Note that the elements could have also

[10] Source: Pimmler and Eppinger (1994).

been grouped together according to other parameters (e.g. spatial, information, or energy), or a combination of them (e.g. weighted average of different parameters).

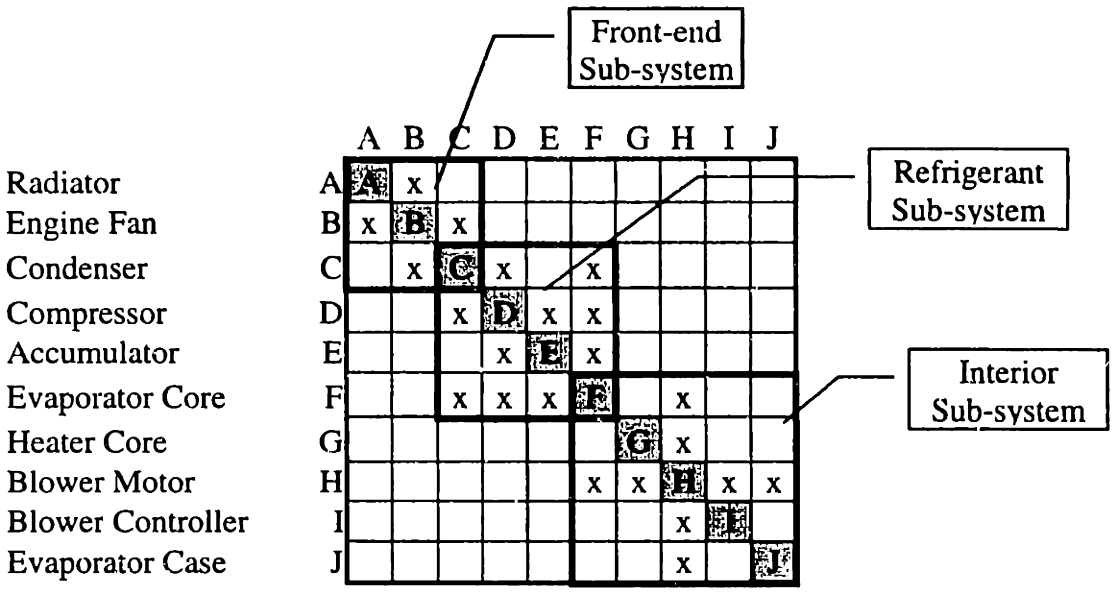


Figure 1-4. Sample parameter-based DSM showing elements of a climate control system grouped into sub-systems.[11]

In the optimized matrix, elements in the system have been arranged into three sub-systems or modules, which display the minimum amount of material interaction (no marked cells outside the sub-systems). Note that in the case of the Condenser and the Evaporator Core there exists overlap, i.e. both belong to more than one sub-system.

In using a parameter-based DSM early in the product development process, the interactions between sub-systems can be appropriately identified and addressed. The DSM tool is quite efficient to map and decompose large complex systems into appropriate sub-systems on the basis of a variety of parameter interactions. This analysis also "facilitates the a priori determination of the best organizational structure to develop a system. If the program organization mirrors the product architecture, sub-systems and elements grouped appropriately from a design standpoint will reduce inter-team integration issues from an organizational standpoint later."[12]

[11] This is part of a larger matrix presented by Pimmler and Eppinger (1994).
[12] Source: Browning (1997), p. 7

Task-based Use of the DSM

Planning is one of the biggest challenges in the development of complex products. In many cases, hundreds of tasks need to be identified and analyzed in order to compile a project schedule. The DSM tool can be effectively used to capture required product development tasks and task dependencies. The matrix representation is ideal for analyzing the relationships among tasks to see whether they are performed in series, or parallel; tasks that involve rework or iteration can be identified as coupled.

In the task-based DSM, tasks are ordered within the matrix according to their sequence of execution (from top to bottom row). Cells within the matrix display dependencies between tasks. A metric can also be used to indicate the degree of dependency among tasks (e.g. high, medium, or low information dependencies). A simple example of a task-based DSM, with seven arbitrary tasks is illustrated in Figure 1-5.

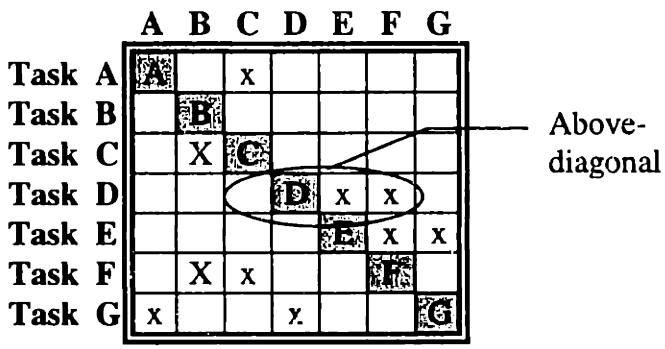


Figure 1-5. Task-based sample DSM.

In Figure 1-5, task D is dependent on information from tasks E, and F. In the current sequencing, task D will be performed before all the necessary information is available (before tasks E, F and G are executed). All dependencies that are above the diagonal present a feedback problem that involves iteration and delays. The objective is to minimize feedback dependencies. The matrix can be re-arranged to an optimal configuration by choosing the sequence of tasks that will minimize above-diagonal dependencies. Rearranging task A through G in the example, leads to the optimal task sequence shown in Figure 1-6, where the number of above-diagonal dependencies has been reduced from 5 to 1.

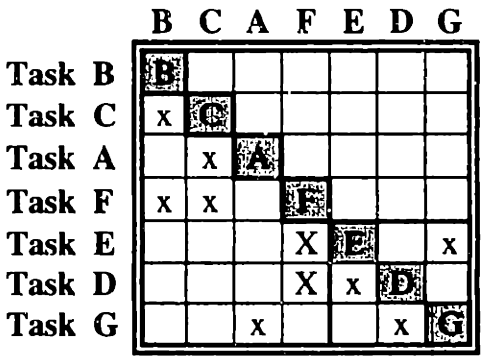


Figure 1-6. Rearranged task-based DSM to minimize feedback.

In the rearranged version, tasks B C, and A occur in series, tasks A and F are parallel, and tasks E, D, and G are coupled. This technique will ensure that the optimal task sequence is reflected in product development planning. It also ensures that delays due to task iterations are minimized. The matrix model also presents a simplified and integrated view of tasks and their relationships within the overall development effort. Other variations of the task-based DSM model include probabilistic representation of task dependencies and estimates of task duration.[13]

The clustering algorithm that we present in this thesis is not designed to find the optimal sequencing of tasks. It does not tell us how to sequence clusters, or tasks within clusters. However, it can be used to group highly interdependent tasks together into clusters. Such clusters help reduce the impact of iterations because most task dependencies occur within clusters, rather than outside them.

People-based Use of the DSM

The development of complex products requires a significant amount of information exchange among different teams working on the same project. Addressing communication issues among product development teams is critical to the success of each product development initiative. The people-based DSM can be effectively applied to organizational planning within product development. The DSM tool is used to analyze communication patterns among various teams in order to develop an efficient organizational structure.

The people-based DSM captures information flowing between product development teams. The goal here is to identify clusters of highly interactive teams through the reordering of the matrix. Information flow can be represented in a variety of ways. For example, communications can be evaluated with respect to their strength (through documentation, general meeting, face to face etc.), frequency (once a day, once a week etc.), and direction (one way, two way).

The goal here is to modify the matrix and cluster teams together into highly interactive groups known as system teams. A simple example of a people-based DSM is illustrated in Figure 1-7

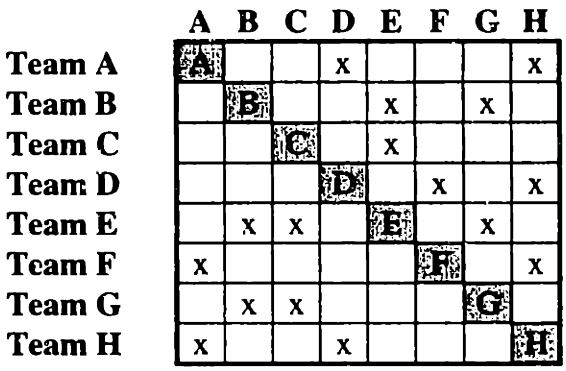


Figure 1-7. Sample people-based DSM.

According to Figure 1-7, team A has interactions with teams D, F, and H. By reordering the above matrix and optimized solution can be found (see Figure 1-8).

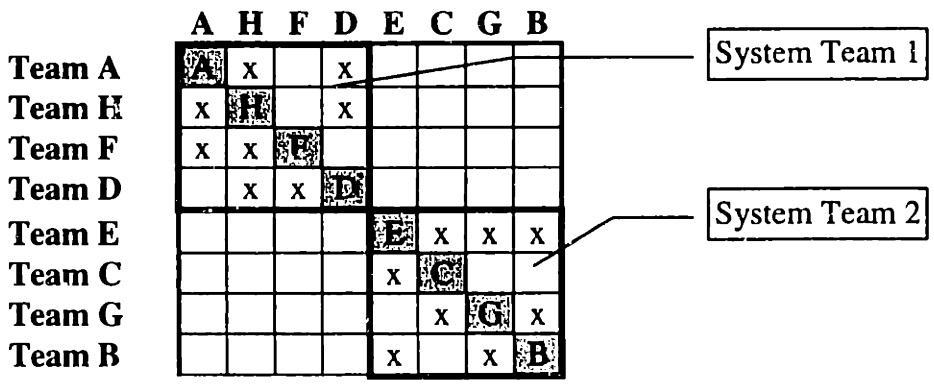


Figure 1-8. Reordered DSM showing two system teams.

[13] Refer to Smith and Eppinger (1992), Smith and Eppinger (1996), and Eppinger, Whitney, Smith, and Gebala (1994) for more information.

The optimized matrix was obtained by exchanging the position of groups B and H, and groups C and F. In this example, two system teams were distinguished as the best configuration: system team 1 with teams A, H, F, and D; and system team 2 with teams E, C, G, and B. As seen above, the system teams are a collection of teams where high levels of communication-needs were identified.

McCord and Eppinger used the DSM to help reorganize teams working on an engine development project. The project involved 22 product development teams. Such teams have been established around the product architecture of the engine, i.e. around the major components, or sub-systems of the engine. These teams meet regularly to address interactions and exchange information pertaining to the components that they are responsible for. The frequency of their meetings is depicted in the entries of the DSM according to four different levels: high (meeting regularly, perhaps daily), medium (meeting weekly or bi-weekly), low (infrequently, yet sometimes), and zero (never meeting).

The 22 product development teams were originally organized into 4 different system teams whose goal was to integrate the different teams and serve as a forum to bring up issues and technical conflicts between member teams. These system teams can help to discover challenging issues between teams early in the design process and avoid wasted time and resources later. The problem with the existing composition of system teams becomes apparent by looking at Figure 1-9. There are a lot of important interactions not addressed by system teams.

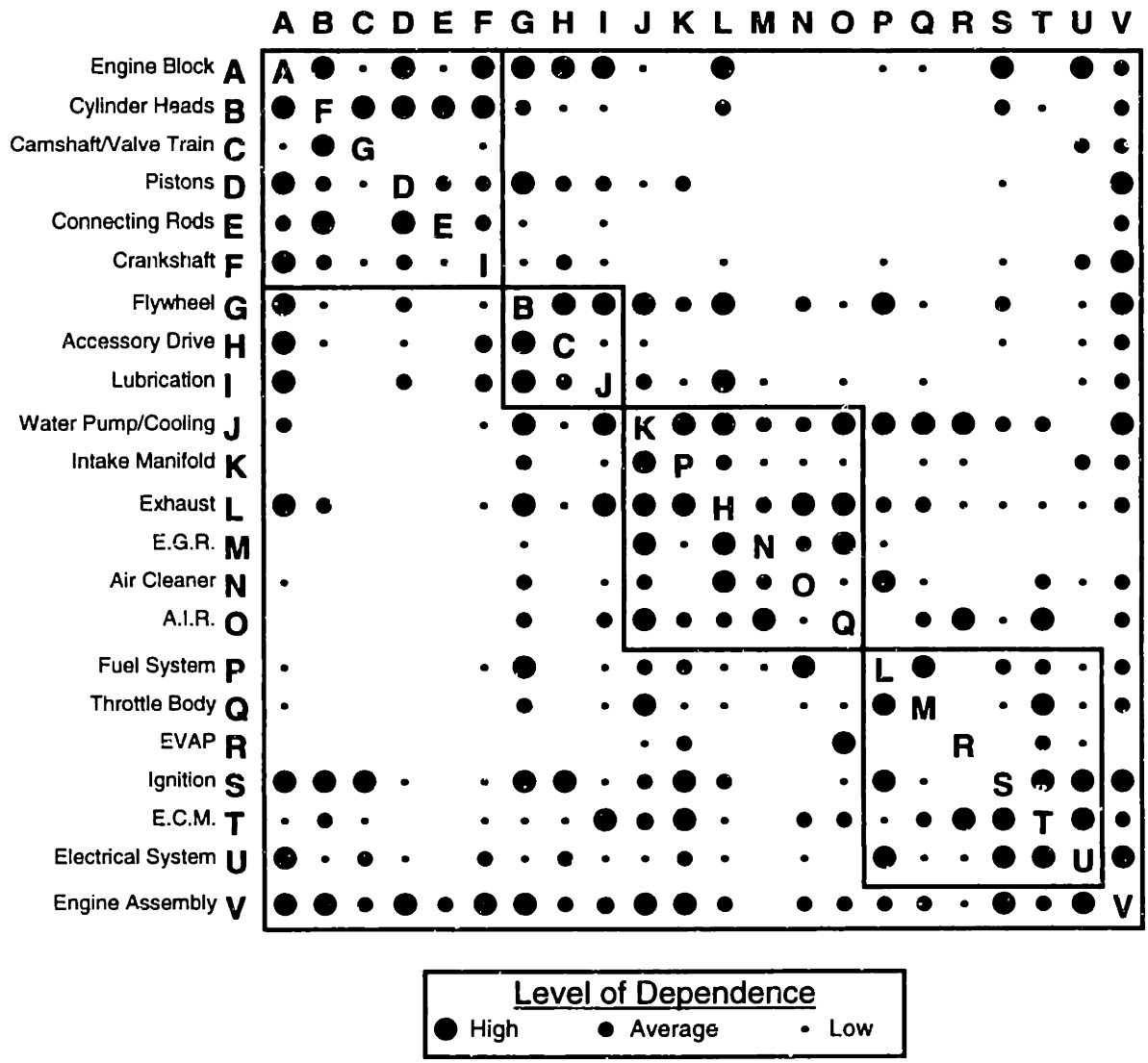


Figure 1-9. Engine development project showing existing system teams.[14]

McCord and Eppinger reorganized the composition of system teams by reconfiguring and overlapping teams. Their objective was to address more interactions within system teams and keep a small number of system teams. The proposed configuration is shown in Figure 1-10. Note that the proposed changes leave very few interactions outside the scope of the four system teams shown. All interactions to the right and bottom of the matrix are addressed by an additional system team involving teams H, S, T, U, and V. This system team serves as an "umbrella" for all other system teams.

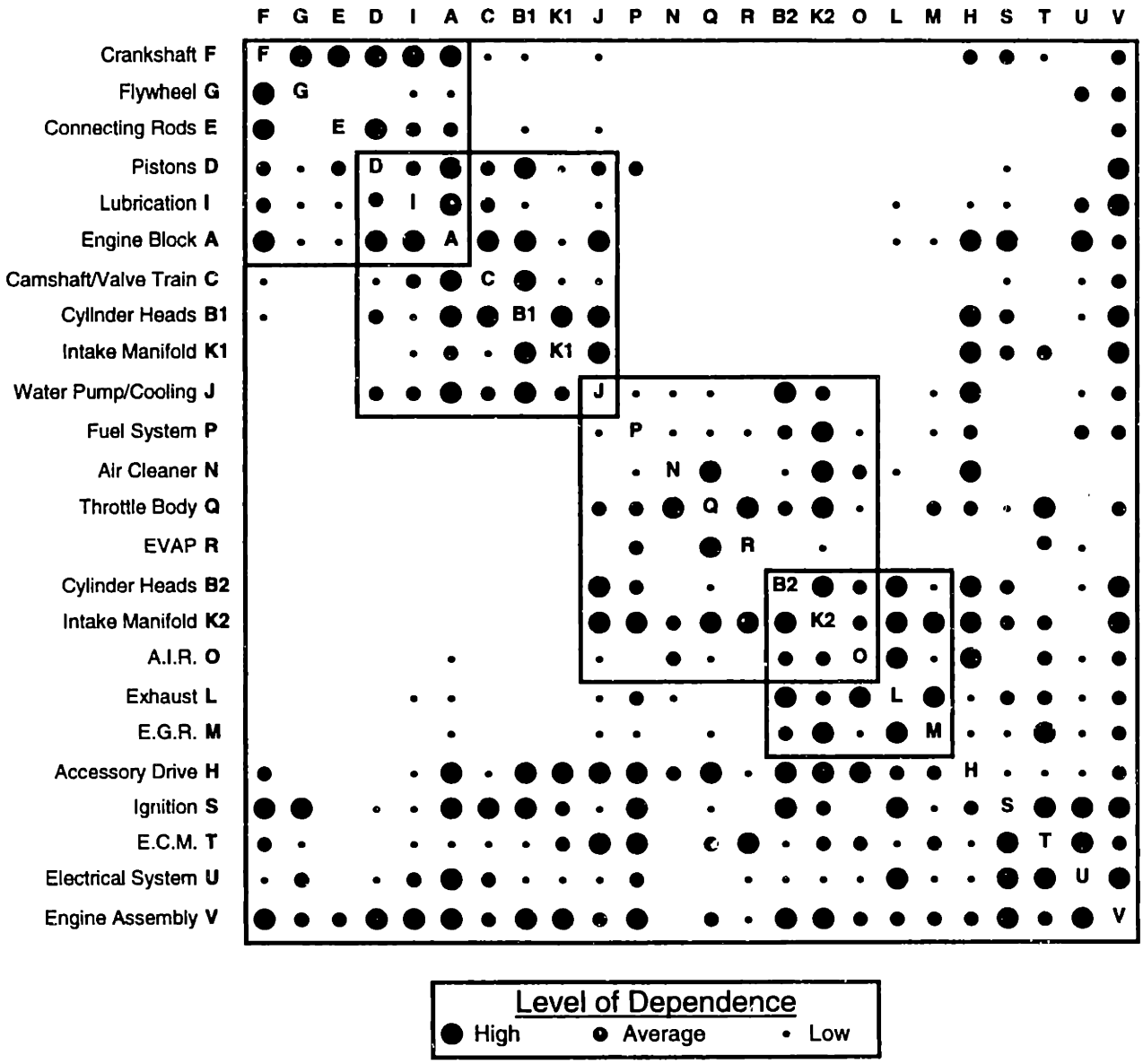


Figure 1-10 Reconfigured composition of system teams for the engine development project.[15]

Implementing Integration

Having reviewed the need for integration, and how the DSM is useful to model complex systems, we are left with the question of what to do with the results. Implementation involves taking action based on the results of the optimization process. What we do with the results depends on the type of application, and the type of

[14] Source: McCord and Eppinger (1993).
[15] Source: McCord and Eppinger (1993).

information that was used in the application. Management or system designers should evaluate the results before implementation. They need to analyze if the results are in accordance with issues not captured by the DSM, and perhaps incorporate changes to reflect these issues. Then implementation can take place in a variety of ways. The different integrative mechanisms should be analyzed and selected depending on the nature of the results.

When dealing with physical elements of a product, (such as in the climate control system) integration analysis produces different sub-systems that contain different functional elements. The different sub-systems can be analyzed and re-evaluated according to different parameters, or a combination of parameters. New versions of the product can be designed with functional elements grouped into modules. These modules can be interchangeable, and add flexibility and modularity to the product. Integration in this case is straightforward: the design should group elements in such a way that they are part of a single functional module.

The sequencing of tasks is aimed at finding the shortest way to achieve a series of tasks. Implementation of such results involves doing the tasks in the order in which the solution is optimal. Special attention should be placed on tasks that are on the critical path, i.e. those that if delayed would directly affect the whole development process. Furthermore, the tasks that are coupled should be analyzed to see if there are ways to reduce the iteration involved. As we mentioned before, in task-based problems we are not concerned about grouping task into clusters, but rather on finding the optimal sequencing of tasks.

The third major use of the DSM involved interactions between teams. There are a variety of integrative mechanisms to address interactions within system teams. McCord and Eppinger provide a good overview of some of the most useful ones. Browning has analyzed the effects of integrative mechanisms listed in Table 1-3 applied to integrated

product teams and functional groups. [16] He presents the integration mechanisms as part of six-step process required for design for integration.[17] The steps are:

1. Know system architecture.
2. Assign integrated product teams to system elements.
3. Systematically group integrated product teams.
4. Apply integrative mechanisms.
5. Manage interfaces.
6. Reassess status.

Table 1-3. Integrative mechanisms.[18]

Integrative mechanisms
Systems engineering and interface optimization
Improved information and communication technologies
Training
Co-location
"Town meetings"
Manager mediation
Participant mediation
Interface management groups
Interface contracts and scorecards

The use of integrative mechanisms is one of the steps required in the implementation process (the first three steps are facilitated by using the DSM and optimizing the formation of clusters or system teams). Managing interfaces and reassessing the status of the entire process are necessary for a successful implementation stage.

[16] McCord an Eppinger (1993) mention some integrative mechanisms not studied by Browning: Management Hierarchy, Heavy Weight Project Manager, Conflict Resolution Engineers and Liaison Roles, Task Forces, Technically Independent Teams, and Engineering Liaisons. Clark, et. al (1992) focus on the role of Heavy Weight Project Managers.

[17] Walsh (1990) studied failed product development efforts, where the lack of an adequate organizationally-driven development was to blame. He describes four key concepts (instead of six) that should be observed in order to launch a new product: (1) a holistic philosophy of new product development where all functional departments are involved, (2) date making and keeping, (3) education, and (4) the integration role of a new product integration manager.

[18] Source: Browning (1996).

Browning's study of integration mechanisms has some important implications for people-based integration problems[19]:

- The organizational structure of integrated product teams should mirror the product architecture as closely as possible.
- Since this is not completely possible, a systematic methodology should be used to group integrated product teams into system teams and to determine how integration will occur within these levels.
- Co-location is an excellent integrative mechanism, although many do not utilize it in its most effective form. Constraints on co-location force the use of alternative integrative mechanisms.

The clustering algorithm that we present in this thesis is a method that, given the mapping of inter-team or inter-element interactions through the use of the DSM, generates system teams or modules in a systematic way by optimizing an objective function. The algorithm is geared towards solving people-based DSMs, because it deals with positive interactions only. However, it can also be used to form modules of elements if negative interactions are treated as non-existent interactions. Once these highly interactive clusters are identified, appropriate integrative mechanisms can be designed to ensure effective communication. Co-location is the method of choice for people-based DSMs, or for teams that are closely mirroring the architecture of a product (teams identified with particular elements of a product). A representative from each of the teams attends system team meetings to address interactions within system teams.

[19] These are just some of his findings. We list only these because they are more relevant for the topic of this thesis.

Chapter 2. Clustering Algorithm

In this chapter we present the algorithm that allows us to obtain efficient system team configurations that reflect the product architecture of a product. We begin by reviewing clustering algorithms. In particular, we discuss an earlier version of the algorithm developed by John Idicula, and some of the shortcomings of his procedure. Then, we present our algorithm, with a description of its operation and how to control the results it produces. Finally, we present an example to better understand the operation of the algorithm.

Previous Work

The goal of a clustering algorithm is to group elements together into clusters. The rules of how this clustering is performed vary from application to application, and so do the type of solutions obtained.

Hartigan reviews the basic approach of clustering and discusses different applications of clustering algorithms. Clustering algorithms have been applied in a variety of applications such as taxonomy, computer graph theory, international trade, manufacturing operations optimization, and product development. The original goal of clustering was to find similarity between elements and group them together based on a threshold of similarity between the elements. There are several methods to cluster elements together including similarity coefficient, sorting, cost based, search (iterative improvement), and genetic algorithms.[20]

Biles et. al. also give a good overview of cluster analysis. They use computer simulations to evaluate cluster analysis as a tool for designing cellular systems using several performance measures.[21] Other recent applications of cluster analysis include pattern recognition using fuzzy logic. The goal in pattern recognition is to identify distinct objects (clusters) in an image. The image is broken down into pixels, and each pixel is compared with neighbor pixels to check for similar features. Clustering algorithms applied to this problem aim at grouping together pixels that represent a single

[20] Source: Hartigan (1975).

object. Recently some clustering algorithms in fuzzy logic have incorporated the distinct feature of allowing an element to be a member of more than one cluster.[22]

Genetic algorithms can be used in a variety of problems by combining different solutions into a new hybrid solution. Several rules on how individual solutions are combined, and built in randomness in the process of mixing solutions, allow genetic algorithms to explore better solutions. Genetic algorithms can be used to the clustering problem when a clear objective function is defined, and some solutions have been previously generated.[23]

Most clustering algorithms used other means of data representation rather than a DSM. Used means include directed graphs, hypergraphs, matrices and so forth. Pimmler and Eppinger use the DSM to reorder elements with an algorithm that optimizes a distance penalty.[24]

The clustering algorithm that we present on this thesis is based on an algorithm first developed by John Idicula in 1995.[25] In his thesis, Idicula addresses two problems faced in concurrent engineering, where he deals with tasks that have to be performed in a product development project. The first problem is to "identify the sets of mutually interdependent tasks in a product development effort," and the second one is to "determine clusters of tasks in a large set of mutually interdependent tasks."[26] He solves the first problem with a Block Determining Algorithm that searches for mutually interdependent teams. His algorithm assumes an underlying directed graph model for the development effort, and uses a depth-first-search technique to solve the problem. Once he has identified the tasks, Idicula solves a second problem using the DSM. He captures the interactions between the tasks that have been previously identified, and clusters tasks together with a clustering algorithm. It is this second algorithm that we have used as the basis for the algorithm that we present in the next section.

---

[21] Source: Biles, et al. (1991).
[22] Windham presents a geometrical fuzzy clustering algorithm that allows for membership of one member in more than one cluster, but the number of clusters has to be determined beforehand. Published in Bezdek, et. al. (1992).
[23] For an example of genetic algorithms refer to Kusiak (1993), or Altus (1995). Rogers (1996) applies genetic algorithms to task-based DSMs, and gives a good overview of genetic algorithms in general.
[24] See Pimmler and Eppinger (1994).
[25] As a student at NTU in Singapore, John Idicula worked with Prof. Eppinger at MIT. Refer to Idicula (1995) for his algorithm.
[26] Idicula (1995), p. 33

Idicula's work centers on individual tasks that need to be performed as part of a product development project. These tasks may have interdependencies with each other, and can be quantified and captured in a Design Structure Matrix. The algorithm developed by Idicula groups the "project tasks into clusters that are loosely connected with each other, while each cluster consists of densely connected inter-coupled tasks."[27] To find such clusters he uses a stochastic algorithm that iteratively attempts to decrease the value of a total coordination cost function. The algorithm can be run several times to produce different cluster configurations.

The total coordination cost function is the aggregate coordination cost of tasks. Each coordination cost takes into account the strength of interdependencies between two tasks, and the number of tasks in the smallest cluster that contains both tasks. The specific coordination cost equations are listed in Equations 2-1 and 2-2.

$$Total\ Coordination\ Cost = \sum_{i=1}^{n} CoordinationCost(Task_i), \qquad \textbf{Eq. 2-1}$$

and

$$Coordination\ Cost(task_i) = \sum_{j=1}^{n} (TPM(i,j) + TPM(j,i)) * size_{i,j}^{\ 2}, \qquad \textbf{Eq. 2-1}$$

where

$size_{i,j}$ is n, the total number of tasks if i & j do not belong to the same cluster, or *size* of the smallest cluster containing both i and j otherwise

$TPM(i,j)$ is the value of the interaction in the Design Structure Matrix.

The procedure to find a clustering arrangement is shown in Figure 2-1. The algorithm randomly selects a task and calculates a bid from clusters. The highest bid is chosen, and if there is an improvement in the total coordination cost, the task is added to the bidding cluster. This process continues until, after several attempts, there is no further improvement in the coordination cost. Since the algorithm we present later in this section is based on Idicula's algorithm, we leave the detailed description for the next section.

[27] Idicula (1995), p. 44

We should mention that there are many other methods to cluster data, when the size and number of clusters are known.[28] In our case however, the number and size of clusters are inherent in the structure of the project, and are usually not known in advance, therefore traditional clustering approaches fail to solve DSM clusterings. Idicula's algorithm is innovative in this sense because it produces clusters whose contents and size are not known in advance. It also allows one element to be a member of more than one cluster, as opposed to the common practice of having one element in just one cluster.[29]

```
Procedure Cluster;
Begin
    Input the weighted task precedence matrix;
    Form initial solution;
    Do
        Repeat rep times{
            1. Select a task t;
            2. Accept bids for task t from the clusters;
            3. Determine the highest bid;
            4. If the bid is acceptable, modify the clusters;
        }
        5. Determine whether the clusters are stable;
    Until clusters are stable;
Output the clusters;
End; {Cluster}
```

Figure 2-1. Outline of the algorithm developed by Idicula.[30]

Idicula's algorithm was the first successful attempt to stochastically produce clusters from a Design Structure Matrix. However, his algorithm was cumbersome to use and the user had little control over the type of solutions obtained. This prompted us to write a more flexible and practical algorithm incorporating new control features and using a new platform to expedite the solution of problems. The old version of the algorithm was written as a single file in C with no subroutines. The input matrix had to be saved as a separate file, then the program was run and the output cluster matrix had to be read from a separate output file. The new version of the algorithm allows the user to solve problems directly from Excel. The code for the algorithm, in case the user wants to

[28] See Hartigan (1975).
[29] Idicula (1995), p. 44
[30] Idicula (1995), p. 47

modify it, has been rewritten with separate subroutines for each separate function such as the coordination cost, the bidding function, and the modification of clusters. This separation of functions makes it easier to understand and follow the steps of the algorithm; it also allows the user to incorporate new parameters or changes in the algorithm. The operation of the algorithm and the type of solutions obtained can be easily controlled by adjusting eight different parameters. Obtaining and visualizing a cluster solution takes less than a couple of seconds for a matrix with 30 tasks. With the previous algorithm it could take ten minutes to do the same. By adjusting the eight control parameters, many solutions can be explored in a matter of seconds because of the Excel interface and the background processing of the algorithm. For the code of the new algorithm and instructions on how to use it in Excel, refer to Appendix A.

One final note before presenting the algorithm: Idicula's work centers on tasks. He develops a process to find clusters that address interdependencies across tasks. Our work centers on forming clusters (called system teams for people-based problems) that address the interactions of teams, rather than tasks. We deal with a Design Structure Matrix to represent interactions between teams, instead of interactions between tasks. The distinction is trivial as far as the DSM representation is concerned. However, the interpretation of the coordination cost of a task, and that of a team differs. The new coordination cost attempts to measure the cost of addressing interactions between teams by concentrating on different issues faced by those teams. We will see how in the next section.

The Algorithm

In this section we present the algorithm. First, we discuss the total coordination cost function, and how it addresses specific issues faced by system teams. Then, we talk about the data and parameters passed to the algorithm. We continue with a presentation of the method of operation of the algorithm. Finally, we discuss the effect of incorporating some randomness into the algorithm through simulated annealing. Doing so allows the algorithm to potentially find better solutions.

Total Coordination Cost

The algorithm is built around an objective function called the total coordination cost. The total coordination cost attempts to capture the following observations in a mathematical form[31]:

1. *It is more convenient to address interactions between teams formally in a system team, rather than ignoring them hoping that they will be addressed informally by the teams alone.* When interactions between teams are not formally addressed through effective communication, it is usually observed that the development effort experiences more iterations or failure because of poor exchange of information.[32] Reducing iteration shortens developing time and helps in two important ways, it reduces development costs, and it brings the product earlier to market allowing the company to sell the product before the competition.[33]
2. *The time or cost of addressing an interaction is proportional to the frequency or importance of the interaction.* An important or more frequent interaction requires more attention, more resources, or more commitment between the interacting teams. The interaction can be one or a combination of different metrics such as: frequency of interactions, amount of information to be passed, importance of interaction, amount of energy exchange, type of spatial interaction, etc. Regardless of the type of interaction, we expect this interaction to be quantified into relative numbers, with the value of the interaction proportional to the importance, frequency, etc. Therefore, an interaction with higher value will have a higher coordination cost.[34]

---

[31] These observations are not perfect. They are models of reality that allow us to express them in mathematical form. We have included footnotes of supporting or related research where appropriate.

[32] Szakonyi (1990) analyzes interactions in R&D projects and comments: "good communication ultimately comes down to two or a few people talking[...]any relationships that consist primarily of a great flow of paper back and forth should be suspect." (p. 44). Walsh (1990) studies failure in new product development projects and finds that "miscommunication across functional department lines" was one of the factors to blame (p. 32). He mentions that integration managers should "ensure that the new product needs of all lines of the operation are being adequately addressed" (p. 35).

[33] Hoedemaker (1995) also talks about the need for good communication between integrated teams in product development: "faulty communication can introduce design flaws which remain undetected until an integration test is conducted. Correction of defects makes necessary rework of the design modules and this can extend the project completion time" (p. 12).

[34] Kahn (1996) studies interaction between departments involved in a product development effort and describes interdepartmental contacts "as lasting only as long as a certain meeting remains in session or information is exchanged; such contact is temporary as all formal contacts between departments incur costs" (p. 140).

3. *It is easier for teams to interact in smaller groups, rather than large ones.* The fewer teams that are part of a system team, the easier a system team will accomplish its goal of facilitating the explicit interactions between member teams. In other words, the fewer people working together, the more productive they will be.[35]
4. *The difficulty of managing a system team and the effectiveness to address the interactions between member teams increases with the number of teams. Presumably this increase is not linear, but closer to quadratic.* This observation is closely tied to the previous one. Here we deal with the type of penalty for having larger system teams. There are pros and cons for having larger system teams. The previous point deals exclusively with explicit interactions, which, given a larger system team are more difficult to address. We believe that this difficulty grows quadratically with the number of teams. However, it could be argued that if teams A and B have an explicit interaction to address in a system team, where team C is also present, C could somewhat benefit from witnessing the A-B interaction, despite not having explicit interactions with A, or B, or both, i.e. team C gets insight into the broader picture of the development effort and learns from interactions that are not directly related to its team.[36] This benefit could be reflected by having a less than quadratic penalty for the size of the system team. The algorithm that we present can be fine tuned to reflect the beliefs of the system manager regarding this issue and adjust how much this point is a concern when forming clusters (system teams).
5. *For individual teams, the cost of being a member of system teams increases with the number of system teams addressing an interaction.* This last point reflects the fact that if a team were to attend different system team meetings to address an interaction, the cost will be proportional to the number of system teams where the interaction is addressed.[37] Note that this point was not incorporated into Idicula's version of the clustering algorithm. Because he dealt with tasks, rather than teams, he was

[35] Hoedemaker (1995) finds that concurrency has its limits, because "the returns from additional simultaneity diminish, turn negative and eventually increase expected project completion time" (p.16)
[36] Clark (1992) discusses the benefits of having a person or group of people responsible for similar subsystems over various development efforts because the "functions and subfunctions capture the benefits of prior experience and become the keepers of the organization's depth of knowledge" (p. 13).
[37] Clark (1992) mentions that there is a "risk of allowing core team members to be assigned to multiple projects [because] they are neither available when their inputs are most needed nor as committed to project success as their peers" (p. 21).

concerned about the cost of an interaction between tasks only in the smallest cluster where such interaction was addressed.

For each team in the DSM the algorithm calculates a coordination cost. Then the sum of the coordination costs for each team gives a total coordination cost. Equations 2-3 and 2-4 show the coordination cost for a team *i*.

If both teams *i* and *j* are in any cluster *k*, then

$$Coordination\ Cost(team_i) = \sum_{j=1}^{size} (DSM(i,j) + DSM(j,i)) * \sum_{k=1}^{Cl} cl\_size(k)^{pow\_cc}, \quad \textbf{Eq. 2-3}$$

else (if no *k* cluster contains both *i* and *j*, the entire DSM acts as cluster containing *i* and *j*)

$$Coordination\ Cost(team_i) = \sum_{j=1}^{size} (DSM(i,j) + DSM(j,i)) * size^{pow\_cc}, \quad \textbf{Eq. 2-4}$$

where

size is the size of the DSM, i.e. the number of teams in the DSM.

DSM(i,j) is the value of the interaction or dependency between teams *i* and *j*. Note that when i=j, DSM(i,j)=0, and DSM(j,i)=0, because the diagonal entries in the DSM do not list interactions (they can be used to represent the team name for that row or column).

Cl is the maximum number of clusters or system teams that the algorithm explores, in the algorithm *Cl* is equal to *size*, i.e. the algorithm can come up with as many as *size* system teams.

cl_size(k) is the number of teams contained in cluster *k*.

pow_cc is a parameter that controls the type of penalty assigned to the size of the cluster in the coordination cost (2 implies a quadratic relationship).

Equation 2-5 is the expression for the total coordination cost. This objective function is the expression that the algorithm attempts to minimize.

$$Total\ Coordination\ Cost = \sum_{i=1}^{size} Coordination\ Cost(Team_i). \quad \textbf{Eq. 2-5}$$

It is important to note that Equations 2-3, 2-4 and 2-5 differ from Equations 2-1 and 2-2 developed by Idicula in two important aspects. Idicula sums the values of the interactions over the smallest cluster where both teams *i* and *j* belong, whereas we sum these interactions over all the clusters that contain both teams. This change enhances the effectiveness of the algorithm at finding optimal or nearly optimal solutions, and discourages the formation of clusters that are similar in contents. We also introduce a variable power coefficient *pow_cc*, rather than a fixed power of 2. By varying *pow_cc* we can control the formation of clusters by the algorithm.

The coordination cost equations address the concerns we mentioned above in a specific mathematical expression. Let us review how the equations deal with these concerns.

1. *It is more convenient to address interactions between teams formally in a system team, rather than ignoring them hoping that they will be addressed informally by the teams alone.* This is reflected by assigning a high default coordination cost for interactions outside clusters equal to the product of the value of the interaction, times the size of the whole DSM raised to a power. The coordination cost will be very high when interactions, particularly important ones, are not addressed by a cluster. The cost function assigns a cost equivalent to having the entire DSM as a system team that addresses those interactions not addressed by any cluster.
2. *The time or cost of addressing an interaction is proportional to the frequency or importance of the interaction.* This is reflected in the numerical values of the above and below diagonal entries of the DSM: *DSM(i,j)* and *DSM (j,i).*
3. *It is easier for teams to interact in smaller groups, rather than large ones.* This is reflected when the value of the interactions is multiplied by the size of the cluster or system team. The cost is proportional to the number of teams in the cluster.
4. *The difficulty of managing a system team and the effectiveness to address the interactions between member teams increases with the number of teams. Presumably this increase is not linear, but closer to quadratic.* When we raise the size of the team to *pow_cc* we are accounting for this point.

5. *For individual teams, the cost of being a member of system teams increases with the number of system teams addressing an interaction.* This is reflected in the fact that we sum over all system teams or clusters where an interaction is addressed.

Data and Parameters

The data and parameters passed to the algorithm determine the behavior of the algorithm and the type of result that is obtained. The data is the Design Structure Matrix listing the interactions between teams in a numerical form. The parameters control how the algorithm explores and finds a solution.

The data in the Design Structure Matrix should quantitatively list the values of the interactions. The user can use values such as 0.15, 0.3, 0.5, 1, 3, etc. for weak values and correspondingly larger ones for stronger interactions. For non-existing interactions, the corresponding value should be zero. There can be many levels of interactions, i.e. the same matrix can have three, or five, or ten different numbers that correspond to different strengths. We suggest that the user think about the relative importance of interactions when assigning numbers. For example if there are only two levels of interactions: strong and weak, the user should think whether two weak interactions added together are more, less or equally important than a single strong interaction. So, for the three cases we could have the following corresponding pairs of values: 0.3-weak and 1-strong, 1.5-weak and 2-strong, or 1-weak and 2-strong. The algorithm cannot handle negative interactions. Therefore it can handle "attractions" between teams, but "rejections" should be valued just like non-existing zero interactions.

There are eight different parameters used to control the algorithm.[38] Here we briefly discuss what they control and how they can be used to affect the outcome of the algorithm. Some parameters "encourage" a certain type of solution, but do not guarantee that such will occur because of the stochastic nature of the algorithm. When we discuss the effect of a certain parameter on the behavior of the algorithm, the reader should keep this in mind. Some of these parameters will be easier to understand once the remaining sections of this chapter have been read. The parameters passed to the algorithm are:

[38] Only *rand_accept* was present in the algorithm developed by Idicula. The new parameters provide direct control over the performance of the algorithm.

- *pow_cc* This parameter controls the type of penalty assigned to the size of the cluster in the coordination cost (2 implies a quadratic relationship). Try 2 when running the algorithm for the first time on a new problem.
- *pow_bid* This parameter is similar to the previous one, except that it is used in the bidding function explained later. Typical values range from 0-3 with a high value penalizing formation of large clusters. Try 2 when running the algorithm for the first time on a new problem.
- *pow_dep* This parameter is also used in the bidding function. Typical values range from 0-2, with high values emphasizing high interactions during the bidding process. Try 2 when running the algorithm for the first time on a new problem.
- *max_Cl_size* This parameter prevents the formation of clusters containing more teams than *max_Cl_size*. If the user wants to restrict the number of teams in a cluster, this parameter should be adjusted to the desired number. Otherwise, try setting it equal to *size* when running the algorithm for the first time on a new problem.
- *rand_accept* Setting it to *N* tells the algorithm to proceed with changes 1 out of approximately every *N* times, despite the fact that there is no improvement in coordination cost. Try a value between half and twice the size of the DSM when running the algorithm for the first time on a new problem.
- *rand_bid* Setting it to *N* tells the algorithm to take the second highest bid instead of the highest one 1 out of approximately every *N* times. Try a value between half and twice the size of the DSM when running the algorithm for the first time on a new problem.
- *times* The algorithm will attempt (*times* x *Size*) times to pick a task and form clusters before checking for system stability. For example if this variable is 2 and we have a DSM with 10 teams the algorithm will loop 20 times before checking for stability. Try 2 when running the algorithm for the first time on a new problem.

- *stable_limit* The algorithm will have to loop at least (*stable_limit* x (*times* x *Size*)) unsuccessful attempts (no improvement in coordination cost) before ending. It is recommended to set it equal to 2 when running the algorithm for the first time on a new problem.

The Methodology of the Algorithm

The basic methodology of the algorithm is to initially assign each team in the DSM to a cluster or system team and calculate a total coordination cost. Then a team is randomly chosen and assigned to a cluster that has "strong interactions" with the team (we will define "strong interactions" later in this section). If assigning such team to a cluster reduces the total coordination cost, the change is made permanent. Empty clusters, repeated clusters and clusters contained entirely in other clusters are deleted. The algorithm repeats this process until, after several attempts, there is no improvement in the total coordination cost. There are several random features built into the algorithm that let clusters be formed despite not showing "strongest interactions" or an improvement in coordination cost. This built-in randomness improves the ability of the algorithm to obtain final solutions with lower total coordination cost.

A flowchart of the algorithm can be seen in Figure 2-2. The most important features of the algorithm are included in the flowchart. We will now describe these and other features of the algorithm.[39]

[39] Refer to Appendix A for the explicit C code and instructions on how to use the algorithm.

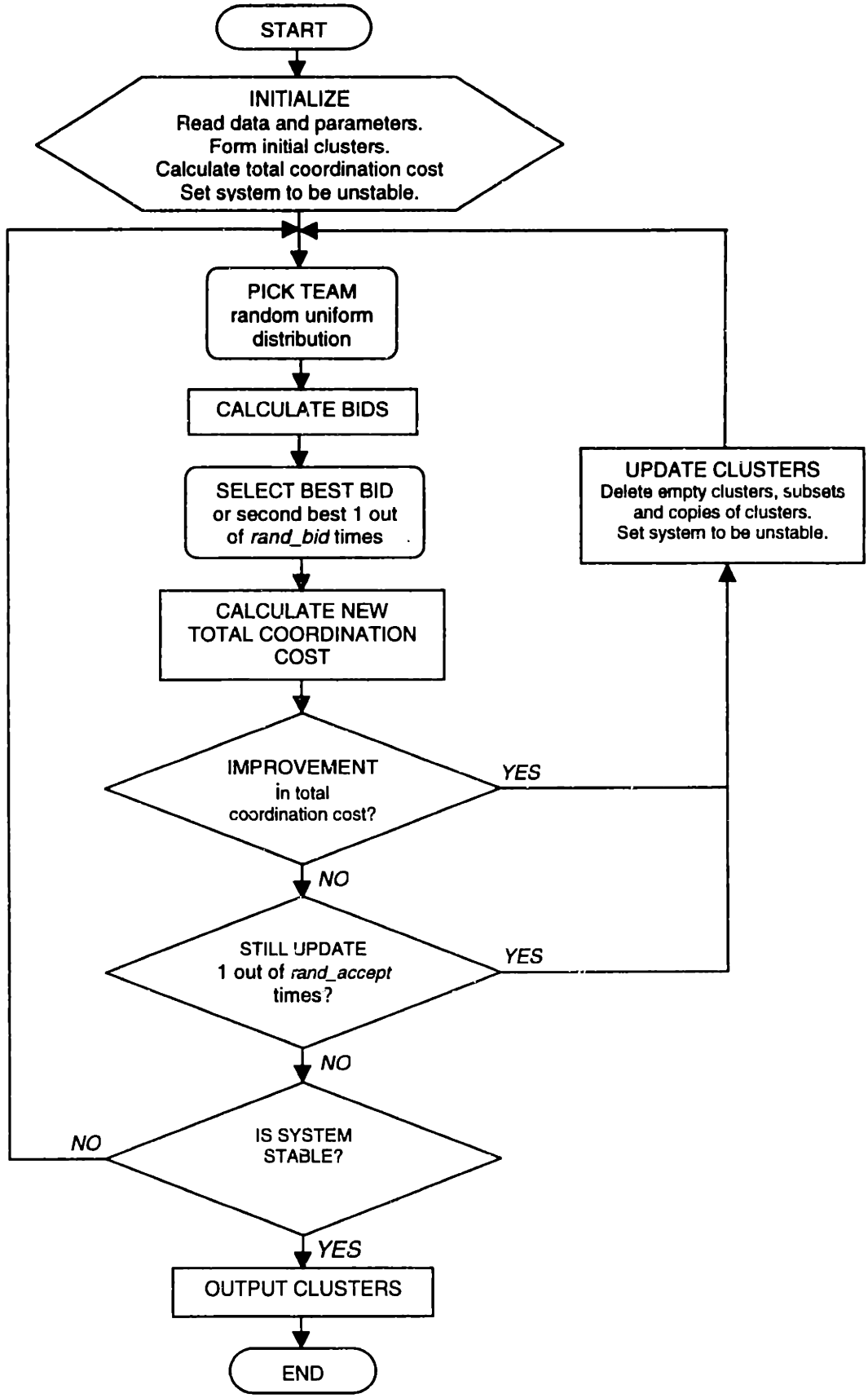


Figure 2-2. Flowchart of the clustering algorithm.

The algorithm starts by reading the data and parameters discussed before. Then, the algorithm assigns each team to a cluster. There are initially as many clusters as there are teams. With this configuration, an initial total coordination cost is calculated. Such cost is equal to the number of interactions multiplied by the size of the DSM raised to the parameter *pow_cc*, because no interaction is yet addressed by any cluster. The initial coordination cost will be high. Therefore, any cluster formation will achieve a reduction in coordination cost. The system is then set as unstable, i.e. it is set to repeat a clustering procedure until it cannot improve the total coordination cost after repeated trials.

The algorithm randomly selects a team. Any team has the same probability of being selected. Once a team is chosen, the algorithm calculates a bid from each cluster. Such bid is a measure of how "strong" team members of a cluster want to interact with the selected team. Equation 2-6 shows the exact form of the bid function.

$$Bid(cluster_k, task_t) = \sum_{j=1}^{size} \frac{((DSM(t,j) + DSM(j,t))^{pow\_dep} * CL\_MAT(k,j)}{cl\_size(k)^{pow\_bid}} \qquad \textbf{Eq. 2-6}$$

where

$Bid(cluster_k, task_t)$	refers to the bid from cluster k for task t.
size	is the size of the DSM, i.e. the number of teams in the DSM.
DSM(t,j)	is the value of the interaction or dependency between the randomly selected team t and team j.
cl_size(k)	is the number of teams contained in cluster k.
pow_dep	controls the importance given to strong interactions over weak ones. For example, a value of 0.5 makes different interactions more similar, while a large value like 3 increases their difference.
pow_bid	controls the value of a bid depending on the size of the clusters. A large value like 3 discourages the formation of large clusters and encourages the formation of clusters with similar sizes, whereas when using a value as small as 0 cluster size becomes unimportant for the bid and clusters can have a large range of sizes.
CL_MAT(k, j)	is a 0-1 variable. It takes the value of 1 when task j is present in cluster k.

Given a selected task *t*, the algorithm calculates a bid for each cluster. The bid is directly proportional to the interactions from the members of a cluster with team *t*, and inversely proportional to the size of the cluster. The parameter *pow_dep* controls how the bid places greater emphasis on strong vs. weak interactions, while the parameter *pow_bid* controls how much the size of the cluster affects the bid. The parameter *max_Cl_size* causes the bid from a cluster of such size to equal zero regardless of the interactions with task *t*. Doing so effectively prevents a task *t* to be grouped with such cluster.

Then, the algorithm selects the highest or second highest non-zero bid. The algorithm will on average select the second highest bid 1 out of every *rand_bid* times. So, on average, the highest bid will be chosen *rand_bid* - 1 times. For example if *rand_bid* is 10, the highest bid will be selected 90% of the times, and the second highest bid will be chosen 10% of the times.

Team *t* is temporarily assigned to the cluster with the selected bid. If there is an improvement in total coordination cost, the change will be made permanent, otherwise another random process takes place. One out of every *rand_accept* times the change is still made, despite the lack of improvement in coordination cost.

When a change is made, the algorithm analyses the composition of clusters to delete clusters that have identical contents, clusters that have empty contents, and clusters that are subsets of other clusters. Cluster A is a subset of cluster B if all teams present in A are also members of B (B can have more teams). Then the system is set to be unstable, and a new team is randomly selected.

When no change occurs, a new task is selected and the process repeats itself until, after several attempts, the algorithm achieves no improvement in coordination cost. The specifics of how this occurs are easier to understand with the code summary of Figure 2-3[40].

[40] Note that the summary code includes additional details on the way the algorithm reaches stability and comes to a stop. These details were left out of the flowchart for simplicity.

```
START
INITIALIZE (Read data and parameters. Form initial clusters. Calculate total coordination
cost. Set system to be unstable: system = 0)
WHILE system is less than stable_limit) {
    REPEAT size x times {
            PICK TASK
            CALCULATE BIDS (random uniform distribution)
            SELECT BEST BID (or second best bid 1 out of rand_bid times)
            CALCULATE NEW TOTAL COORDINATION COST
            if    IMPROVEMENT in total coordination cost
                    UPDATE CLUSTERS
            or if  STILL UPDATE(1 out of rand_accept times) then
                    UPDATE CLUSTERS
    }
    system= system + 1
}
OUTPUT CLUSTERS
END
```

Figure 2-3. Code summary for the algorithm.

We have highlighted in bold the specifics of how the algorithm stops. The process of selecting a task is repeated in the inner REPEAT loop *size* x *times* times. The outside WHILE loop is repeated for as long as the inner loop achieves decreases in the total coordination cost. Once no decreases are found it attempts to improve the cost *stable_limit* times more. Therefore, there will be at least *size* x *times* x *stable_limit* attempts to improve the coordination cost before the algorithm finishes.

Simulated Annealing

In two different steps of the algorithm, the decision on what to do next step is not determined through the available data, but is made instead through a random process. Such process is known as simulated annealing. By incorporating simulated annealing into the decision process, the algorithm can reach solutions that it would otherwise have left out.

Parameters *rand_bid* and *rand_accept* control two important steps in the algorithm. The first one affects the probability that either the first or second best bid is taken. Parameter *rand_accept* controls the likelihood that cluster changes will be made when there is no reduction in the total coordination cost. In both cases simulated annealing introduces random changes in the steps taken by the algorithm, changes that would not have taken place otherwise. The effect of simulated annealing is controlled by

either parameter to affect the probability that one of the two possible outcomes will occur. The solution space explored by the algorithm is increased, and the likelihood of being "trapped" in a sequence of steps that end up in a sub-optimal solution is minimized. Figure 2-4 attempts to depict the effect of simulated annealing in a graphical manner.

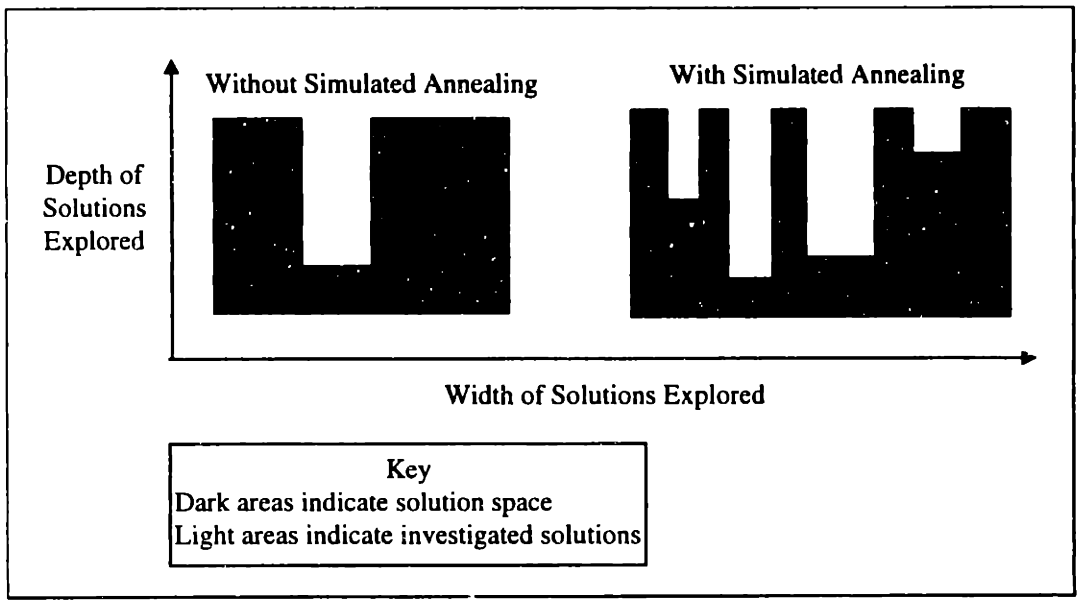


Figure 2-4. Solution space explored with and without simulated annealing.

In some respects simulated annealing offers the algorithm the ability to "think out of the box." Without some method to explore other potential solutions besides selecting a team at random, the results of the algorithm would be limited by the highest bid and strict improvements in coordination cost. The result would be a strong function of the first few random selections of tasks, because once the algorithm started down one path there would be no way for it to begin investigating alternative paths.

Running the Algorithm

Perhaps the best way to understand how the algorithm works is through a simple example. We have selected a small problem with seven teams to show the important features of the algorithm. Real life problems can have dozens or even hundreds of teams; however, the way the algorithm operates is still the same.

Figure 2-5 contains 15 matrices, the top 5 matrices are DSM representations of the same problem. The bottom 10 matrices are cluster configurations, with each row

representing a cluster, and each column representing a team. The *Initial DSM* shows the interactions between 7 teams, with light-gray and dark-gray depicting weak and strong interactions respectively. The *Ordered DSM* shows the same information, but the order of the teams has been changed to easily visualize three alternative clustering solutions: *Final DSM*, *Alternate Solution A*, and *Alternate Solution B*.

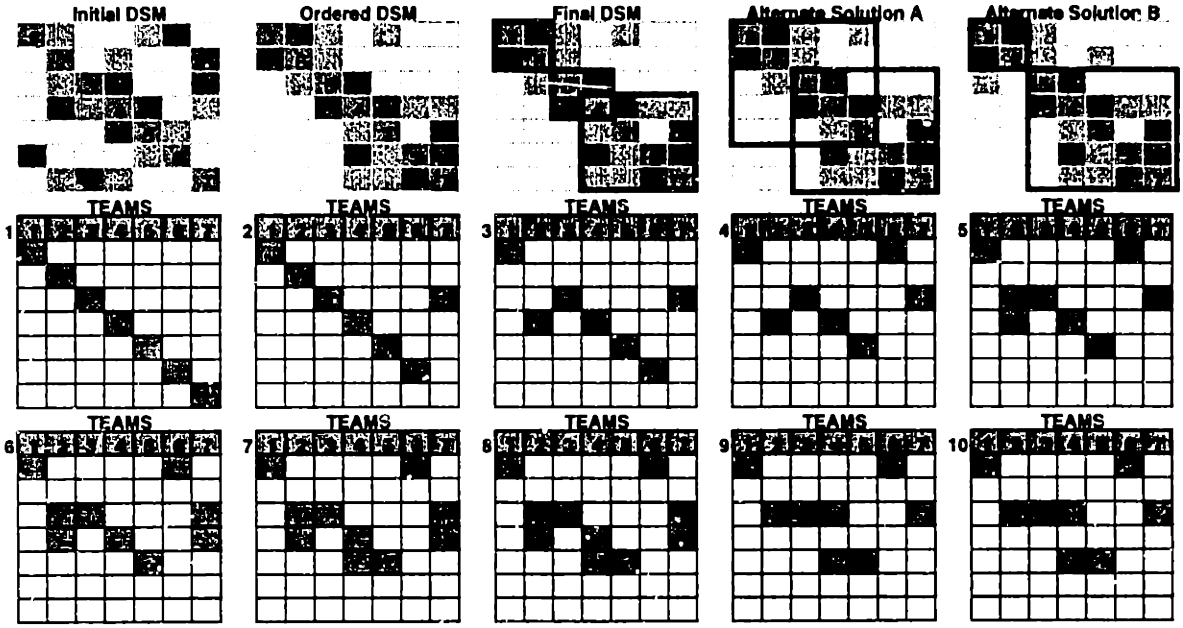


Figure 2-5. Sample problem with three possible solutions. The cluster matrices show the steps taken by the algorithm to find the first solution.[41]

The 10 cluster configurations depicted in Figure 2-4 show the steps taken by the algorithm to arrive at the clustering solution shown in the *Final DSM*. As we can see in the first matrix, the algorithm starts by assigning each team to a cluster. Then it randomly selects team 7 and calculates bids from every cluster. It turns out that the highest bid comes from the third cluster, which contains team 3. If we look at the *Ordered DSM*, this bid makes sense because team 3 has strong interactions with team 7. Therefore, the algorithm incorporates team 7 into the third cluster as shown in the second cluster matrix. Then a new team is selected; this time team 2 is randomly chosen. The two clusters containing teams 4 and 7 should have the highest bids, because team 2 has a weak and a strong interaction with both teams. However, because team 7 is already in a cluster containing two teams, and the cluster containing team 4 has one team only, the bid from

the latter cluster will be higher (the bid is inversely proportional to the size of the cluster, unless *pow_bid* is equal to 0). Therefore, team 2 is added to the fourth cluster as shown in the third cluster matrix. Then team 6 is selected. The highest bid comes from the first cluster, because team 1 has two strong interactions with team 6. The fourth matrix shows the result after both teams are grouped together in the first cluster. Team 2 is selected for the second time. Now the highest bid comes from the third cluster, and team 2 is added. Note that team 2 is now a member of two different clusters as shown in the fifth matrix. The next two matrices show, respectively, the addition of team 7 to the fourth cluster, and team 4 to the fifth one. The eighth matrix shows no changes. This can be either because the team that was randomly selected had no positive bids, or because adding it to a cluster would not have improved the coordination cost. Team 4 is selected and incorporated to the third cluster as shown in the ninth matrix. The fourth cluster has been deleted, because all of its members (teams 2, 4, and 7) are also part of the third cluster, i.e. the fourth cluster was a subset of the third one. The next cycle produces no changes (the last matrix), and neither do the remaining attempts (not shown). Therefore, the algorithm has reached a stable solution and stops. The final result is shown in *Final DSM*. There are 3 clusters, one includes teams 1 and 6, another one includes 5 and 4, and the remaining cluster incorporates teams 4, 2, 3, and 7.

The evolution of the total coordination cost for this example is shown in Figure 2-6. In this particular run, the algorithm always selected the highest bid, and produced changes only if there was an improvement in coordination costs. In other words, simulated annealing was not used. As we can see, the total coordination cost started at 686 and went down to a stable value of 242. As we saw before, there was no decrease in cost between steps 7 and 8 because the cluster matrix was left unchanged.

[41] Values of 1 and 0.5 were used for the strong and weak interactions.

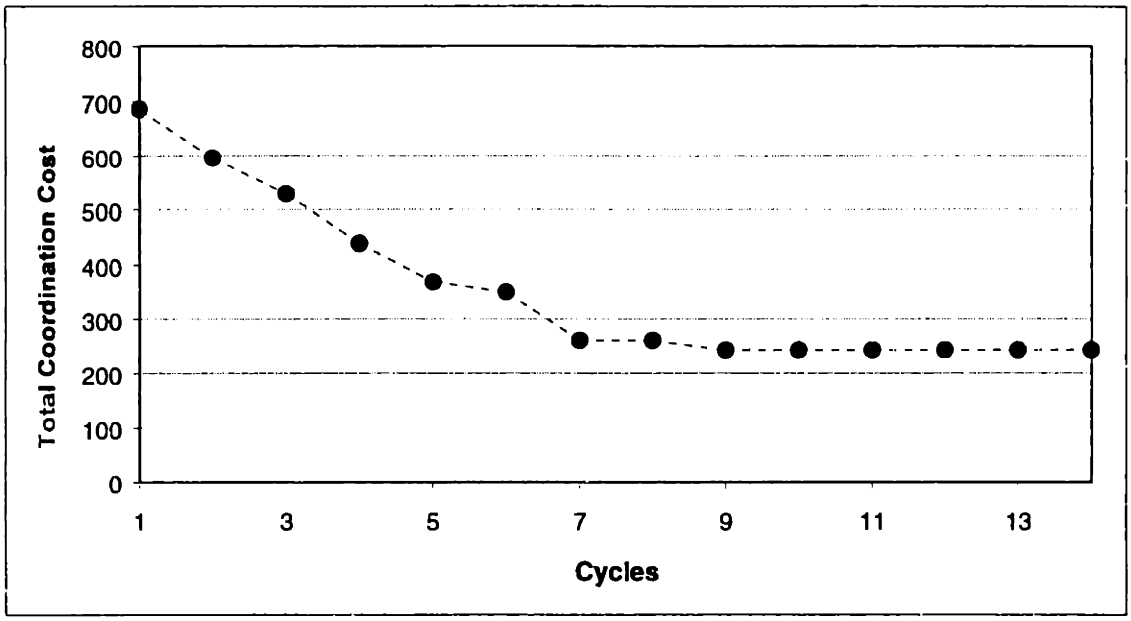


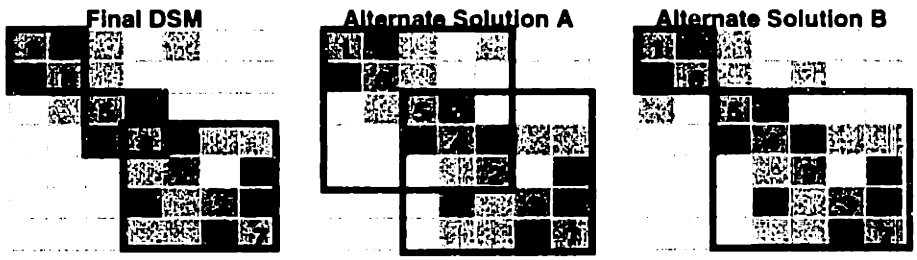
Figure 2-6. Total coordination cost history during the operation of the algorithm.

Because of the random selection of tasks, and simulated annealing, if we run the algorithm again with the same parameters, we are not guaranteed to get the same result. For example, we saw that during the third cycle, teams 4 and 7 had equal interactions with team 2, but because team 7 was already in a larger cluster, teams 2 and 7 were not grouped together. If the random selection of teams had followed a different order, the opposite situation could have occurred: team 4 could have been in a larger cluster, and teams 2 and 7 would have been put together. Clearly, the result is still somewhat dependent on the random selection of teams. Therefore, the user should run the algorithm several times to explore different solutions, and select the most convenient one.

When a problem is solved several times with the same parameters, most solutions will have similar features such as cluster size, level of overlapping, type of interactions addressed by the clusters, and range of cluster sizes. If we change the effect of simulated annealing we will broaden or shorten the range of solutions explored, but the solutions will not change dramatically. This happens because the algorithm will calculate the bid and coordination cost in a consistent manner. However, if we want to shift the likelihood of getting a particular type of solutions, then we can change the parameters that control the bid and coordination cost functions. These parameters can have more drastic effects in the type of results obtained, because they modify the bid and cost values that drive the

algorithm. This does not mean however, that changing them will definitely prevent us from obtaining results that were obtained with a different set of parameters. The effects are a function of the specific problem, and the specific parameters. The user should play with the parameters and explore how solutions behave. The general observations about parameters described in the previous section can be used as a guideline.

Figure 2-7 shows the effect of changing parameters *pow_dep* and *pow_bid*. By decreasing *pow_dep* from 2 to 0.5, strong and weak interactions are treated as if their values were more alike. Comparing *Final DSM* with *Alternate Solution A* we see that the clusters in the second case pay more attention to the weak interactions that were not addressed in the first solution. When the value of *pow_bid* is decreased from 2 (*Final DSM*) to 0 (*Alternate Solution B*), we see that the size of the clusters becomes irrelevant in the bidding process, and the range of cluster sizes increases.



Total Coordination Cost	242.0	437.5	356.0
pow_dep	2.0	0.5	2.0
pow_bid	2.0	2.0	0.0

Figure 2-7. Three distinct solutions and the parameters used to produce them.

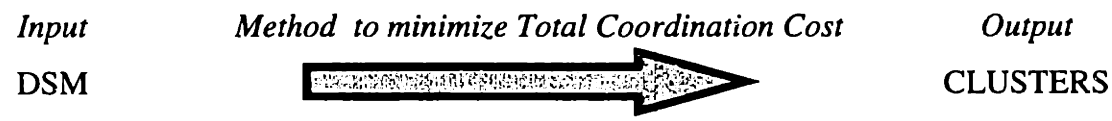
The example discussed in this section should allow the user to understand the basic inner workings of the algorithm. In the next chapters the user can find more insight into how different parameters are adjusted to encourage the formation of certain type of cluster solutions.

Chapter 3. Integer Program

Our goal is to obtain an efficient clustering algorithm that will tell us how to group elements into system teams. In this section we deal with an alternative method to get to a clustering arrangement. The alternative method is a mathematical model expressed as an integer program. It allows us to know the efficiency of the clustering algorithm that has been developed, i.e. it tells us whether a particular solution obtained by the algorithm is indeed optimal, and if not, how far it is from optimality.

Understanding the Clustering Problem

Before we discuss the integer program, let us review the problem at hand. We want to find a set of clusters (or system teams, or modules) that minimize the total coordination cost of our interaction matrix (from now on we will call it DSM, for Design Structure Matrix). The input for our model is the DSM and the desired output is the optimal clustering arrangement.



As we saw in the previous chapter, the coordination cost for a team is given by

If both teams i and j are in any cluster k, then

$$Coordination\ Cost(team_i) = \sum_{j=1}^{size} (DSM(i,j) + DSM(j,i)) * \sum_{k=1}^{Cl} cl\_size(k)^{pow-cc}, \quad \textbf{Eq. 3-1}$$

else (if no k cluster contains both i and j, the entire DSM acts as cluster containing i and j)

$$Coordination\ Cost(team_i) = \sum_{j=1}^{size} (DSM(i,j) + DSM(j,i)) * size^{pow-cc}, \quad \textbf{Eq. 3-2}$$

where the second form is used if no k cluster addressed an i-j interaction. The total coordination cost was simply the sum of the the individual coordination costs according to

$$Total\ Coordination\ Cost = \sum_{i=1}^{size} Coordination\ Cost(Team_i). \quad \textbf{Eq. 3-3}$$

To find the optimal solution, we first need to understand the possible solutions that any problem can have. It turns out that there are many solutions even for a small problem. For example, for a DSM with only two teams, we have 16 possible clustering arrangements (see Figure 3-1). The rows list the clusters, and the columns list the elements or teams included in each cluster. The top left case has both clusters empty. In the second case, the second team is a member of the second cluster. In the third case, it is team one that belongs to the second cluster. These first three examples yield the same total coordination cost. In fact, all cases labeled *A* yield the same cost, because in all of them no team interaction is addressed by any cluster. All cases labeled *B* yield a different cost, because in all of them two teams belong to one cluster. Finally, in case *C* both clusters contain both teams. Solutions *C* and *B* are the same for the purpose of addressing an interaction between teams 1 and 2, but they have a different coordination cost. Case *C* has twice the total coordination cost of case *B*.

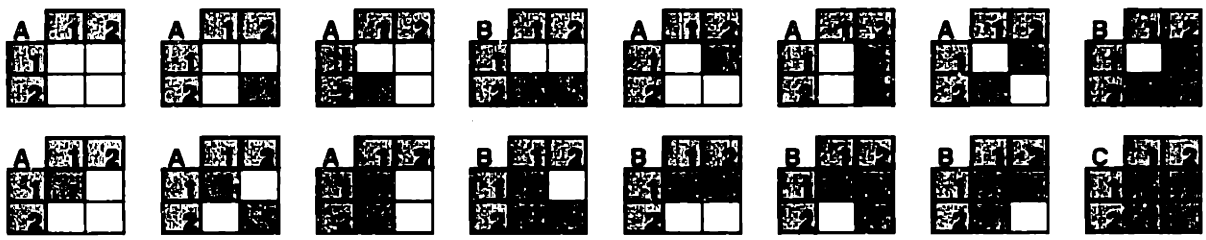


Figure 3-1. Possible cluster solutions for a DSM of size 2.

In order to validate the effectiveness of the clustering algorithm, we need to obtain the optimal clustering configuration for any particular problem. We understand that the optimal solution is the one with the lowest total coordination cost for *the number of clusters produced.* This last restriction was not apparent when we started working on this problem, but became evident as we tried to validate the algorithm by developing an alternative method. Let us explain.

As the number of items in the DSM increases, the number of possible solutions grows up exponentially. We need to limit the number of possible solutions by first deciding on a maximum number of clusters that we would allow the method to explore. This is a necessary restriction for any method, and depending on the particular DSM, it will be a binding or non-binding constraint. Restricting the number of clusters that can be

formed is a binding constraint if by relaxing the number of clusters (i.e. increasing the number of clusters that can be formed) we obtain a better overall solution. Therefore, given a number of clusters, we can either have an overall optimal solution if the constraint on the number of clusters is non-binding, or a "local" optimal solution for a limited number of clusters.

In Figure 3-2 we have a DSM with 7 teams. We are showing 5 possible solutions. Solutions A and B are restricted to two clusters. Solution *B* has a total coordination cost of 512 vs. *A*'s of 668. Solution B is optimal given the binding restriction of limiting the clusters to 2. When we allow 3 clusters in the solution, we have the optimal configuration depicted in *C* with a cost of 408. In *D*, we break up the third cluster into pairs, so rather than having a cluster with teams 1, 2 and 3 together, we form the pairs 1-2, 1-3 and 2-3 to obtain a lower cost of 348. In case *E* we further break up the second cluster from case *C* into pairs to obtain a cost of 272. Note that cases *D* & *E* are not optimal for the number of clusters allowed, yet they are better solutions than *A*, *B* or *C*.

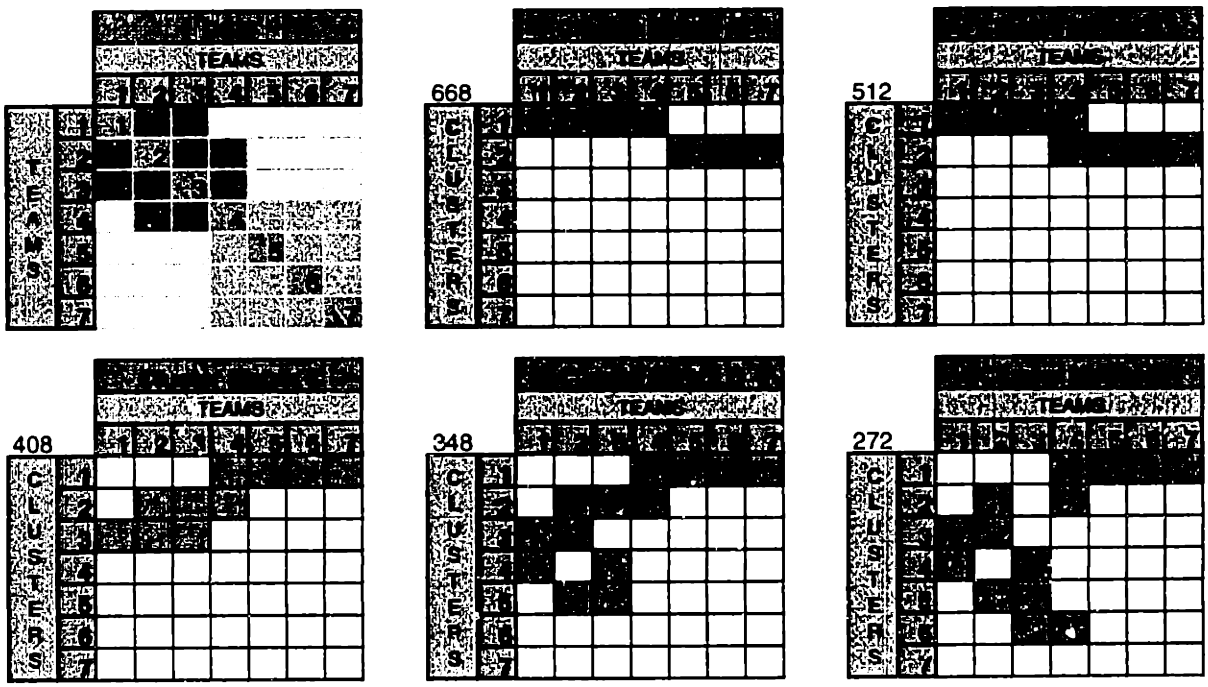


Figure 3-2. DSM with 7 elements and 5 possible solutions. The DSM has strong (dark gray) and weak interactions (light gray). Solutions list the total coordination cost.[42]

[42] The strong interactions have a value of 2, the weak ones a value of 1. Parameter *pow_cc* is equal to 2.

The overall optimal solution would be the case when we allow as many clusters as there are interacting pairs. For this example we would need to have a cluster for each interacting team: 1-2, 1-3, 2-3, 2-4, 3-4, 4-5, 4-6, 4-7, 5-6, 5-7, 6-7. A total of 11 or more clusters would guarantee that this restriction would be non-binding, and the absolute optimal configuration would be found. For practical purpose, we do not want to cluster just pairs, but we want a useful solution like the one in case B or C. So we need to be aware of the need to restrict the number of clusters to obtain a local optimal configuration.

There are several ways of getting to the optimal solution (the local optimal solution), each one with its own strengths and weaknesses. The most straightforward method is to enumerate all the possible solutions and calculate the total coordination cost for each one. The solution with the lowest total coordination cost is the optimal solution. However, listing all the solutions is a daunting task even for a small matrix. If we limit a DSM to be arranged into as many clusters as it has teams, we can have up to 2^{NxN} solutions, where N is the number of teams in the DSM. As we saw in the case of a DSM of size 2, the 16 solutions only represent 3 distinct solutions. So 2^{NxN} represents an upper limit on the number of distinct solutions.

An easy way of visualizing the number of solutions that can be obtained is to imagine that each square in the cluster matrix is a 0-1 variable (see cluster solutions A-E shown in Figure 3-2). When the variable is 1, the element belongs to a cluster and we have a gray shading, otherwise when the variable is 0, the square is blank. Every square in the cluster matrix represents a 0-1 variable and all of them together form a binary number of NxN bits. For the 7x7 cluster matrix we can have as many as 2^{7x7} solutions, i.e. $5.6x10^{14}$ (560 trillion) solutions. If a computer explored one million solutions per second, it would take it over 17 years to explore all the solutions! Clearly, even for a small matrix this is not an easy task.

To reduce the number of solutions explored, we can limit the size of a cluster, i.e. the number of teams that can be part of any cluster. Just as in the case when we limited the number of clusters, this restriction can also be binding or non-binding. The easiest way to check for this is to inspect the results, and see whether the maximum number of teams in a cluster is at least one number less than the limit value (if it is equal we can

only guess whether it is binding or not by looking at the size of the remaining clusters; if most of them are equal to the limit value, the restriction is likely to be binding).

To validate our model, we need a method that can tell us for some DSM problems what the optimal solution is. Methods like simulated annealing or genetic algorithms can get to the optimal solution, but there is no guarantee that this will happen. Therefore, we decided to write an integer program that would tell us the optimal solution for small problems. In designing this model, we took into consideration the restrictions available to us to reduce the number of solutions explored by limiting the number and the size of the clusters. We did not incorporate restrictions to limit the solutions explored to only those that are distinct, as this would have required complicated constraints that would have made the model too complicated. Instead, visual inspection can tell us what the unique solution we are looking for is from the output solution given by the method.

Visual inspection is an issue for DSMs where the number of clusters allowed exceeds the number of interacting pairs. As we can see in Figure 3-3, forcing a solution with three clusters for a problem with only one interacting pair (3-1 interaction) produces one meaningful cluster. The first cluster does not contribute to the coordination cost because only one team is present, and no interaction is addressed. The second cluster does not affect the coordination cost either, because interactions between teams 1, 2 and 4 do not exist. Only cluster three is meaningful, because it addresses the interaction between teams 1 and 3.

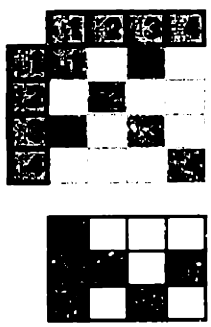


Figure 3-3. Sample DSM and cluster output where visual inspection is necessary.

The Mathematical Model

The objective of our algorithm is to minimize the total coordination cost. Likewise, in the mathematical model, we want to minimize the total coordination cost given a set of constraints. Our model is expressed as an integer program. It has four parts:

1. Minimize: Objective function
2. Subject to: Constraints
3. Variables
4. Parameters

We present the model with a brief description for reference. We also include the number of terms for each equation. Then we explain the details of the constraints.

MINIMIZE:

$$\sum_{k=1}^{Cl}\sum_{l=1}^{ClS}\sum_{i=1}^{T}\sum_{j=i+1}^{T}(DSM_{ij}+DSM_{ji})Y_{ijkl}l^{pow\_cc}+\sum_{i=1}^{T}\sum_{j=i+1}^{T'}(DSM_{ij}+DSM_{ji})Z_{ij}\cdot T^{pow\_cc} \quad \textbf{Eq. cost}$$

The first term is the interaction times the size of the cluster, the second is the interaction times the size of the matrix (when no cluster addressed the interaction)
Cl x ClS x T x T/2 + T x T/2 approximately $Team^4/2 + Team^2/2$ terms

SUBJECT TO:

$$\sum_{l=1}^{ClS} Y_{ijkl} \le 1 \qquad \forall_{i<j,k} \qquad \textbf{Eq. one\_size}$$

A cluster k that addresses an i-j interaction can have only one size l
$Team^3/2$ equations

$$2\sum_{l=1}^{ClS} Y_{ijkl} \le X_{ik} + X_{jk} \qquad \forall_{i<j,k} \qquad \textbf{Eq. presence1}$$

$$\sum_{l=1}^{ClS} Y_{ijkl} + 1 \ge X_{ik} + X_{jk} \qquad \forall_{i<j,k} \qquad \textbf{Eq. presence2}$$

If and only if i & j are in cluster k, then Y_{ijkl} records i & j's presence in cluster k (of size l)
$Team^3/2$ equations

$$Y_{ijkl} \le M \times A_{ijkl} \qquad \forall_{i<j,k,l} \qquad \textbf{Eq. neg1}$$

$$-\left(\sum_{t=1}^{T} X_{tk} - l\right) \le M\left(1 - A_{ijkl}\right) \qquad \forall_{i<j,k,l} \qquad \textbf{Eq. neg2}$$

$$Y_{ijkl} \le M \times B_{ijkl} \qquad \forall_{i<j,k,l} \qquad \textbf{Eq. pos1}$$

$$\sum_{t=1}^{T} X_{tk} - l \le M\left(1 - B_{ijkl}\right) \qquad \forall_{i<j,k,l} \qquad \textbf{Eq. pos2}$$

These four equations ensure that Y_{ijkl} takes one size, the size l should equal the number of tasks contained in cluster k
Team^4/2 equations

$$\sum_{k}^{Cl}\sum_{l=1}^{ClS} Y_{ijkl} + Z_{ij} \geq 1 \qquad \forall_{i<j} \qquad \textbf{Eq. large\_mat}$$

If an i-j interaction is not addressed by a cluster, then Z_{ij} is 1 (the i-j interaction is addressed by a cluster of size T, the size of the DSM in the objective function)
Team^2/2 equations
Total number of constraints approximately Team^4.

VARIABLES

Y_{ijkl} binary 1 if i & j tasks are present in cluster k (containing l teams)
Task^4/2 variables

X_{ik} binary 1 if task i is a member of cluster k
Task^2/2 variables

Z_{ij} binary 1 if no cluster addressed an i-j interaction.
Task^2/2 variables

A_{ijkl} binary dummy variable
Task^4/2 variables

B_{ijkl} binary dummy variable
Task^4/2 variables

Total number of variables approximately Task^4.

PARAMETERS

T	number of teams/elements in the DSM
ClS	the maximum allowable size of a Cluster (usually less than or equal to T)
Cl	the maximum allowable number of Clusters (in the algorithm it is equal to T, in the model it is usually set to less than T to limit the solutions explored)
pow_cc	the exponent used in the objective function
M	a large number (larger than the number of tasks in the matrix, e.g. 100)
DSM_{ij}	the entries in the Design Structure Matrix

Explanation of *cost*, *one_size* and *large_mat*

The *cost* objective function is the sum of all the interaction values in the DSM multiplied by the size of the cluster where an *i-j* interaction is being addressed (note that we sum over all clusters where such interaction is being addressed), or by the size of the DSM when no cluster addresses such interaction. We show both the model *cost* equation and the algorithm equations 3-1, 3-2 and 3-3 to see how they represent the same objective.

$$\sum_{k=1}^{Cl}\sum_{l=1}^{ClS}\sum_{i=1}^{T}\sum_{j=i+1}^{T}(DSM_{ij}+DSM_{ji})Y_{ijkl}l^{pow\_cc}+\sum_{i=1}^{T}\sum_{j=i+1}^{T}(DSM_{ij}+DSM_{ji})Z_{ij}\cdot T^{pow\_cc}$$

$$Coordination\ Cost(team_i)=\sum_{j=1}^{size}(DSM(i,j)+DSM(j,i))*\sum_{k=1}^{Cl}cl\_size(k)^{pow\_cc}\ \text{ or}$$

$$Coordination\ Cost(team_i)=\sum_{j=1}^{size}(DSM(i,j)+DSM(j,i))*size^{pow\_cc}$$

$$Total\ Coordination\ Cost=\sum_{i=1}^{size}Coordination\ Cost(Team_i).$$

The size of such cluster is given by the product of the binary variable Y_{ijkl} and the constant *l*, and is raised to the *pow_cc* power. By adding the *l* index we allow the size of the cluster to be a constant raised to a power which is another constant. The Y_{ijkl} variable is one only when the number of teams in cluster *k* match the size of the cluster (the size given by *l*). With this trick we avoid the need to create an integer variable that holds the size of the cluster. Such variable when raised to the *pow_cc* power would make the objective function quadratic, and the model would be much harder and time consuming to solve. When no cluster addresses an *i-j* interaction then we multiply that interaction by the size *T* of the entire DSM matrix. *T* is a constant raised to a power so it is just another constant and the objective function remains linear.

The Z_{ij} variable is one only when no cluster addressed such interaction. That is the role of constraint *large_mat*.

$$\sum_{k}^{Cl}\sum_{l=1}^{ClS}Y_{ijkl}+Z_{ij}\geq 1 \qquad \forall_{i<j} \qquad \textbf{Eq. large\_mat}$$

Constraint *one_size* ensures that a cluster *k* that addresses an *i-j* interaction can have one size at most. We have for each *i-j* interaction a *k* cluster, and for each such

cluster we have l variables. We want only one of them to be one, the one where l matches the number of teams contained in the cluster. That is the role of constraints *pos1*, *pos2*, *neg1*, *neg2* explained below.

$$\sum_{l=1}^{CIS} Y_{ijkl} \le 1 \qquad \forall_{i<j,k} \qquad \textbf{Eq. one\_size}$$

Explanation of *presence1* and *presence2*

Let A, B, C represent events that can be either true or false. If A, B, C are binary variables, they take the value of 1 when an event is true, and 0 if it is false. We want to map the relation:

C is true if and only if both A & B are true.

This can be mapped with the following two equations:

$A + B - 2C \ge 0$

$A + B - C \le 1$

Table 3-1 shows how C takes the value of 1 only when both A and B are 1.

Table 3-1. Mapping of equations.

A	B	C	A + B - 2C	A + B - C
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	1	0	1

Then by substituting $A \rightarrow X_{ik}$, $B \rightarrow X_{jk}$, and $C \rightarrow Y_{ijkl}$, we can map the constraint:

Y_{ijkl} is true if and only if both X_{ik} and X_{jk} are true.

$$A + B - 2C \ge 0 \quad \rightarrow \quad 2\sum_{l=1}^{CIS} Y_{ijkl} \le X_{ik} + X_{jk} \qquad \forall_{i<j,k} \qquad \textbf{Eq. presence1}$$

$$A + B - C \le 1 \qquad \sum_{l=1}^{CIS} Y_{ijkl} + 1 \ge X_{ik} + X_{jk} \qquad \forall_{i<j,k} \qquad \textbf{Eq. Presence2}$$

The first equation is analogous to *presence1*. The second equation is analogous to *presence2*. So these two equations make sure that when two teams i and j both belong to

a cluster (X_{ik} and X_{jk} are both one), then the *i-j* interaction has been addressed by cluster *k* that has *l* elements in it (Y_{ijkl} is 1).

Explanation of *pos1, pos2, neg1, neg2*

Let event C be either true or false, and event D can either take negative integer values, 0, or positive integer values. Then we let C be a binary variable and D an integer variable. We want to map the constraint:

If D is negative or positive, then C is false, else if D is 0, then C can be true or false.

This expression can be expressed as:

If $D > 0$ then $C = 0$

If $D < 0$ then $C = 0$

If $D = 0$ then $C = 0$ or 1

We only need to worry about the case when D is nonzero (C is free to be 1 or 0 when D is zero; C will be 1 or 0 depending on other constraints and the objective function being optimized), i.e. when D is positive or negative. These two constraints have the familiar form[43]:

If $f(x_1, x_2,...,x_k) > 0$ then $g(x_1, x_2,...,x_k) \geq 0$

which is mapped as:

$-g(x_1, x_2,...,x_k) \leq My$ and

$f(x_1, x_2,...,x_k) \leq M(1-y)$ with

$y = 0$ or 1

So to map

If $D > 0$ then $C = 0$

we use the form

If $D > 0$ then $-C \geq 0$ $(C = 0)$

where $f \rightarrow D$ and $g \rightarrow -C$ to get

$C \leq My_1$ and

$D \leq M(1-y_1)$.

Likewise, to map

If $D < 0$ then $C = 0$

we use the form

If $-D > 0$ $(D < 0)$ then $-C \geq 0$ $(C = 0)$

where f→-D and g→-C to get

$C \leq My_2$ and

$-D \leq M(1-y_2)$

So with four equations we map the desired constraint. In our problem this constraint has a more complicated form. The constraint is:

If D is negative or positive, then C is false, else if D is 0, then C can be true or false.

We substitute C→Y_{ijkl} and -D→$\sum_{t=i}^{T} X_{tk} - l$. We also use the dummy variables $y_1 \rightarrow A_{ijkl}$ and $y_2 \rightarrow B_{ijkl}$. This way we ensure that when the expression for D is positive or negative, C will be zero, and only when D is zero i.e. when the number of tasks in a cluster *k* is equal to the size of the matrix *l*, then Y_{ijkl} can be zero or 1. The substitutions yield the four equations *pos1*, *pos2*, *neg1*, *neg2*.

$C \leq My_1$	→	$Y_{ijkl} \leq M \times A_{ijkl}$	$\forall_{i<j,k,l}$	**Eq. neg1**
$D \leq M(1-y_1)$	→	$-\left(\sum_{t=1}^{T} X_{tk} - l\right) \leq M\left(1 - A_{ijkl}\right)$	$\forall_{i<j,k,l}$	**Eq. neg1**
$C \leq My_2$	→	$Y_{ijkl} \leq M \times B_{ijkl}$	$\forall_{i<j,k,l}$	**Eq. pos1**
$-D \leq M(1-y_2)$	→	$\sum_{t=1}^{T} X_{tk} - l \leq M\left(1 - B_{ijkl}\right)$	$\forall_{i<j,k,l}$	**Eq. Pos2**

Integer Program Implementation Using AMPL

In order to use the mathematical model presented in the last section, we need to use a modeling language and a solver. Advanced modeling languages allow the user to easily convert symbolic equations from a mathematical model into a set of useful equations. These equations are fed to an optimizer (solver) to analyze them and get a solution. There are several languages at our disposal. Popular commercial packages

[43] Winston, "Operations Research, Applications and Algorithms," pp. 464-501

include AMPL, Lingo, What's Best, GAMS, MPL and others. We initially referred to Table 3-2 to study the best alternatives.

Table 3-2. Modeling language overview. [44]

Name	Integrated System?	Prob Size var/const	Problem Types LP	MIP	NLP	Solvers included	Platform Win	Unix	Mac	Cost
GAMS	N	?	✓	✓	✓	15+	✓	✓		?
AMPL	Y	?	✓	✓	✓	3+	✓	✓		$2.1K
AIMMS	Y	?	✓	✓	✓	3	✓	✓		$7.5K
LINGO	Y	100k/32k	✓	✓	✓	1+	✓	✓		$5K
MPL	Y	?	✓	✓	✓	6	✓	✓	✓	?
MathPro	Y	30k/25k	✓	✓	✓	1-6	✓	✓		$6K
Whats Best	Y	32k / 16k	✓	✓	✓	1	✓			$5K
Matlab	Y	?	✓	?	✓	1	✓	✓	✓	$.5-1.0K
Excel Solver	Y	200 vars	✓	✓	✓	1	✓		✓	$.3K-1K

Because the equations in our model include indices and constraints in the indices like i<j, more simple solvers like the Excel Solver and What's Best which is an Excel add-on were discarded. All of the remaining languages could handle IP problems. Selecting one was a question of price, ease of use, the ability to separate the model from the data, solver programs included with the language, and ease of interaction with the solvers. We tried the student versions for GAMS, LINGO, MPL and AMPL and found that AMPL (Advanced Mathematical Modeling Language) was the most convenient package. AMPL interacts with CPLEX, which is an optimization package for solving linear, integer, network, and quadratic problems.[45]

We used the windows version of AMPL, which allows the user to easily set up a model and solve different problems in a matter of minutes. There are two main components needed to set up the IP model: the model file and the data file.

[44] Taken from a report by Roberto Caccia, Michael Cuppernull, Daryl Hunt, Thane Morgan, Justin Zhuang for System Optimization class at MIT, April 1997.
[45] AMPL and CPLEX can be obtained from Compass Modeling Solutions at www.modeling.com

The model file is very similar to the set of equations that we presented in the previous section. The only differences are the order in which the elements are listed, and the addition of sets, which for our model is the range over which indices operate. Therefore the model file includes;

1. Sets
2. Parameters
3. Variables
4. Minimize: Objective function
5. Subject to: Constraints

The data file includes the particular problem that we are trying to solve, as well as the specific parameters that will control the constraints and the solution. The model and the data file are independent of each other. For any DSM, the only thing that we modify is the data file.

The model file and a particular data file are shown in Figures 3-4 and 3-5 on the next pages.[46]

[46] For specifics on the AMPL modeling language refer to *AMPL, A Modeling Language for Mathematical Programming*, by Robert Fourer, David M. Gay and Brian W. Kernighan, Boyd & Fraser Publishing Company, 1993.

```
param T   > 0;        # number of tasks
param ClS > 0;        # max. size of cluster
param Cl  > 0;        # max. number of clusters
param pow_cc;         # used in the objective function
param M;
set IJ     :=  {i in 1..T, j in 1..T};
set ILTJ := {i in 1..T, j in 1..T: i < j };  # i less than j
set INEJ := {i in 1..T, j in 1..T: i <> j};  # i not equal
to j
param DSM {(i,j) in IJ}  >= 0;          # DSM(i,j)

var Y {(i,j) in ILTJ, k in 1..Cl, l in 1..ClS} binary;
          # 1 if i & j tasks are present in cluster k
          # (of size l)
var X {i in 1..T, k in 1..Cl} binary;
          # 1 if task i is a member of cluster k
var Z {(i,j) in ILTJ} binary;
          # 1 if no task addressed an i-j interaction
var A {(i,j) in ILTJ, k in 1..Cl, l in 1..ClS} binary;
          # dummy variable
var B {(i,j) in ILTJ, k in 1..Cl, l in 1..ClS} binary;
          # dummy variable
minimize cost:
     sum {(i,j) in ILTJ, k in 1..Cl, l in 1..ClS}
     ( DSM[i,j] + DSM[j,i] ) * Y[i,j,k,l] * l^pow_cc +
     sum {(i,j) in ILTJ}
     ( DSM[i,j] + DSM[j,i] ) * Z[i,j] * T^pow_cc;
subject to one_size {(i,j) in ILTJ, k in 1..Cl}:
     sum {l in 1..ClS} Y[i,j,k,l] <= 1;
          # a cluster k can only have one size l
subject to presence1 {(i,j) in ILTJ, k in 1..Cl}:
     2*(sum {l in 1..ClS} Y[i,j,k,l]) <= X[i,k] + X[j,k];
subject to presence2 {(i,j) in ILTJ, k in 1..Cl}:
     sum {l in 1..ClS} Y[i,j,k,l]+1 >= X[i,k] + X[j,k];
          # if i&j are in cluster k, then Y[ijkl] records
i&j's                presence in k
subject to neg1 {(i,j) in ILTJ, k in 1..Cl, l in 1..ClS}:
     Y[i,j,k,l] <= M*A[i,j,k,l];
subject to neg2 {(i,j) in ILTJ, k in 1..Cl, l in 1..ClS}:
     -( sum{t in 1..T} X[t,k]-l ) <= M*(1-A[i,j,k,l]);
subject to pos1 {(i,j) in ILTJ, k in 1..Cl, l in 1..ClS}:
     Y[i,j,k,l] <= M*B[i,j,k,l];
subject to pos2{(i,j) in ILTJ, k in 1..Cl, l in 1..ClS}:
     sum{t in 1..T} X[t,k]-l      <= M*(1-B[i,j,k,l]);
subject to large_mat {(i,j) in ILTJ}:
     sum {l in 1..ClS, k in 1..Cl} Y[i,j,k,l] + Z[i,j] >= 1;
          # if the ij interaction is not addressed by a
cluster,
          # then Z[i,j] is one (ij is addressed by cluster
of size T)
```

Figure 3-4. IP Model in AMPL (cluster.mod).

```
param T:=       4;    # number of tasks
param Cl:=      3;    # max. number of clusters
param ClS:=     4;    # max. size of cluster
param pow_cc:= 2.0;
param M:=       100; # a large number
param DSM:
      1     2     3     4:=
1     1     1     0     0
2     1     2     0     0
3     0     0     3     1
4     0     0     1     4;
```

Figure 3-5. Data File in AMPL (cluster.dat)

The problem we have given to AMPL is a DSM with 4 teams and two pairs of interacting teams as shown in Figure 3-5. For this particular problem, we need to specify (refer to Figure 3-4) the size of the matrix (T=4), the maximum number of clusters (Cl=3), the maximum size of the clusters (ClS=4), the power used in the total coordination cost (pow_cc=2.0), a number larger than the size of the DSM (M=100) and the DSM.

Once the model and the data file are ready, we build the model and run the solver to obtain a solution. This problem contains 234 variables and 330 constraints. For this particular problem AMPL takes approximately 9 seconds to find an optimal solution.

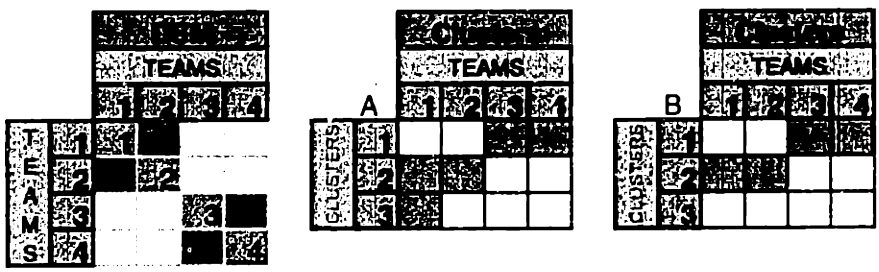


Figure 3-5. Input DSM, solution given by AMPL, implied solution after visual inspection.

We could output all the variables from the model, but we are only interested in X_{ik}, which shows what teams belong to what clusters. Such output is shown in Table 3-3, and captured in matrix A in Figure 3-5. The output from the table has three columns, the first one is the team number or i index, the second is the cluster number or k index, and the last column is the value of the variable. As we can easily see in Figure 3-5, the

optimal solution has teams 1 and 2 in cluster 2, and teams 3 and 4 in cluster 1. Note that the order of the clusters is irrelevant and different runs can yield different results, however the coordination cost and distinct solution is the same. We also note that the output shows team 1 in cluster 3. This presence is irrelevant because there is no interaction addressed by cluster 3, i.e. there is only one team present in this cluster. Therefore, by visual inspection we see that the actual solution is matrix *B*. As we mentioned before, this problem of a single team appearing in a cluster happens only when the number of clusters explored is larger than the number of interacting pairs.

Table 3-3. Output given by AMPL.

```
X :=
1 1    0
1 2    1
1 3    1
2 1    0
2 2    1
2 3    0
3 1    1
3 2    0
3 3    0
4 1    1
4 2    0
4 3    0;
```

Because the number of possible solutions grows up exponentially with the number of teams, size of clusters, and number of clusters, we used AMPL to solve DSMs with up to 8 teams. We will discuss more about these solutions in the next chapter, where we compare the performance of the algorithm with the optimal results found by AMPL.

Chapter 4. Algorithm Evaluation

In the previous two chapters, we presented two different approaches to solve the problem of finding clusters of teams in a Design Structure Matrix. In chapter 2 we talked about the stochastic approach to solve this problem. In the previous chapter the discussion centered on a deterministic integer program to find a local or global optimal solution. In this chapter, we evaluate the performance of the clustering algorithm by comparing results from the algorithm with those obtained by the integer program. We start by describing the comparison tests that were performed. Then, we show the results and discuss the strengths and limitations of the algorithm.

Comparison Tests

The objective of the comparison tests is to evaluate the performance of the clustering algorithm. A particular DSM problem is solved both by the clustering algorithm and the integer program. Because of the stochastic nature of the algorithm, we experiment with the control parameters. Once these have been adjusted to the problem at hand, we generate a set of results. The integer program is solved once. With both the algorithm runs and the optimal solution reached by the integer program, we measure the performance in two practical ways: we record how frequently the algorithm reaches the optimal solution; when it doesn't, we measure how far the total coordination cost is from the optimal solution.

The problems used in the comparisons ranged in size from four to seven teams. Larger problems were not solved because the integer program would either take too long to solve them, or could not solve them given the computer resources available.[47] Using a dream machine with the fastest processor available and plenty of RAM-memory could perhaps had allowed us to solve problems with up to nine teams. As we saw in the previous chapter, because of the exponential nature of the problem, finding the optimal solution for larger problems is impossible.

Each problem was randomly generated. It could contain empty, weak, or strong interactions. Weak and strong interactions had values of 0.15 and 0.5 respectively, and

[47] We used a Pentium machine running at 200 MHz, with 64 MB of Ram.

each one had an average probability of $^1/_6$ of appearing in any entry. Because each entry was individually generated, matrices are almost always asymmetrical, i.e. the i,j entry differs from the j,i entry. We measured such asymmetry and selected test cases with a bias towards symmetry to compensate for this effect, and reflect matrices more likely to be found in real life.

Seventy problems were solved, with matrices ranging between four and seven teams. The number of *clusters allowed* ranged between one and three clusters. There were ten problems with four, five, six, and seven teams for a total of forty randomly generated matrices. For the matrices having four teams, we restricted the problem to one, two, or three clusters; when there were five teams, we allowed three or two clusters; and for problems with six or seven teams, two clusters were allowed. These combinations are listed in Table 4-1. The combination of *clusters allowed-number of teams* is not arbitrary. It is the product of some iteration because of limitations in the ability to produce satisfactory results in either the algorithm or the integer program model.

Table 4-1. Range of problems used in the comparison tests.

Number of distinct DSMs	10			10		10	10
Size of DSM (number of teams)	4			5		6	7
Number of clusters allowed	3	2	1	3	2	2	2

The algorithm is sometimes unsuccessful at reaching very few or many clusters compared to the number of teams in the problem. This happens because the process by which teams are selected and grouped together through the bidding process prevents some solutions to be explored. For instance, it is almost impossible to form a single cluster when there are more than six teams with each team having at least one interaction. As one cluster gets larger, any team will be more likely to be grouped into a smaller cluster, because the bids from small clusters will be larger. The opposite is also true. It is very unlikely that many small clusters will be formed unless we restrict the cluster size allowed through the parameter *max_CL_size*. This happens because, for example, to form many clusters having only two teams would require that no team were selected more than once or twice, and that when chosen, the cluster containing the other team has the highest bid precisely with the selected team. Either of these issues is not important for larger

problems when the number of clusters is near a tenth to half the number of teams in the matrix. Both issues prevented us from exploring single cluster solutions for some of the test cases having five or more teams. Producing many clusters, was an issue only when trying to produce four clusters for some test cases with four teams. The integer program usage of computer resources was the limiting factor for test cases with five or more teams.

The exponential nature of the integer program was the reason behind us not exploring more clusters with the test cases having five, six or seven teams. Since the integer program has the number of clusters as one of its parameters, increasing this number by one increases the number of variables and constraints by an order of magnitude and makes the problem harder to solve with the computer resources available.[48]

Having explored the solution space in which we could compare both methods, we performed the comparison tests on the *clusters allowed-number of teams* shown in Table 4-1. Now, let us describe the specifics of the comparisons.

All tests were run with some parameters held constant, and some adjusted to facilitate the formation of solutions with the desired number of clusters. As far as the integer program is concerned, the only significant adjustment is the number of clusters allowed in the solution. The values taken by each parameter are:

- *T* it is the number of teams, so it varies according to the matrix being solved.
- *pow_cc* it was always held constant at 2.
- *Cl* the number of clusters allowed varied according to the test case.
- *ClS* the size of clusters was equal to the number of teams in the matrix, or less for large problems. However, this constraint was never binding, i.e. it played no role in the outcome of the problem.
- *M* equal to 100 in all cases.
- *DSM(i,j)* since this are the entries of the DSM , they vary according to the problem tested.

[48] Problems with four teams take around a minute to be solved. Problems with seven teams can take more than 10 hours to be solved.

Each test case was solved once by the integer program. For each solution, the cluster arrangement and the total coordination cost were recorded. We also recorded whether the solution obtained was indeed optimal. It turns out that during the solution process for a few of the most complicated cases the integer program solver uses up all the computer resources.[49] The best solution found until that point is recorded, despite not being confirmed as the optimal one.

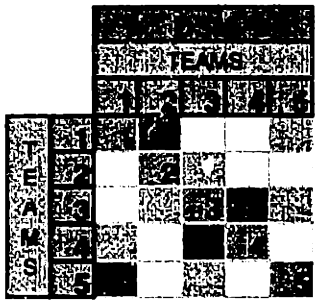
The algorithm is then run several times. A few exploratory runs are made to adjust the parameters for the problem at hand. Initial parameter values are given to the algorithm following the guidelines given in Chapter 2 under the parameter section. The range of values used in the test cases is as follows:

- *pow_cc* was fixed at 2.0 for all simulations.
- *pow_bid* For most cases it was set equal to 2.0. In one test case, we had to set it equal to 1.0 to reach the optimal solution.
- *pow_dep* Either 1.0 or 2.0 was used.
- *max_Cl_size* always set equal to the size of the DSM. We could have adjusted this parameter to form small clusters in some hard to get cases where many clusters were desired, but decided to skip this possibility and test the algorithm without its help.
- *rand_accept* it ranged in value from 1 (for small problems with few interactions) to 10.
- *rand_bid* it ranged in value from 1 (for small problems with few interactions) to 40.
- *times* for most of the test cases having four or five teams, a value of 1 was used. For all remaining cases, including those with six or seven teams, a value of 2 was used.
- *stable_limit* for most of the test cases having four or five teams, a value of 1 was used. For all remaining cases, including those with six or seven teams, a value of 2 was used.

[49] Ram memory is used up first, then the computer uses the hard disk as a swap file, slowing down the calculations, the process continues until all hard disk space is consumed.

Once the parameters have been selected, several runs are made. Each run can result in solutions with different number of clusters. Of these, only those having the number of clusters we are interested in are recorded until we reach ten different results. For instance, to solve a matrix with five teams, we can run the algorithm 50 times, with 4 runs resulting in one cluster, 13 having two clusters, 20 having three clusters, 12 resulting in four clusters, and 1 resulting in five clusters. If we were interested in solutions having two clusters, we would run the algorithm until we captured ten solutions meeting this requirement.

Let us illustrate how the comparison tests were performed with an example. Figure 4-1 depicts a randomly generated DSM with five teams. This matrix was used for two different test cases having three and two clusters each. The DSM was solved with the IP model, and the results recorded. When the number of clusters was set to three, the optimal configuration yielded a total coordination cost of 23.5. And when two clusters were allowed, the optimal cost was 33.1. The algorithm was fine-tuned, and run until ten results having three clusters were obtained. For each solution having three clusters, the results were recorded in the left-hand table. The same procedure was followed for the two-cluster case. We can see that the algorithm reached the optimal solution one of the ten runs for the three-cluster case, and seven of the eight runs for the two-cluster case. If it had not reached the optimal solution in the first case, the algorithm would have been 14% off the optimal solution (26.8 is the best non-optimal solution, and 26.8/23.5 is 1.14)



IP cost	
23.50	
Algorithm	
run	cost
1	27.35
2	26.80
3	33.10
4	27.35
5	27.35
6	**23.50**
7	27.35
8	27.35
9	27.55
10	26.80

IP cost	
33.10	
Algorithm	
run	cost
1	**33.10**
2	35.75
3	**33.10**
4	**33.10**
5	**33.10**
6	**33.10**
7	33.30
8	43.45
9	**33.10**
10	**33.10**

Figure 4-1. Sample comparison tests.

Results

In this section we present a summary of the comparison tests. As we mentioned before, a total of seventy problems were solved. Since we are interested in measuring the performance of the algorithm, and not in the test cases themselves, we only present the optimal solution of each problem, and the number of times that the algorithm found such solution. Refer to Appendix B to see the DSMs, the cluster results, and the specific costs of each of the ten runs made by the algorithm. Here, we analyze the aggregate of the results to get a broad picture of what to expect from the algorithm.

Figure 4.2 contains the results of all the comparisons. The seventy comparison tests are organized by *number of teams-clusters allowed* combinations. There are seven such combinations, and each one includes ten comparison tests. For each test, we report the following:

- *total coordination cost* this is the total coordination cost obtained by the integer program. It is the local optimal solution, unless the figure is underlined. In five of the seventy cases, the integer program could not find the optimal solution (the integer

program used up all computer resources and reported the best solution found so far as a node solution). The cost shown is then the best solution that the IP could find. If the figure is in ***italics and bold***, it means that the algorithm could never find a solution with the required number of clusters. In five such cases the optimal cost obtained by the IP is compared with the results from the algorithm in a different category (same DSM, but fewer number of clusters). In two of such cases, the number of interacting pairs was less than the number of clusters required. The three other cases, the algorithm could only produce solutions with less number of clusters than those required, despite having enough interacting pairs.[50]

- *opt. solution found=1* — If the algorithm found the optimal solution at least one out of the ten runs, then we list a 1.
- *% cost difference* — If the optimal solution is found, then this figure is 0. Otherwise, we report the percentage cost difference from the best solution found with respect to the IP solution. Note that if the IP did not find the optimal solution (instead it found a node solution), the algorithm can find a better cost; thus, this figure can be negative. In such cases, we do not know if the solution found by the algorithm is indeed optimal, but we know that it is better than the best solution (node solution) that the integer program could find.

[50] To see these problems, refer to Appendix B and identify these special cases by looking at the total coordination cost.

Test cases with 4 tasks and 3 clusters											
total coordination cost	14.4	11.6	***4.0***	22.4	***12.0***	15.4	***7.2***	4.4	5.8	22.8	
opt. solution found=1	-	1	1	1	-	1	-	1	1	-	60%
% cost difference	4.2%	-	-	-	100.0%	-	25.0%	-	-	2.6%	32.9%
# times found	-	10	10	10	-	10	-	10	10	-	100%
# times better than IP	-	-	-	-	-	-	-	-	-	-	
Test cases with 4 tasks and 2 clusters											
total coordination cost	18.6	15.2	***4.0***	27.9	24.0	22.6	9.0	8.0	7.6	27.1	
opt. solution found=1	1	1	1	1	1	1	1	1	1	1	100%
% cost difference	-	-	-	-	-	-	-	-	-	-	
# times found	6	4	10	5	10	5	10	3	6	6	65%
# times better than IP	-	-	-	-	-	-	-	-	-	-	
Test cases with 4 tasks and 1 cluster											
total coordination cost	25.8	25.2	4.0	39.9	27.0	33.2	16.2	11.6	11.2	39.1	
opt. solution found=1	1	1	1	1	1	1	1	-	1	1	80%
% cost difference	-	-	-	-	-	-	-	3.4%	-	-	3.4%
# times found	4	7	10	5	10	4	10	-	3	1	60%
# times better than IP	-	-	-	-	-	-	-	-	-	-	
Test cases with 5 tasks and 3 clusters											
total coordination cost	28.4	23.3	9.3	5.2	23.5	***10.0***	32.2	5.3	7.2	29.4	
opt. solution found=1	1	-	-	1	1	-	1	-	1	-	50%
% cost difference	-	0.4%	17.7%	-	-	105.0%	-	31.1%	-	11.9%	33.2%
# times found	1	-	-	10	1	-	2	-	10	-	48%
# times better than IP	-	-	-	-	-	-	-	-	-	-	
Test cases with 5 tasks and 2 clusters											
total coordination cost	38.9	29.7	14.1	6.7	33.1	20.5	38.5	8.5	10.4	39.2	
opt. solution found=1	1	1	1	1	1	1	1	1	1	1	100%
% cost difference	-	-	-	-	-	-	-	-	-	-	
# times found	2	1	3	2	7	10	6	2	7	6	48%
# times better than IP	-	-	-	-	-	-	-	-	-	-	
Test cases with 6 tasks and 2 clusters											
total coordination cost	130.8	40.5	51.3	57.6	27.9	114.7	26.1	101.3	32.9	87.7	
opt. solution found=1	-	1	1	1	-	-	1	-	1	1	60%
% cost difference	-49.8%	-	-	-	-4.3%	3.6%	-	4.0%	-	-	4.0%
# times found	-	1	3	2	-	-	3	-	1	3	19%
# times better than IP	10	-	-	-	2	-	-	-	-	-	
Test cases with 7 tasks and 2 clusters											
total coordination cost	75.6	131.2	67.1	57.3	88.8	89.3	55.1	138.4	63.9	85.4	
opt. solution found=1	1	1	-	-	1	-	-	1	1	1	60%
% cost difference	-	-	-11.0%	12.0%	-	11.8%	0.5%	-	-	-	11.9%
# times found	3	1	-	-	6	-	-	1	5	1	28%
# times better than IP	-	-	2	-	-	-	-	-	-	-	

NOTES:

A. Percent number of cases where the solution was found at least once.

B. Percent cost difference of the best non-optimal solution. 0% and non opt. cases excluded.

C. Percent number of times that the solution was found within succesful cases.

Bold italic costs are those taken from different cluster categories directly above or below.

Underlined costs are those where the IP solution is not optimal, but the best the IP could find.

Figure 4-1. Comparison test results.

- *# times found* Here we report how many times the algorithm could find the optimal solution. If the IP could not find the optimal solution this figure is necessarily zero, because even if the algorithm found a lower cost solution, it is not known if such is optimal.
- *# times better than IP* In the five cases, where the IP could not find the optimal solution, we report how many times the algorithm found a lower cost solution.

Going back to the example shown in the previous section (Figure 4-1), we list the results of that case in the fifth column in Figure 4-2 under the combinations of five teams, and either three or two clusters. In the case of three clusters, the optimal solution was 23.5 and the algorithm found it once. With two clusters, the optimal cost was 33.1. The algorithm reached this solution seven-out-of-ten attempts.

The last column of Figure 4-2 summarizes the results of the ten different test comparisons for each *number of teams-clusters allowed* combination. We list those numbers again in Figure 4-3.

number of teams	4			5		6	7
number of clusters	3	2	1	3	2	2	2
found at least once	60%	100%	90%	50%	100%	60%	60%
percent off from opt.	32.9%	0.0%	3.4%	33.2%	0.0%	4.0%	11.9%
times found	100%	65%	60%	48%	46%	19%	28%

Figure 4-2. Summary statistics for comparison test results.

The summary statistics measure the performance of the algorithm in the comparison tests. The three different measurements for each *number of teams-clusters allowed* combination are:

- *found at least once* This measures the percentage number of test cases where the algorithm found the solution at least once.
- *percent off from opt.* It lists the average cost difference between the best non-optimal solution and the optimal solution. The cases where the optimal solution was found and those where the IP did

not find an optimal solution are excluded to compute the average.

- *times found* This measures the percentage number of times that the algorithm found a solution out of ten attempts. Only those cases where the algorithm found the solution at least once are included to compute this figure.

Averaging the percentage difference between the optimal solution, and the best solution found by the algorithm for all seventy cases yields **3.84%**. This number is the average of all seventy *percent cost difference* results listed in Figure 4-2. Therefore, this number includes all cases, regardless of whether they are negative or positive. A histogram of results including this average is shown in Figure 4-4.

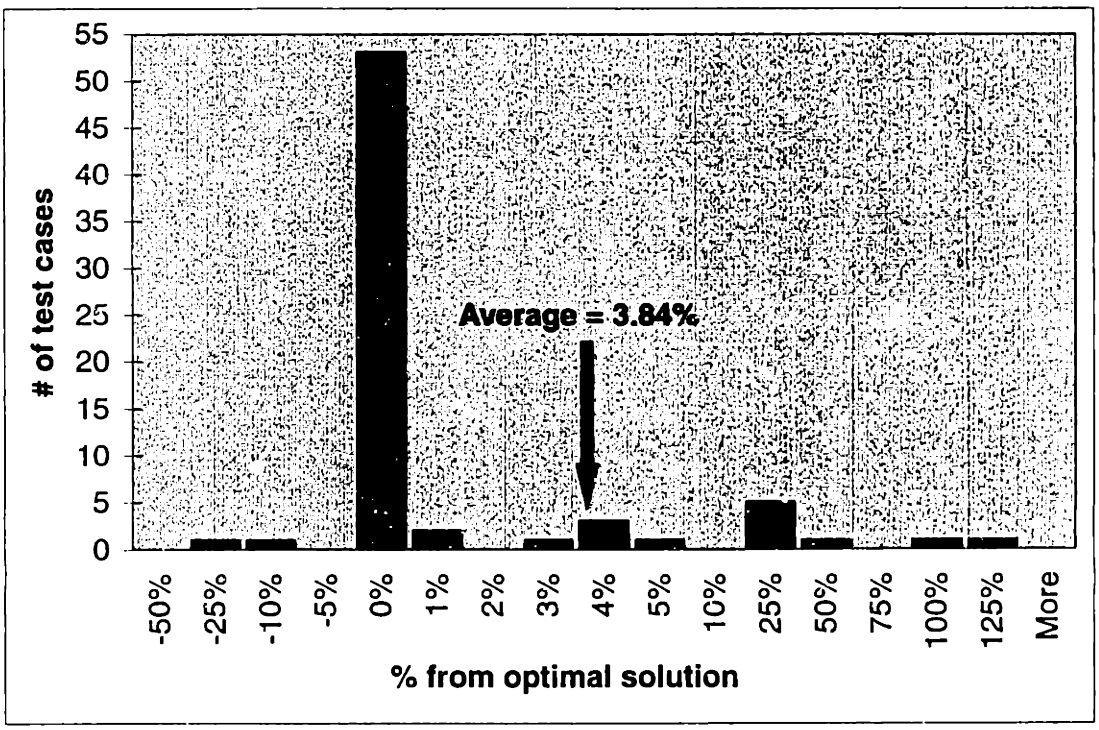


Figure 4-4. Histogram of cost performance for the best solution found by the algorithm. Note that the horizontal axis is not shown to scale.

Discussion

The results of the test comparisons are empirical measurements of the performance of the algorithm with small problems. They can help us infer what to expect from the algorithm with larger problems. Because of the limited number of test cases, and the limited number of runs within each case, the performance measurements cannot be taken as precise statistical measurements. Even for small problems, there are millions of test cases that we could have generated with different interaction values, and different levels of interactions. The measurements are averages of a few cases, and any particular result can significantly affect its value. Our intent by showing this metrics is to give a rough estimate of how the algorithm behaves and give an educated guess on the performance of the algorithm for larger problems.

For the seventy different combinations tested, the algorithm found the optimal solution between 50% and 100% of the ten randomly generated problems. When the optimal solution was found at least once, the average number of times that the algorithm reached optimality ranged between 19% and 100%. As the size of the problem increases from four to seven, the latter metric decreases from 100%, 65 % and 60% to 19% and 28%.

As we have discussed before, as the size of the problem increases, the number of possible solutions increases and the likelihood that the algorithm will find it decreases. For a larger problem with dozens of teams, the likelihood that the algorithm will find the optimal solution should be very small. By looking at these metrics we learn two things: the algorithm has to be run several times to get a good solution, and as the problem size increases to a real-life problem, the likelihood that the optimal solution will be found decreases practically to zero.

The percentage cost difference metric is very informative, because it gives us an insight into how far the cost obtained by the algorithm will be from an optimal solution for larger problems. For the first combination, there were four cases when the algorithm could not find the optimal solution. The average cost difference for the best non-optimal solution is 32.9%. Because the problems are so small (four teams), not addressing a single interaction has a large impact on the coordination cost as in the fifth column, where the percentage difference is 100%. This effect decreases with problem size. For

problems with six and seven teams, the largest single difference is 12%, and the average cost difference for cases where the algorithm did not reach optimality is 4% and 11.9 %, respectively.

The histogram of percentage cost difference shown in Figure 4-4 depicts all seventy cases in a single chart. The algorithm was able to find the optimal solution for more than 50 test cases. Only in a couple of cases the algorithm found solutions that were more than 50% larger than the optimal solution. In three of the five occasions where the integer program did not find the optimal solution, the algorithm was able to find better solutions. Averaging all cases gave a percentage from optimality equal to 3.84%. The algorithm is thus effective at finding near optimal solutions in most cases.

As the problem size increases to become a real-life problem, the percentage cost difference between the best results of the algorithm (those obtained after adjusting the parameters and selected from a large number of runs) and the optimal solution can be expected to be close to the optimal solution. This will be the case if solutions have near a tenth and half as many clusters as there are teams (the algorithm is not effective when reaching solutions with very few or many clusters). Not addressing a few interactions on larger problems has less impact on the cost than not addressing a single interaction in a small problem. In addition, with larger problems, the built in randomness of the algorithm makes it unlikely that specific team order selection during the execution of the algorithm will result in a far from optimal solution. The filtered results of the tests, and the factors mentioned ensure that the lower cost solutions will be reasonably close to an optimal solution that we cannot find by any means, but that nonetheless exists.

We could have presented different metrics for the performance of the algorithm, but we believe that the ones selected are the most appropriate to give an educated guess about the behavior of the algorithm for larger problems. In a larger problem, the user should follow a similar approach to the one followed here to perform these comparison tests. Namely, the user should play with the parameters of the algorithm to favor solutions of a desired type. Then the algorithm should be run several times. The number of clusters will vary with each run. Those within a few clusters of the desired number of clusters should be recorded, and of those, only the ones with lowest costs should be selected. These selected solutions will be comparable to those that we have measured

here. They will almost never be optimal, but they will be close to the optimal solution. Then, the user can study the best solutions and adjust them to reflect other issues not addressed explicitly in the DSM to obtain a practical solution that can be implemented. In the next chapter, we present real-life problems from industry, and solve them using the algorithm.

Chapter 5. Industrial Application

In this chapter we use the algorithm to find several solutions to a set of DSM problems provided by Visteon Systems Engineering Group (part of Ford Motor Company). These problems analyze the interactions between different physical components of an automobile cockpit module according to six categories. We start by describing the contents of the different DSM problems and the type and level of interactions. Then, we present different solutions obtained by the clustering algorithm for some of the DSM matrices. Finally, we discuss how these solutions should be adjusted by Ford[51] to reflect information not contained in the matrices. Once final solutions are produced, Ford can implement them to improve the development of car cockpit module.

Interacting Matrices

The data presented here relate to interactions between the components of a car cockpit module. This data was provided by a team at Ford analyzing the interactions between cockpit module components in a particular vehicle program. This team studied the architecture of the front cockpit module, and identified 23 different components. They studied each component trying to identify how it interacts with other components. Then they created six matrices that describe six different types of interactions between these components[52]. For each type of interaction they assigned five different levels of interactions between each pair of components. Finally, they filled-in the six matrices giving a numerical value to each level of interaction, according to Pimmler-Eppinger method.[53]

The 23 cockpit module components are listed in Table 5-1. Each component has been assigned a number that will be used to present the interaction matrices and clustering results later in this chapter. Some components refer to individual items such as cup holder, glove box, and air bag, while others refer to a group of items or modules such as ashtray/powerpoint, and steering column/wheel/switches/ignition switches. The interactions listed in the matrices refer to the whole component listed.

---

[51] For simplicity, we refer to Ford Motor Company as Ford.
[52] Similar interactions are listed in Table 1-1 in Chapter 1, and Pimmler and Eppinger (1994).
[53] See Table 1-2 in Chapter 1, and Pimmler and Eppinger (1994).

Table 5-1. Components of an automobile cockpit module.

COMPONENT NAME	#
Cup Holder	1
Ashtray / Powerpoint	2
Glove Box	3
Knee Bolster	4
Center Structure / Stack	5
Cross Car Beam	6
Climate Ducting - Air Handling Ventilation	7
Mini / Center Console	8
Comp/Clock/D.I./Message/Voice/Guidance/Nav/EMC	9
Cellular Phone	10
Instrument Cluster	11
Radio (Traditional Brick & Controls)	12
Climate Controls	13
Fuse Box / JB	14
GEM Module	15
Misc. Switches	16
Dash Insert	17
Climate HVAC - Heating / Defrosting	18
Steering Column / Wheel / Switches / Ignition Switches	19
Air Bag	20
Brake Booster	21
Pedal Package / Clutch Controls	22
Substrate / Pad	23

These 23 components interact in several ways. The team at Ford listed the interactions between components into six different categories, five of them relate to physical adjacency, and one relates to energy transfer between components. The categories are:

1. ENERGY: Energy transfer.
2. STRUCTURE ROBUSTNESS: Physical adjacency for structural design loads.
3. ERGONOMICS: Physical adjacency for human factors.
4. SERVICABILITY: Physical adjacency for service.
5. NVH: Physical adjacency for Noise, Vibration, and Harshness.
6. ASSEMBLY: Physical adjacency for assembly.

For each type of interaction, they classified pairs of interactions between components into one of the following five levels:

- -2 interaction strongly discouraged
- -1 interaction not recommended
- 0 no interaction
- 1 interaction recommended
- 2 interaction strongly recommended

The team at Ford composed six matrices listing the six different types of interactions according to the values listed above. Then they attempted to cluster the components by using a macro developed in Excel. Soon they discovered that the number of solutions is prohibitively large, and realized that their macro would not solve the problem. The negative interactions were also a problem, since it is difficult to handle both "attractions" and "rejections" simultaneously. Interested in our algorithm, they gave us the data and asked us for some results, including a weighted-average combination of all six matrices.

Since the algorithm cannot handle negative interactions, we substituted all negative entries with zeroes. We also produced a combination matrix that is a linear combination of all six matrices. For each entry in the combination matrix, we added all corresponding entries from the six matrices (before substituting negative entries for zeroes) and divided the result by 6. Then we substituted the negative values by zeroes. In this way the combination matrix contains fractional values between 0 and 1. For the purpose of graphical display, low interactions include values between 0 and 0.5 inclusive and are shown in light-gray, strong interactions are those above 0.5 and are shown in dark-gray. For all other matrices, weak interactions have a value of 1, and strong ones have a value of 2. For this thesis, we are interested in the process of obtaining solutions, rather than in the solutions themselves. Therefore we show only one of the individual matrices and the combination matrix, and the results for each.

Results

As we mentioned in the previous chapter, the algorithm should be fine tuned to favor a particular type of solution. Then it should be run several times to select the best solutions. Figures 5-1, 5-2 and 5-3 show the distribution of solutions for the combination matrix (the combination matrix is shown in Figure 5-4 later in this section) according to three different sets of parameters. Each figure shows the total coordination cost and the number of resulting clusters for 100 different runs.

The results shown in the three figures have the same values for the following parameters: *pow_cc* = 2.0, *pow_dep* = 2.0, *rand_accept* = 23, *rand_bid* = 10, *stable* = 2, and *times* = 2. Parameters *pow_bid* and *max_Cl_size* affect the outcome of solutions in a very clear and distinct form, and vary across figures.

A high value of *pow_bid* such as 3.0 in Figure 5-1 causes the majority of solutions to have a large number of clusters centered on 12. When this parameter decreases to 2.0 as in Figure 5-2, the number of clusters centers on 10 and 11. A low value like 1.0 in Figure 5-3 favors solutions with fewer clusters. In this particular case, such value produces 6 clusters on average.

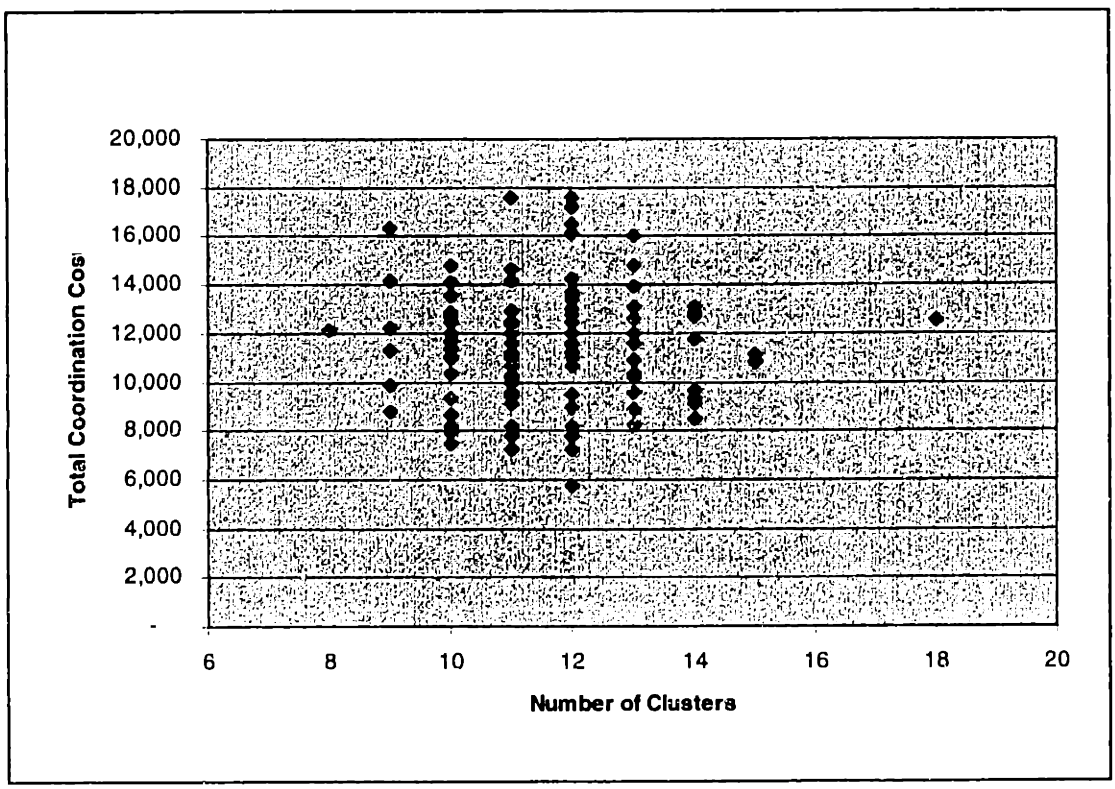


Figure 5-1. Distribution of solutions for the combination matrix with *pow_cc*=2.0, *pow_bid*=3.0, *pow_dep*=2.0, *max_Cl_size*=23, *rand_accept*=23, *rand_bid*=10, *stable*=2, *times*=2.

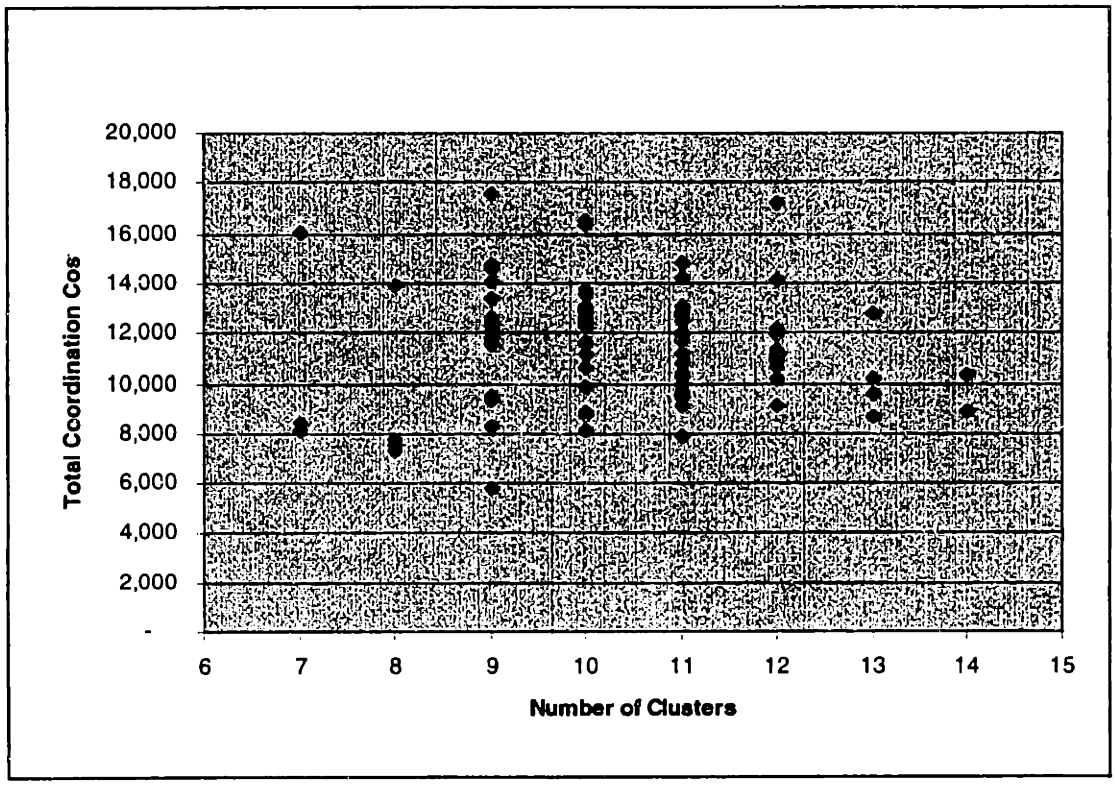


Figure 5-2. Distribution of solutions for the combination matrix with *pow_cc*=2.0, *pow_bid*=2.0, *pow_dep*=2.0, *max_Cl_size*=7, *rand_accept*=23, *rand_bid*=10, *stable*=2, *times*=2.

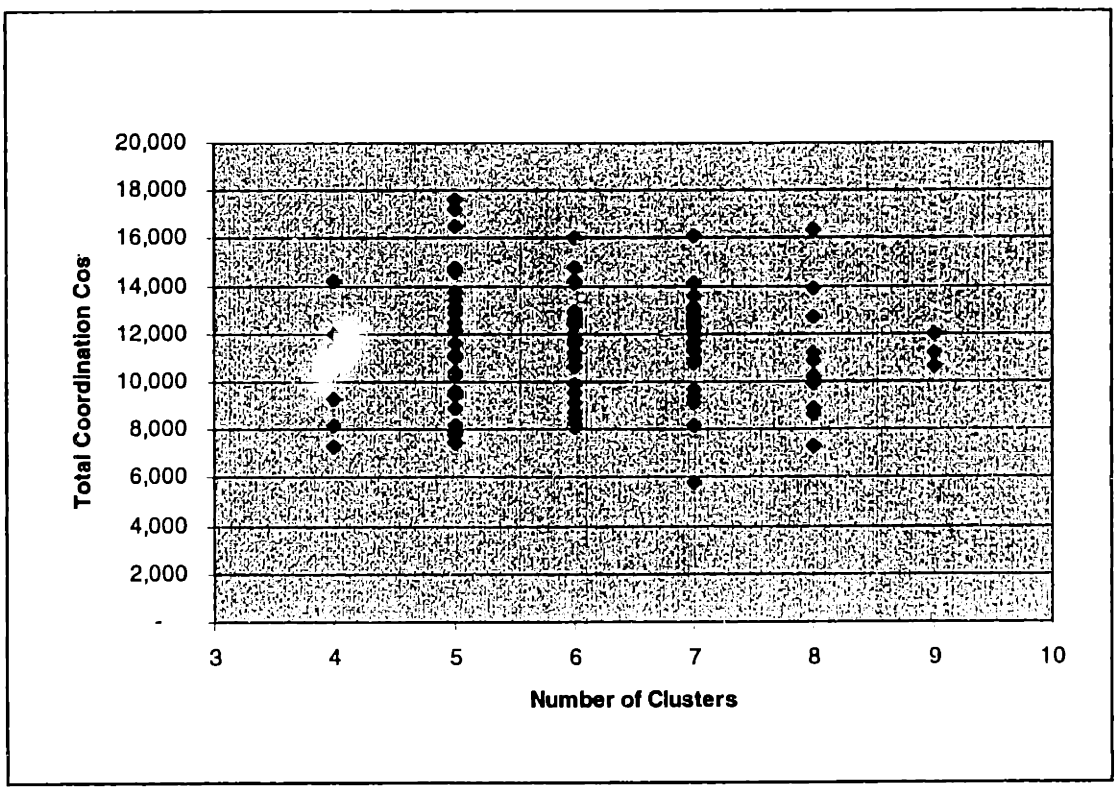


Figure 5-3. Distribution of solutions for the combination matrix with *pow_cc*=2.0, *pow_bid*=1.0, *pow_dep*=2.0, *max_Cl_size*=10, *rand_accept*=23, *rand_bid*=10, *stable*=2, *times*=2.

The effect of limiting the size of clusters can have some effect on the number of solutions. If most clusters are filled to capacity, the solution will tend to have more clusters. The most important effect though, is not on the number of clusters, but on the uniformity of clusters. If the size of clusters is not limited, there can be a whole range of cluster sizes. Limiting the size makes it easier to find lower cost solutions.

Figure 5-4 on the next page shows the combination matrix and ten different results obtained by the algorithm. There results cover a range of cluster sizes that could be useful to Ford, and were selected among the best solutions obtained during several runs. The particular costs, the parameters used for each solution, and the number of clusters for each solution are listed in Table 5-2. Looking carefully at the solutions, and comparing them to Figures 5-1, 5-2, and 5-3, one can see that the first three solutions are similar to some of the lower cost solutions of Figure 5-1. The next two solutions are similar to the lowest found in Figure 5-2. Of the last five solutions, only solution number 8 has the same parameters as those solutions listed in Figure 5-3. We can see that we could have selected a lower cost solution having 4 clusters (lowest cost is around 7,000 in Figure 5-3, vs.11,230 in Table 5-2) . The ten selected solutions listed were not generated during the runs used to obtained Figures 5-1, 5-2, and 5-3, but are clearly similar. The solutions listed in Table 5-2 were the three best solutions found when we produced those results. If we had performed more runs, lower solutions such as those shown in Figure 5-3 could have been selected. Clearly, as more solutions are produced, the likelihood of finding better solutions increases.

Table 5-2. Summary of selected results for the combination matrix, Fixed parameters for all solutions were *pow_cc*=2.0, *pow_dep*=2.0, *rand_accept*=23, *rand_bid*=10, *stable*=2, *times*=2.

Solution Number	Number of Clusters	Total Coordination Cost	Value of *pow_bid*	Value of *max_Cl_size*
1	10	8,510	3	23
2	11	7,922	3	23
3	11	7,241	3	23
4	9	6,925	2	7
5	7	8,155	2	7
6	6	9,495	1	9
7	4	11,500	1	9
8	4	11,230	1	10
9	4	13,526	1	11
10	5	10,522	1	8

Figure 5-5 shows the STRUCTURE matrix and five different results obtained by the algorithm. The particular costs, parameters, and the number of clusters for each solution are listed in Table 5-3.

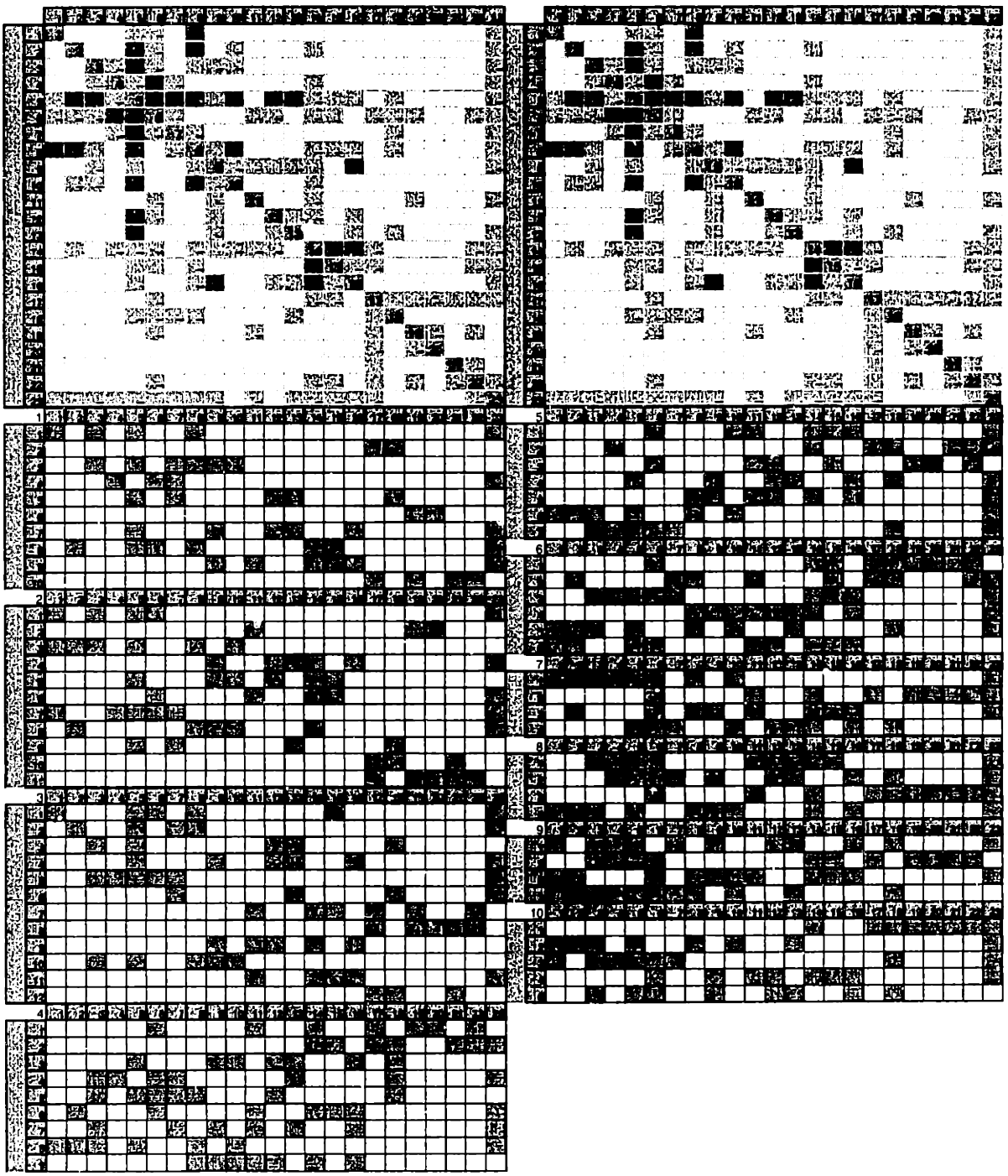


Figure 5-4. Selected cluster solutions for the Combination matrix.

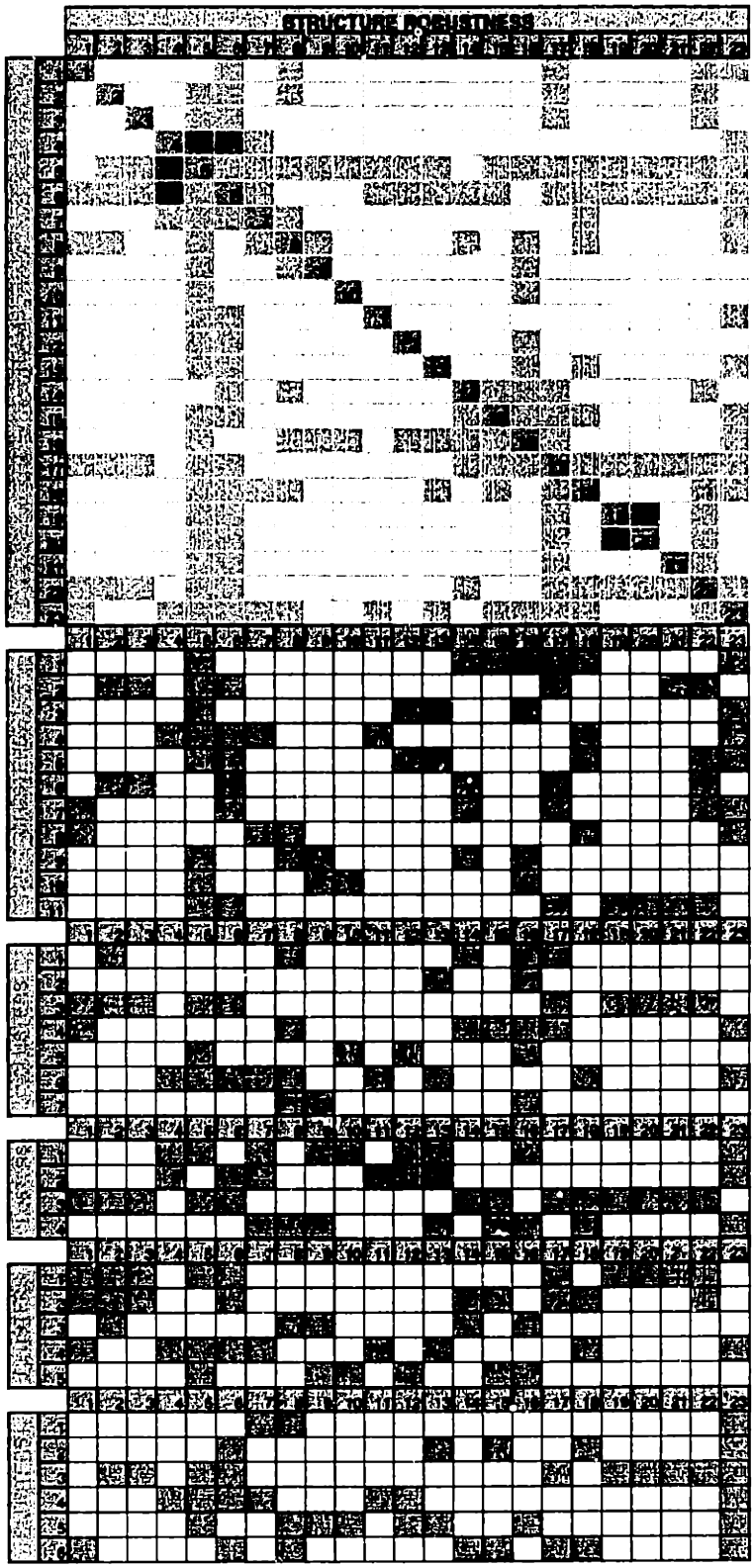


Figure 5-5. Selected cluster solutions for the Structure matrix.

Table 5-3. Summary of selected results for the structure matrix. Fixed parameters for all solutions were *pow_cc*=2.0, *pow_dep*=2.0, *rand_accept*=50, *rand_bid*=50, *stable*=2, *times*=2.

Solution Number	Number of Clusters	Total Coordination Cost	Value of *pow_bid*	Value of *max_Cl_size*
1	11	14,096	2.5	23
2	7	23,196	1.0	23
3	4	31,254	1.0	23
4	5	27,458	1.0	23
5	6	23,758	1.0	23

Discussion

We presented a selection of the results given to Ford. The results shown in Figures 5-4 and 5-5 were selected from hundreds of possible solutions. They were chosen because, for the number of clusters obtained, they were among the solutions with lowest total coordination cost, and are closest to the local optimal solution (as we discussed in Chapter 3, any problem has one local optimal solution for a given number of clusters). Since we cannot exactly control the number of clusters produced by the algorithm, we play with the parameters, in particular *pow_bid*, to encourage the formation of solutions centered on an average number of clusters. We adjust the parameters to include results that range across the number of clusters that the algorithm can easily produce.

Depending on the particular problem being solved, solutions having a particular number of clusters will be of more interest. Such solutions should be analyzed and compared not putting so much emphasis on the total coordination cost, but on other features of the solution. Solutions can be evaluated on the specific teams clustered together, or the level of overlapping, or the negative interactions not addressed by the algorithm, or on the range of sizes of clusters. Furthermore, solutions should be evaluated on information not reflected in the matrix, but known to the people who captured the matrix.

For example, in the case of the Combination matrix, we show more than one solution having the same number of clusters. There are three solutions having four clusters: solutions numbered 7, 8 and 9. Based on the coordination cost alone, solution number 8 is better. But perhaps solution number 9 is easier to implement, or it does not

group together teams that had negative interactions (and were substituted for zeroes to be solved by the algorithm). Maybe solution number 7 is more attractive because clusters are limited to 9 teams rather than 10 (in solution number 8), or 11 (in solution number 9), and the teams would be better off by being members of fewer system teams.

The analysis is not over there, once a particular solution has been selected, the user can explore variations around this solution, by manually modifying the composition of clusters, and observing its effect on the total coordination cost. A separate program to calculate the coordination cost of a solution has been developed (see Appendix A for instructions on how to use it). With this tool, the user is not restricted to blindly manipulate the composition of clusters. The cost function program provides an objective measurement that instantaneously tells the user whether modifying a cluster improves or worsens the total coordination cost.

The final solution that is selected to be implemented will be the product a multistage analysis:

- The user should adjust the parameters of the algorithm to favor solutions of a desired type. Then, the algorithm should be run several times.
- Solutions close to the desired number of clusters should be recorded, and of these, only the ones with lowest costs should be selected.
- Then, the user can study the best solutions, and adjust them to reflect other issues not captured in the clusters to obtain a practical solution that can be implemented. A tool to evaluate the coordination cost through manual manipulation of a solution during this last step helps in the generation of the final solution.

The results obtained by the algorithm and given to Ford (of these, we presented here only those for the combination and structure matrix) have undergone part of this process. People at Ford need to select solutions centered on a particular number of clusters, and inspect them to select one. Then, they can manually modify this last solution with the possible help of the coordination cost program to produce a final result that can be conveniently implemented in accordance to the goals at Ford.

Chapter 6. Conclusions

Summary

Developing successful products is greatly facilitated through the use of integration analysis. Teams working on a product development project need to be brought together into clusters or system teams to address inter-team interactions. Finding clusters of teams that efficiently address interactions has been attempted through different clustering techniques or manual manipulation. In this thesis we have presented a stochastic method to find such clusters in a rapid, practical and efficient manner. We have developed a mathematical model that finds the optimal configuration of teams for a given number of clusters. To evaluate the performance of the algorithm, we generated a series of test problems, and compared solutions given by the algorithm with those obtained by the integer program version of the mathematical model. We also showed an industrial application where the algorithm was used to generate a set of solutions that can potentially be implemented. Analysis of the architecture of an automobile cockpit module served as the basis of our example.

The clustering algorithm that has been developed is based on a previous algorithm. The new algorithm can quickly generate a range of solutions to real-life people-based problems. It can also be used for parameter-based problems if negative interactions are ignored. The algorithm can be adjusted to favor finding solutions with certain characteristics such as level of overlap, number of clusters, maximum number of teams per cluster, and emphasis on the level of interactions addressed by the clusters. The algorithm can be controlled and used directly from a spreadsheet, making it very easy to generate solutions in a matter of seconds.

The mathematical model showed us that a DSM clustering problem has different local optimal solutions, depending on the number of clusters allowed. Global optimality usually implies having more clusters than teams: clearly an impractical solution. Finding local optimal solutions for real-life problems is practically impossible, because of the exponential nature of the problem, and limitations on computer resources. Therefore, the

integer program version of the mathematical model can only be used to solve small problems.

The performance of the algorithm was experimentally measured with seventy randomly generated test cases (ten different problems for each of seven combinations of problem size-number of clusters allowed). For each type of problem ten different algorithm solutions were recorded. The algorithm found the optimal solution at least one out of ten attempts for at least 50% of the test case type of problems. When the optimal solution was found at least once, the average number of times that the algorithm reached optimality ranged between 19% and 100%, depending on the type of problems. When optimal solutions could not be found even one out of ten attempts, the average cost difference for the best non-optimal solution ranged between 32.9% for the smallest problems, to 11.9 %, for the largest.

The performance of the algorithm on small problems helped us predict that for larger problems, the lower cost solutions selected from a large number of runs will be reasonably close to an optimal solution. This will be the case if solutions have approximately between a tenth and half as many clusters as there are teams.

When solving a larger problem, such as in the automobile cockpit module example used in the industrial application, the algorithm should be used following some specific steps:

- The user should adjust the parameters of the algorithm to favor solutions of a desired type. Then, the algorithm should be run several times.
- Solutions close to the desired number of clusters should be recorded, and of these, only the ones with lowest costs should be selected.
- Then, the user can study the best solutions, and adjust them to reflect other issues not captured in the clusters to obtain a practical solution that can be implemented. A tool to evaluate the coordination cost through manual manipulation of a solution during this last step helps in the generation of the final solution. This tool has been included in this thesis.

Future research

The field of integration analysis applied to product design is an evolving area with plenty of opportunities for research. With globalization and increased competition, consumers will expect cheaper and better products. To keep up with these changes and be successful, companies will need to bring products faster to market, using more efficient use of resources while keeping development costs to a minimum. As a consequence, integration analysis tools to improve the product development process will be more widely used, and results from such analysis more frequently and thoroughly implemented. These tools themselves will have to improve, be easier to learn, and use.

The clustering algorithm that has been developed is a tool that quickly generates cluster solutions. It is much better and faster at finding cluster solutions than previous clustering algorithms or manual manipulation. Furthermore, it finds solutions that while rarely optimal, are near the optimal solution. However, the algorithm has ample room for improvement. The following are some areas to explore:

- Incorporate the ability to handle both negative and positive interactions.
- Reorder tasks to automatically visualize cluster results in the DSM.
- Explore the performance of the algorithm with a single level of interaction.
- Study statistically the effect of modifying individual parameters on overlap, number of clusters, range of cluster sizes, and levels of interactions addressed.
- Use genetic algorithms in combination with the clustering algorithms to see if solutions improve.

The means by which implementation of a final solution occurs through integrative mechanisms such as co-location has been studied. However, research into how a final solution is selected, and what factors are commonly considered to arrive at such solution, would facilitate the process of selecting comparable solutions obtained by the clustering algorithm.

Research performed at MIT and elsewhere can provide answers in the near future to some of the issues raised here.

Bibliography

Altus, Stephen S, Ilan M. Kroo, and Peter J. Gage, "A Genetic Algorithm for Scheduling and Decomposition of Multidisciplinary Design Problems", *21st ASME Design Automation Conference*. Boston. 1995.

Bezdek, James C., Sankar K. Pal, and Sankar E. PalList, "Fuzzy Models for Pattern Recognition: Methods That Search for Structures in Data", *IEEE*. August 1992.

Biles, William E., Adel S. Elmaghraby, and Ismail Zahran, "A Simulation Study of Hierarchical Clustering Techniques for the Design of Cellular Manufacturing Systems", *Computers Industrial Engineering*. Volume 21, Nos. 1-4, 1991. 267-271

Browning, Tyson R., "An Introduction to the Use of Design Structure Matrices for Systems Engineering, Project Management, and Organization Planning", *Lean Aircraft Initiative Working Paper*. February 18, 1997.

Browning, Tyson R., "Systematic Integrated Product Team Integration in Lean Development Programs", *Master's Thesis. MIT*. June 1996.

Clark, Kim B., and Steven C. Wheelwright, "Organizing and Leading 'Heavyweight' Development Teams", *California Management Review*. Volume 34, No. 3. Spring 1992. 9-28

Eppinger, Steven D., Daniel E. Whitney, Robert P. Smith, and David A. Gebala, "A Model-based Method for Organizing Tasks in Product Development", *Research in Engineering Design*. Vol. 6, 1994. 1-13

Fourer, Robert, David M. Gay, and Brian W. Kernighan, "AMPL, a Modeling Language for Mathematical Programming", Boyd & Fraser Publishing Company, Danvers, MA. 1993.

Hartigan, John A., "Clustering Algorithms", John Wiley & Sons, New York, 1975.

Hoedemaker, Geert M., Joseph D. Blackburn, and Luk N. Van Wassenhove, "Limits to Concurrency", *INSEAD Working Paper*, INSEAD, Fountainebleau, France. January, 1995.

Idicula, John, "Planning for Concurrent Engineering", *Thesis draft*, Nanyang Technological University. March 1995.

Kahn, Kenneth B., "Interdepartmental Integration: A Definition with Implications for Product Development Performance", *The Journal of Product Innovation Management*. Vol. 13, No. 2. March 1996.

Kelley, Al, and Ira Pohl, "A Book on C: Programming in C", 2nd Ed. The Benjamin/Cummins Publishing Company, Inc. 1990.

Kusiak, A, and J. Wang, "Decomposition of the Design Process", *Journal of Mechanical Design*. Volume 115. December 1993. 687-695.

Lerman, Steven R. "Problem Solving and Computation for Scientists and Engineers: An Introduction Using C", Prentice Hall. 1993.

McCord, Kent R. and Steven D. Eppinger, "Managing the Integration Problem in Concurrent Engineering", *MIT Sloan School of Management Working Paper, No. 3594*. August 1993.

Morelli, Mark D., "Evaluating Information Transfer in Product Development", *Master's Thesis. MIT Sloan School of Management.* 1993.

Pimmler, Thomas U. and Steven D. Eppinger, "Integration Analysis of Product Decomposition", *ASME Design Theory and Methodology Conference*, Minneapolis, Minn. September 1994.

Rechtin, Eberhardt, "Systems Architecting: Creating & Building Complex Systems", Englewood Cliffs, New Jersey: P T R Prentice Hall. 1991.

Rogers, James L., and McCulley M. Collin, "Integrating a Genetic Algorithm into a Knowledge-Based System for Ordering Complex Design Processes", *NASA Technical Memorandum TM-110247*. April 1996.

Smith, Robert P., and Steven D. Eppinger, "A Predictive Model of Sequential Iteration in Engineering Design", *MIT Sloan School of Management Working Paper, No. 3160*. March 1996.

Smith, Robert P., and Steven D. Eppinger, "Identifying Controlling Features of Engineering Design Iteration", *MIT Sloan School of Management Working Paper, No. 3348*. September 1992.

Steward, Donald V., "The Design Structure System: A Method for managing the Design of Complex Systems", *IEEE Transactions on Engineering Management.* August 1981.

Szakonyi, Robert, "101 Tips for Managing R&D more Effectively-I", *Research Technology Management.* Volume 33 No. 4. July-August 1990.

Walsh, William J., "Get the Whole Organization Behind New Product Development", *Research Technology Management.* Volume 33, No. 6. Nov.-Dec. 1990. 32-36

Winston, Wayne L., "Operations Research, Applications and Algorithms", Duxbury Press, 3rd. Ed. 1994. 464-501

Appendix A. Using Excel to Solve DSMs, and C Code for the Algorithm and Cost Program

Advanced programming languages, and the latest features built into spreadsheet programs, allow the user to easily setup, solve, and visualize a DSM clustering problem. In this appendix, we explain how to run the clustering algorithm and coordination cost programs directly from Excel.[54] We show how to use the graphical features of this spreadsheet to visualize data more easily. We also present the C code for the Clustering Algorithm and the Coordination Cost Program.

Using the Programs in Excel

For the purpose of this thesis, we have used two useful features of Excel to expedite the analysis of a DSM and the generation of clustering arrangements: conditional formatting, and the CALL function. Excel's conditional formatting has been used extensively to present DSMs in an easy to visualize form.[55] We wrote two C programs, and compiled them as dynamic link libraries to run them directly from Excel. Using the *CALL* function in Excel, we can easily pass data to the programs and get results back into the spreadsheet. We will explain how to use both features with an example.

Using conditional formatting, the format of a cell can be automatically adjusted according to the value of its contents. We have used this feature in two ways: to depict interactions in a DSM, and to show how teams are grouped into clusters. Up to three different levels of interactions between teams in a DSM can be shown visually in Excel. We've selected white, light-gray, and dark-gray to depict whether interactions are nonexistent, low or high. To accomplish this, we enter the limiting values for weak and strong interactions in 2 different cells in excel. Then we select all the entries in the DSM and apply the conditional formatting feature to the selected region. We create a white font and background for cells whose value is zero (i.e. those where teams have no interactions); a light-gray font and background for cells whose value is between zero and the top value for weak interactions; and a dark-gray font and background for strong

[54] Microsoft Excel 97 was used.

interactions (cells with values between weak and strong limits). We have also used conditional formatting to show in white or gray whether a team belongs to a cluster or not.

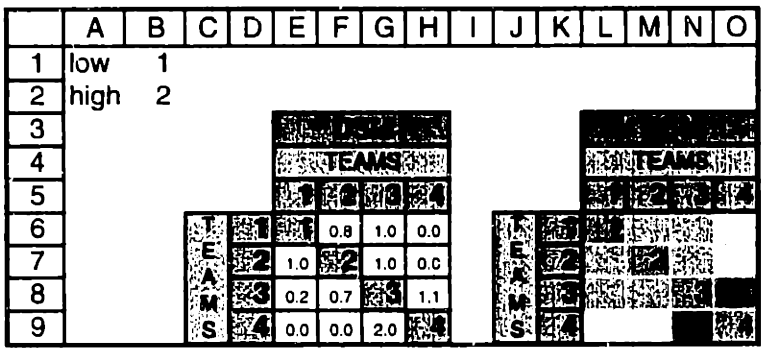


Figure A-1. DSM before and after conditional formatting using 1 for weak interactions and 2 for strong interactions.

Figure A-1 shows the use of conditional formatting to depict the level of interactions in the DSM. In cells B1 and B2 we have specified the limit values for weak and strong interactions as 1 and 2. The first DSM shows the values of each cell. The second DSM displays the same values using conditional formatting. Note that values ranging between 0 and 1 inclusive are shown in light gray, and those grater than 1 (and up to 2) are shown as dark gray. It is clearly more convenient to visualize the data in the second matrix.

Excel can exchange data and call external programs through the use of the *CALL* function. To use the call function we need two things: to compile the program that we want to call as a dynamic link library, and to decide the type of parameters that we are going to exchange.

The first requirement is easily met with Visual C++ by creating a project of type *dynamic link library*, incorporating all the files that we list later in this appendix, and compiling the program.[56] For example, the program to perform the clustering algorithm is compiled with the name *clusterdl.dll*. In the next page we show how to call this program from Excel.

---

[55] Conditional formatting is a new feature of Microsoft Excel 97.

[56] For specifics on how to create a dynamic link library refer to the manuals of Visual C++ for Windows.

Exchanging parameters requires more thoughtful planning when writing the program to be called from Excel. For the case of the clustering algorithm we are passing 10 different arguments from Excel to the clustering program, and we are getting the matrix with the clustering arrangement back to Excel. For the program to calculate the coordination cost, we are passing three arguments to the program, and getting one number back.

The specifics on how to setup a DSM problem in Excel can be seen in Figure A-2. It is convenient to prepare 4 square matrices to hold the DSM, the clustering result (Cl_Mat OUT), a dummy input matrix (Cl_Mat IN), and a second clustering matrix to manually enter cluster arrangements and observe the changes in coordination cost (Cl_Mat Manual Test).

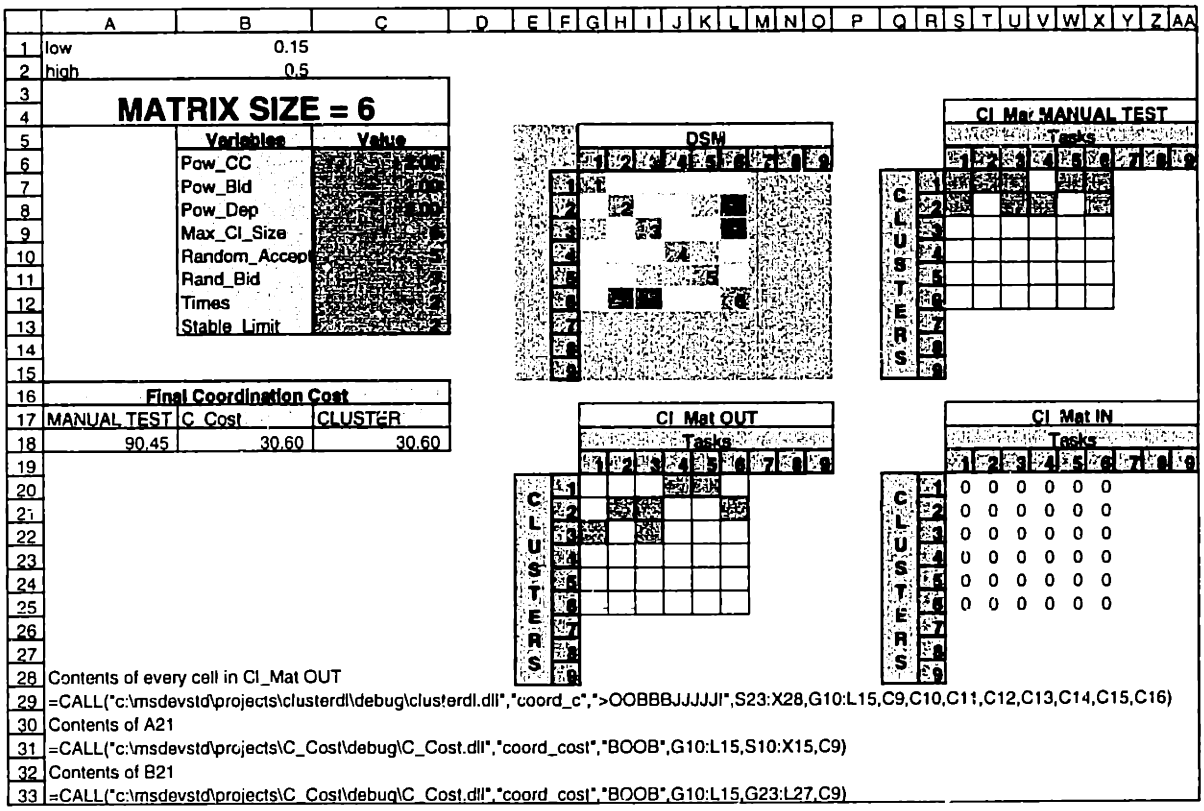


Figure A-2. Setting up a DSM in Excel to call the clustering algorithm and the program to calculate the total coordination cost.

Entering data into the *DSM*, *CL_Mat Manual Test*, and *Cl_Mat_IN* is straightforward. For the clustering matrix holding the result *CL_MAT OUT* we select all the cells in the matrix and enter the formula:

```
=CALL("c:\msdevstd\projects\clusterdl\debug\clusterdl.dll","coord_c"
,">OOBBBJJJJJ!",S23:X28,G10:L15,C9,C10,C11,C12,C13,C14,C15,C16)
```

Then we hold Ctrl-Shift-Enter to enter the same formula in all cells at once. The *CALL* formula contains the location of the clustering algorithm dynamic link library (the compiled program), the name of the function within our code that we are calling (coord_c in this case), the type of arguments passed to Excel (the ">" sign preceding the first O specifies that the first argument is also the return argument; O stands for arrays, B for doubles, and J for integers), an exclamation sign to recalculate every time the spreadsheet changes (if it is not included, the computer will take a very long time to recalculate the worksheet when any of the input arguments changes), and the location of the arguments.[57] The counterpart for the *CALL* function in the C code is the declaration of the function *coord_c* within our clustering program. This declaration is:

```
_declspec(dllexport)void coord_c(unsigned short int *n_row2,
          unsigned short int *n_col2, double *cl_mat, unsigned short
          int *n_rows, unsigned short int *n_cols, double *dsm,
          double pow_cc, double pow_bid, double pow_dep, int
          max_cl_size, int random_accept, int rand_bid, int times,
          int stable_limit)
```

Similarly for the function that calculates the coordination of a clustering arrangement the *CALL* function is:

```
CALL("c:\msdevstd\projects\C_Cost\debug\C_Cost.dll","coord_cost","BO
OB",G10:L15,S10:X15,C9)
```

or

```
=CALL("c:\msdevstd\projects\C_Cost\debug\C_Cost.dll","coord_cost","B
OOB",G10:L15,G23:L27,C9)
```

depending on whether we calculate the coordination cost for the manual matrix, or the output matrix. The C counterpart in the function *coord_cost* inside the program to calculate the total coordination cost is:

```
_declspec(dllexport)double coord_cost(unsigned short int *n_rows,
          unsigned short int *n_cols, double *dsm, unsigned short
          int *n_row2, unsigned short int *n_col2, double *cl_mat,
          double pow_cc)
```

[57] For specific usage of the CALL function and how to declare programs to be called from Excel refer to Excel's on-line help under the word CALL.

It should be noted that there's a one-to-one correspondence between the variables declared in Excel and C except for the arrays. An array in Excel translates into 3 variables in C (number of rows, columns, and a pointer to the contents of the array).

Designing a sheet similar to the one shown in Figure A-2 is the easiest way for someone to run the clustering algorithm and the program to calculate the total coordination cost. The remaining issues on how to setup such sheet should be obvious from looking at Figure A-2.

Clustering Algorithm

The clustering algorithm is composed of the following C files[58]:

- bid.c
- c_cost.c (this is the main program that calls all other functions)
- Coord_Cost.c
- Copy_Mat.c
- Create_Mat.c
- Delete_Clusters.c
- Delete_Clusters_1.c

The files are included in alphabetical order.

bid.c

```
#include <math.h>
#define OK                              1
#define CL_MAT(I,J)             (*(cl_mat    + ((I)-1)*size + ((J)-1) ))
#define CL_SIZE(I)              (*(cl_size   + ((I)-1) ))
#define DSM(I,J)                (*(dsm       + ((I)-1)*size + ((J)-1) ))
#define CL_BID(I)               (*((*cl_bid) + ((I)-1) ))

/* ======================================================  */
/* Function to calculate the bid matrix (cl_bid) from      */
/* the chosen task "task" to each cluster                  */

int Bid(double **cl_bid, double *dsm, int size, int n_clusters, int task,
              int *cl_size, double *cl_mat, double pow_bid, double pow_dep,
              int max_cl_size)
{
        int i, j;               /* counters */
        double in;                      /* upper diagonal dependency in the DSM */
        double out;             /* lower diagonal dependency in the DSM */

                                /* CL_BID(i) holds the bid for cluster i in cl_mat */
        for(i=1; i<=n_clusters; i++)
                CL_BID(i)=0.0;                          /* initialize cl_bid to 0 */
        for(i=1; i<=n_clusters; i++){
                in  = 0;
                out = 0;
                for(j=1; j<=size; j++){
```

[58] Both programs were written using C and complied in Microsoft Visual C++. For C usage refer to Lerman and/or Kelley.

```
                        if((CL_MAT(i,j)==1)&&(j!=task)){/* Task j is in cluster i; need
        j!=task to avoid diagonal */
                                if ( DSM(j,task) > 0)
                                        in  += DSM(j,task);
                                if ( DSM(task,j) > 0)
                                        out += DSM(task,j);
                        }
                }
                if ( (in>0) || (out>0) ){                       /* Accept the bid */
                        if ( CL_SIZE(i) == max_cl_size)
                                CL_BID(i)=0;/*pow((in+out),pow_dep)/( pow( (CL_SIZE(i)) ,
(pow_bid+2) ) );*/
                        else
                                CL_BID(i)=pow((in+out),pow_dep)/( pow( (CL_SIZE(i)) ,
pow_bid ) );
                }
        } /* for i */
        return(OK);
}
```

c_cost.c

```
#include <stdlib.h>
#include <math.h>
#include <time.h>

#define OK                              1
#define TEST                    if(f != OK) goto end
#define CL_MAT(I,J)             (*(cl_mat    + ((I)-1)*size + ((J)-1) ))
#define CL_MAT_C(I,J)   (*(cl_mat_c  + ((I)-1)*size + ((J)-1) ))
#define CL_SIZE(I)              (*(cl_size   + ((I)-1) ))
#define CL_SIZE_C(I)    (*(cl_size_c + ((I)-1) ))
#define CL_BID(I)               (*(cl_bid       + ((I)-1) ))
#define C_COST(I)               (*(c_cost    + ((I)-1) ))
#define C_COST_C(I)             (*(c_cost_c  + ((I)-1) ))
#define DSM(I,J)                (*(dsm       + ((I)-1)*size + ((J)-1) ))
#define RND_T_A(I)              (*(rnd_t_a      + ((I)-1) ))
#define CL_LIST(I,J)    (*(cl_list      + ((I)-1)*2    + ((J)-1) ))

int Create_Mat_d(double **, int, int);
int Create_Mat_i(int **   , int, int);
int Copy_Mat_d (double **, double *, int , int, int);
int Copy_Mat_i (int **   , int *   , int , int, int);
int Coord_Cost (double **, double *, int, int, int, int *, double *,  double);
int Bid                 (double **, double *, int, int, int, int*, double*, double, double,
int);
int Delete_Clusters  (int **, double **, int);
int Delete_Clusters_1(int **, double **, int);

_declspec(dllexport)void coord_c(unsigned short int *n_row2, unsigned short int *n_col2,
                                                                double *cl_mat,
                                                                unsigned short int *n_rows,
unsigned short int *n_cols,
                                                                double *dsm,
                                                                double pow_cc, double
pow_bid, double pow_dep, int max_cl_size,
                                                                int random_accept, int
rand_bid, int times, int stable_limit)
{
        int             size;                   /* # of tasks in the DSM
                */
        double  *cl_mat_c;              /* copy
                */
        int             *cl_size;               /* holds # of tasks in cluster matrix
        */
        int             *cl_size_c;             /* copy
                        */
        int             *cl_size_last; /* another copy
                        */
```

```
double *cl_bid;              /* holds bid for cluster matrix
*/
double  *cl_bid_c;           /* copy
                */
int             n_clusters;          /* number of clusters
        */
int             n_clusters_c;  /* copy
                */
double best_cl_bid;   /* holds best bid in cl_bid
*/
double sec_cl_bid;           /* holds second best bid in cl_bid
*/
int             i_best_cl_bid; /* holds the position of the best bid        */
int             accept;              /* controls acceptance of bids
        */
int             accept1;             /* controls changes    to the cluster matrix
*/
double *c_cost;              /* coordination cost array
*/
double *c_cost_c;            /* copy
                */
double initial_total_c_cost; /* initial total coordination cost    */
double total_c_cost;  /* total coordination cost
*/
double total_c_cost_c;       /* copy
                */
double best_c_cost;   /* best total_c_cost so far
*/
int             *rnd_t_a;            /* Random task array
        */
int             stable;              /* Check for stability of system
*/
int             change;              /* # of changes in system
        */
int             task;                /* chosen task
                */
int             t1;                          /* task index
                        */
int             *cl_list;            /* Temp storage for a list of clusters
*/
int             f;                           /* flag to test for errors
                */
int             i,j,k;               /* counters
                        */

/* CREATE and INITIALIZE cl_size and c_cost matrices */

accept1=0;
size=*n_cols;
n_clusters=size;
n_clusters_c=size;
f=Create_Mat_d( &c_cost   , size, 1   ); TEST;
f=Create_Mat_i( &cl_size  , size, 1   ); TEST;
for (i=1;i<=size;i++){
        for (j=1;j<=size;j++){
                if (CL_MAT(i,j)==1)
                        CL_SIZE(i)+=1;
                else CL_MAT(i,j)=0;
        }
}

/* All matrices are automatically initialized to 0 or 0.0 */

f=Create_Mat_d( &cl_mat_c , size, size); TEST;
f=Create_Mat_i( &cl_size_c, size, 1   ); TEST;
f=Create_Mat_i( &cl_size_last, size, 1); TEST;
f=Create_Mat_d( &cl_bid   , size, 1   ); TEST;
f=Create_Mat_d( &cl_bid_c , size, 1   ); TEST;
f=Create_Mat_d( &c_cost_c , size, 1   ); TEST;
f=Create_Mat_i( &rnd_t_a  , size, 1   ); TEST;
f=Create_Mat_i( &cl_list  , size, 2   ); TEST;
```

```
        /* INITIALIZE the CLUSTER MATRIX */

        /* if CL_MAT(i,j)=1, then task j is in cluster i */
        /* if CL_MAT(i,j)=0, then taks j is not in cluster i */
        for(i=1;i<=n_clusters;i++){
                CL_MAT(i,i)=1;                  /* Each cluster has 1 task (along diagonal)
*/
                CL_SIZE(i)=1;                   /* The size of cluster i is 1 */
        }
        /* DETERMINE INITIAL COORDINATION COST */

        f=Coord_Cost(&c_cost, &total_c_cost, dsm, size, 0,
                cl_size, cl_mat, pow_cc); TEST;
        initial_total_c_cost=total_c_cost;
        best_c_cost = total_c_cost;

        /* INITIALIZE the RANDOM TASK ARRAY and SYSTEM STABILITY */

        srand(time(NULL));
        for(i=1;i<=size;i++)
                RND_T_A(i) = 1 + rand()%size; /* random #s between 1 & size */
        stable = 0;             /* Initialize the system to be unstable    */
        t1=1;
        change=0;

        while (stable != stable_limit){
                for (k=1;k<=size*times;k++){

                        /* 1. PICK TASK in LOCATION t1 */

                        task          = RND_T_A(t1);            /* Select a task index
*/
                        RND_T_A(t1) = 1 + rand()%size;          /* Generate a new task index
*/
                        t1 = 1 + (t1)%size;                             /* Move the task index
(1<=t1<=N) */

                        /* 2. ACCEPT BIDS FOR TASK (task) FROM REMAINING CLUSTERS */

                        /* CL_BID(i) holds the bid for cluster i in cl_mat */
                        f = Bid(&cl_bid, dsm, size, n_clusters, task, cl_size, cl_mat,
                                pow_bid, pow_dep, max_cl_size); TEST;

                        /* 3. DETERMINE THE BEST BID */

                        best_cl_bid=0.0;
                        for (i=1; i<=n_clusters; i++){
                                if (CL_BID(i)>best_cl_bid)
                                        best_cl_bid=CL_BID(i); /* Search for the cluster
with the best bid */
                        }
                        sec_cl_bid=0.0;
                        for (i=1; i<=n_clusters; i++){
                                if (CL_BID(i)>sec_cl_bid && i_best_cl_bid!=i)
                                        sec_cl_bid=CL_BID(i);  /* Search for the cluster
with the second best bid */
                        }
                        if(rand()%(rand_bid+1)==0)              /* pick the second best bid 1
out */
                                best_cl_bid=sec_cl_bid;                 /* of rand_bid times
*/
                        if (best_cl_bid>0)
                                accept=1;
                        else
                                accept=0;

                        if (accept){
                                n_clusters=size;
                                n_clusters_c=n_clusters;
```

```
                        /* 3a. Determine if the BID is acceptable */

                        /* Initialize */
                        for(i=1; i<=n_clusters; i++){
                                CL_LIST(i,1)=0;
                                CL_LIST(i,2)=0;
                        }

                        /* Determine the list of clusters affected */

                        for(i=1; i<=n_clusters; i++){
                                if( CL_BID(i)==best_cl_bid && CL_MAT(i,task)==0 ){
                                        CL_LIST(i,1) = 1;       /* ADD cluster i to
the list            */
                                        CL_LIST(i,2) = 1;       /* a task may be ADDED
to cluster i      */
                                }
                        }

                        /* Copy the cluster matrix */

                        f=Copy_Mat_d(&cl_mat_c , cl_mat , size, size, 0); TEST;
                        f=Copy_Mat_i(&cl_size_c, cl_size, size, 1    , 0); TEST;

                        /* Determine the cluster matrix after change */

                        for(i=1; i<=n_clusters; i++){
                                if(CL_LIST(i,1) == 1){
                                        if(CL_LIST(i,2) == 1){                  /* ADD
task t to cluster i */
                                                CL_MAT_C(i,task) = 1;
                                                CL_SIZE_C(i)     = CL_SIZE_C(i) + 1;
                                        }
                                }
                        }
                        /* Determine the change in coordination costs */

                        f=Delete_Clusters(&cl_size_c, &cl_mat_c, size); TEST;


        f=Coord_Cost(&c_cost_c,&total_c_cost_c,dsm,size,n_clusters_c,cl_size_c,cl_mat_c,po
w_cc); TEST;
                        if(total_c_cost_c <= total_c_cost)
                                accept1 = 1;
                        else{
                                if(rand()%(random_accept+1) == 0) /* still accept 1
out of approx. random_accept times */
                                        accept1 = 1;
                                else
                                        accept1 = 0;
                        }
                }       /* if accept */
                if (accept1) {
                        accept1=0;
                        CL_BID(task) = best_cl_bid;

                        /* 4. UPDATE THE CLUSTERS */

                        total_c_cost = total_c_cost_c;
                        f=Copy_Mat_d(&cl_mat          , cl_mat_c , size, size, 0);
                        f=Copy_Mat_i(&cl_size         , cl_size_c, size, 1   , 0);
                        f=Copy_Mat_i(&cl_size_last    , cl_size_c, size, 1   , 0);
                        n_clusters=n_clusters_c;

                        if( best_c_cost>total_c_cost ){
                                best_c_cost = total_c_cost;
                                change      = change + 1;       /* improvement in
coord. cost */
                        }
                }       /* if accept */
        } /* for (k=1;k<=size;k++) */
```

```
                /* 5. TEST THE SYSTEM FOR STABILITY */

                if (change > 0){        /* ie if there was an improvement in coord. costs */
                        stable = 0;             /* system is unstable */
                        change = 0;
                }
                else
                        stable = stable + 1;
        }       /* while (stable != STABLE_LIMIT) */
        f=Delete_Clusters_1(&cl_size_last, &cl_mat, size); TEST; /* needed for excel to
refresh matrix */
        goto end2;
end: CL_MAT(size,1)=-1;
end2: f=0;
}
```

Coord_Cost.c

```
#include <stdlib.h>
#include <math.h>

#define OK                              1
#define CL_MAT(I,J)             (*(cl_mat    + ((I)-1)*size + ((J)-1) ))
#define CL_SIZE(I)              (*(cl_size   + ((I)-1) ))
#define C_COST(I)               (*((*c_cost) + ((I)-1) ))
#define DSM(I,J)                (*(dsm       + ((I)-1)*size + ((J)-1) ))

/* ================================================== */
/* Function to calculate the coordination cost matrix     */
/* (c_cost) and the total coordination cost               */

int Coord_Cost(double **c_cost, double *total_c_cost, double *dsm, int size,
                        int n_clusters, int *cl_size, double * cl_mat, double pow_cc)
{
        int i, j, l;    /* counters */
        int size_cl;    /* temp size of the cluster matrix */
        double cost_c; /* temporary cost storage */
        double cost_t; /* temporary cost storage */

        *total_c_cost = 0.0;
        for(i=1;i<=size;i++)
                C_COST(i) = 0.0;
        for(i=1; i<=size; i++){
                for(j=i+1; j<=size; j++){               /* j=1+i to skip the diagonal
terms */
                        if(DSM(i,j)>0 || DSM(j,i)>0){ /* dependency exist between tasks i &
j */
                                size_cl=size;
                                cost_t=0.0;
                                for(l=1; l<=n_clusters; l++){
                                        if( (CL_MAT(l,i)+CL_MAT(l,j)==2) ){
                                                /* i.e. if both i & j belong to the same
cluster l */
                                                size_cl = CL_SIZE(l);
                                                cost_t  = cost_t + ( DSM(i,j) + DSM(j,i) ) *
pow(size_cl,pow_cc);
                                        }
                                } /* for l */
                                if (cost_t>0.0)
                                        cost_c=cost_t;
                                else
                                        cost_c=( DSM(i,j) + DSM(j,i) ) * pow(size,pow_cc);
                                C_COST(i)    += cost_c;
                        }
                } /* for j */
        }  /* for i */
        for (i=1; i<=size; i++)
                *total_c_cost += C_COST(i);
```

```
        return(OK);
}
```

Copy_Mat.c

```
#include <stdlib.h>

#define OK              1
#define INPUT(I,J,K) (*( input_mat + ((J)-1) + n_cols*((I)-1) + n_cols*(K) ))
#define MAT(I,J)     (*( (*mat)    + ((J)-1) + n_cols*((I)-1) ))


/* ========================================================  */
/* Function to copy all or part of a matrix (dsm) to a     */
/* newly created matrix (mat), where n_cols=number of      */
/* columns in the dsm, n_rows=number of rows to be copied */
/* and skip_rows=number of rows to skip before copying.    */
/* Matrices are of type double                             */

int Copy_Mat_d(double **mat, double *input_mat, int n_rows,
                        int n_cols, int skip_rows)
{
        int i, j;
        for (i=1;i<=n_rows;i++){
                for (j=1;j<=n_cols;j++)
                        MAT(i,j)=INPUT(i,j,skip_rows);
        }
        return(OK);
}

/* ========================================================  */
/* Function to copy all or part of a matrix (dsm) to a     */
/* newly created matrix (mat), where n_cols=number of      */
/* columns in the dsm, n_rows=number of rows to be copied */
/* and skip_rows=number of rows to skip before copying.    */
/* Matrices are of type int                                */

int Copy_Mat_i(int **mat, int *input_mat, int n_rows,
                        int n_cols, int skip_rows)
{
        int i, j;
        for (i=1;i<=n_rows;i++){
                for (j=1;j<=n_cols;j++)
                        MAT(i,j)=INPUT(i,j,skip_rows);
        }
        return(OK);
}
```

Create_Mat.c

```
#include <stdlib.h>
#define OK 1
#define MAT(I,J)    (*( (*mat)    + ((J)-1) + n_cols*((I)-1) ))


/* ========================================================  */
/* Function to allocate memory and create a matrix mat of */
/* size n_rows by n_cols where each element is of type     */
/* double. The matrix elements are initialized to zero     */

int Create_Mat_d(double **mat, int n_rows, int n_cols)
{
        int i, j;
        *mat=malloc(n_rows*n_cols*sizeof(double));
        if (*mat==NULL) return(-10);
        for (i=1;i<=n_rows;i++){
                for (j=1;j<=n_cols;j++)
```

```
                        MAT(i,j)=0.0;
        }
        return(OK);
}


/* ======================================================  */
/* Function to allocate memory and create a matrix mat of */
/* size n_rows by n_cols where each element is of type    */
/* int. The matrix elements are initialized to zero       */


int Create_Mat_i(int **mat, int n_rows, int n_cols)
{
        int i, j;
        *mat=malloc(n_rows*n_cols*sizeof(int));
        if (*mat==NULL) return(-10);
        for (i=1;i<=n_rows;i++){
                for (j=1;j<=n_cols;j++)
                        MAT(i,j)=0;
        }
        return(OK);
}
```

Delete_Clusters.c

```
#include <math.h>

#define OK                              1
#define CL_MAT(I,J)             (*((*cl_mat) + ((I)-1)*size + ((J)-1) ))
#define CL_SIZE(I)              (*((*cl_size)+ ((I)-1) ))
#define DSM(I,J)                (*(dsm       + ((I)-1)*size + ((J)-1) ))
#define COPIES_TRUE             1
#define CONTAINED_TRUE 1

/* ======================================================  */
/* Function to delete copies of clusters, clusters that   */
/* are contained inside other clusters,and empty clusters */

int Delete_Clusters(int ** cl_size, double ** cl_mat, int size)
{
        int i, j, k, l;         /* counters */
        int n_clusters;
        n_clusters=size;

        /* Delete clusters contained in other clusters forward */

        if(CONTAINED_TRUE){
                for(i=1; i<=n_clusters; i++){
                        for(j=i+1; j<=n_clusters; j++){
                                if(CL_SIZE(i)>CL_SIZE(j) && CL_SIZE(j)>0){
                                        l=0;
                                        while( l<size && CL_MAT(i,l+1)>=CL_MAT(j,l+1) )
                                                l++;
                                        if( l==size ){ /* clusters j is contained in i */
                                                for(k=1; k<=size; k++)
                                                        CL_MAT(j,k) = 0; /* erase the second
copy of the cluster */
                                                CL_SIZE(j) = 0;
                                        }
                                }
                        }
                } /* for i */

                /* Delete clusters contained in other clusters backward */
                for(i=1; i<=n_clusters; i++){
                        for(j=i+1; j<=n_clusters; j++){
                                if(CL_SIZE(i)<CL_SIZE(j) && CL_SIZE(i)>0){
```

```
                                        l=0;
                                        while( l<size && CL_MAT(i,l+1)<=CL_MAT(j,l+1) )
                                                l++;
                                        if( l==size ){ /* cluster i is contained in j */
                                                for(k=1; k<=size; k++)
                                                        CL_MAT(i,k) = 0; /* erase the second
copy of the cluster */
                                                CL_SIZE(i) = 0;
                                        }
                                }
                        }
                } /* for i */
        }
        /* Delete tasks in the copies of clusters */
        if (COPIES_TRUE){
                for(i=1; i<=n_clusters; i++){
                        for(j=i+1; j<=n_clusters; j++){
                                if(CL_SIZE(i)==CL_SIZE(j)){ /* check if the clusters are
identical */
                                        l=0;
                                        while( l<size && CL_MAT(i,l+1)==CL_MAT(j,l+1) )
                                                l++;
                                        if( l==size ){ /* the clusters are identical */
                                                for(k=1; k<=size; k++)
                                                        CL_MAT(j,k) = 0; /* erase the second
copy of the cluster */
                                                CL_SIZE(j) = 0;
                                        }
                                }
                        }
                } /* for i */
        }
        /* Delete clusters with no tasks */
        i=1;
        j=n_clusters;
        while (i<j){
                while( i<=j && CL_SIZE(i)>0 )
                        i++;                                            /* cluster i is not
empty */
                while( i<=j && CL_SIZE(j)==0 )
                        j = j-1;                                        /* cluster j is
        empty */
                if( i<j ){
                        for(l=1; l<=size; l++){                 /* swap clusters i & j, &
delete j */
                                CL_MAT(i,l) = CL_MAT(j,l);
                                CL_MAT(j,l) = 0;
                        }
                        CL_SIZE(i) = CL_SIZE(j);
                        CL_SIZE(j) = 0;
                        j = j-1;
                }
        } /* while i<j */
        return (OK);
}
```

Delete_Clusters_1.c

```
#include <math.h>

#define OK                              1
#define CL_MAT(I,J)             (*((*cl_mat) + ((I)-1)*size + ((J)-1) ))
#define CL_SIZE(I)              (*((*cl_size)+ ((I)-1) ))

/* ======================================================  */
/* Function to delete clusters of size 1                   */

int Delete_Clusters_1(int ** cl_size, double ** cl_mat, int size)
```

```
{
        int i, j, l;    /* counters */
        int n_clusters;
        n_clusters=size;

                                /* Delete clusters with 1 task */
        for(i=1; i<= n_clusters; i++){
                if(CL_SIZE(i)==1){              /* j contained in i */
                        for(j=1;j<=size;j++)
                                CL_MAT(i,j)=0;
                        CL_SIZE(i)=0;
                }
        }
                                /* Delete clusters with no tasks */

        i=1;
        j=n_clusters;
        while (i<j){
                while( i<=j && CL_SIZE(i)>0 )
                        i++;                                            /* cluster i is not
empty */
                while( i<=j && CL_SIZE(j)==0 )
                        j = j-1;                                        /* cluster j is
        empty */
                if( i<j ){
                        for(l=1; l<=size; l++){                 /* swap clusters i & j, &
delete j */
                                CL_MAT(i,l) = CL_MAT(j,l);
                                CL_MAT(j,l) = 0;
                        }
                        CL_SIZE(i) = CL_SIZE(j);
                        CL_SIZE(j) = 0;
                        j = j-1;
                }
        } /* while i<j */
        return (OK);
}
```

Program to Calculate the Total Coordination Cost

The program to calculate the Total Coordination Cost contains the following files:

- c_cost.c (this is the main program that calls all other functions)
- Coord_Cost.c
- Create_Mat.c

c_cost.c

```
/* This program calculates the coordination cost of a DSM passed to it from Excel   */
/* It takes three arguments, the DSM, the cluster matrix, and the power used in the */
/* coordination cost function. It returns a single value: the coordination cost     */
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <stdio.h>

#define TEST                    if(f != OK) goto end
#define CL_MAT(I,J)             (*(cl_mat    + ((I)-1)*size + ((J)-1) ))
#define CL_SIZE(I)              (*(cl_size   + ((I)-1) ))
#define C_COST(I)               (*(c_cost    + ((I)-1) ))
#define DSM(I,J)                (*(dsm       + ((I)-1)*size + ((J)-1) ))
```

```
int Create_Mat_d(double **, int, int);
int Create_Mat_i(int **   , int, int);
int Coord_Costd(double **, double *, double *, int, int, int *, double *,  double);

_declspec(dllexport)double coord_cost(unsigned short int *n_rows, unsigned short int
*n_cols, double *dsm, unsigned short int *n_row2, unsigned short int *n_col2, double
*cl_mat, double pow_cc)
{
        int             n_rows_i;       /* input matrix number of rows        */
        int             n_cols_i;       /* input matrix number of columns     */
        int             size;           /* # of tasks in the DSM */
        int             *cl_size;       /* holds # of tasks in cluster matrix */
        int             n_clusters;     /* number of clusters                 */
        double *c_cost;          /* coordination cost array  */
        double total_c_cost;  /* total for c_cost         */
        int             f;                      /* flag to test for errors */
        int             i,j;            /* counters */

        n_rows_i= *n_rows;
        n_cols_i= *n_cols;
        size=n_cols_i;
        n_clusters=*n_row2;
                /* All matrices are automatically initialized to 0 or 0.0 */
        f=Create_Mat_i( &cl_size  , size, 1   ); TEST;
        f=Create_Mat_d( &c_cost   , size, 1   ); TEST;
        for (i=1;i<=n_clusters;i++){
                for (j=1;j<=size;j++){
                        if (CL_MAT(i,j)==1)
                                CL_SIZE(i)+=1;
                }
        }
        f=Coord_Costd(&c_cost, &total_c_cost, dsm, size, n_clusters,
                cl_size, cl_mat, pow_cc); TEST;
                goto end2;
end: return(-1);
end2: return(total_c_cost);
}
```

Coord_Cost.c

```
#include <stdlib.h>
#include <math.h>

#define OK      1
#define CL_MAT(I,J)             (*(cl_mat    + ((I)-1)*size + ((J)-1) ))
#define CL_SIZE(I)              (*(cl_size   + ((I)-1) ))
#define C_COST(I)               (*((*c_cost) + ((I)-1) ))
#define DSM(I,J)                (*(dsm       + ((I)-1)*size + ((J)-1) ))

/* ======================================================  */
/* Function to calculate the coordination cost matrix     */
/* (c_cost) and the total coordination cost               */

int Coord_Costd(double **c_cost, double *total_c_cost, double *dsm, int size,
                        int n_clusters, int *cl_size, double *cl_mat, double pow_cc)
{
        int i, j, l;    /* counters */
        int size_cl;    /* temp size of the cluster matrix */
        double cost_c; /* temporary cost storage */
        double cost_t; /* temporary cost storage */

        *total_c_cost = 0.0;
        for(i=1;i<=size;i++)
                C_COST(i) = 0.0;
        for(i=1; i<=size; i++){
                for(j=i+1; j<=size; j++){               /* j=1+i to skip the diagonal
terms */
                        if(DSM(i,j)>0 || DSM(j,i)>0){ /* dependency exist between tasks i &
j */
```

```
                                size_cl=size;
                                cost_t=0.0;
                                for(l=1; l<=n_clusters; l++){
                                        if(
(CL_MAT(l,i)+CL_MAT(l,j))>1.9&&(CL_MAT(i,i)+CL_MAT(l,j))<2.1 ){
                                                /* i.e. if both i & j belong to the same
cluster l */
                                                size_cl = CL_SIZE(l);
                                                cost_t  = cost_t + ( DSM(i,j) + DSM(j,i) ) *
pow(size_cl,pow_cc);
                                        }
                                } /* for l */
                                if (cost_t>0.0)
                                        cost_c=cost_t;
                                else
                                        cost_c=( DSM(i,j) + DSM(j,i) ) * pow(size,pow_cc);
                                C_COST(i)   += cost_c;
                        }
                } /* for j */
        }  /* for i */
        for (i=1; i<=size; i++)
                *total_c_cost += C_COST(i);
        return(OK);
}
```

Create_Mat.c

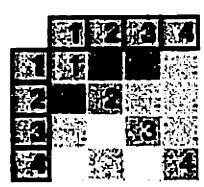
```
#include <stdlib.h>

#define OK      1
#define MAT(I,J)    (*( (*mat)    + ((J)-1) + n_cols*((I)-1) ))


/* ========================================================  */
/* Function to allocate memory and create a matrix mat of */
/* size n_rows by n_cols where each element is of type    */
/* double. The matrix elements are initialized to zero    */

int Create_Mat_d(double **mat, int n_rows, int n_cols)
{
        int i, j;
        *mat=malloc(n_rows*n_cols*sizeof(double));
        if (*mat==NULL) return(-1);
        for (i=1;i<=n_rows;i++){
                for (j=1;j<=n_cols;j++)
                        MAT(i,j)=0.0;
        }
        return(OK);
}


/* ========================================================  */
/* Function to allocate memory and create a matrix mat of */
/* size n_rows by n_cols where each element is of type    */
/* int. The matrix elements are initialized to zero       */


int Create_Mat_i(int **mat, int n_rows, int n_cols)
{
        int i, j;
        *mat=malloc(n_rows*n_cols*sizeof(int));
        if (*mat==NULL) return(-1);
        for (i=1;i<=n_rows;i++){
                for (j=1;j<=n_cols;j++)
                        MAT(i,j)=0;
        }
        return(OK);
}
```

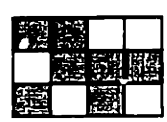
Appendix B. Comparison Tests

In this Appendix we include the seventy test cases that were summarized in Chapter 4. First, we show all the data that was captured with a sample test case. Then, we list all seventy cases.

Explanation of Tests

Figure B-1 shows the way in which a comparison test case was recorded. The randomly generated matrix is shown on top, in this case we have a DSM with four teams. Then we list the optimal configuration given by the integer program for the number of clusters required (3 in this case), and the corresponding optimal total coordination cost (14.40). Then we have two columns of numbers. The left-hand column has the average cost obtained by the ten runs of the algorithm, followed by the individual cost of each run. The right-hand column lists "-", or "1" depending on whether the optimal solution was found. Keeping the parameters used to obtain the ten runs fixed, a hundred runs were made to observe how many times the solutions produced 1, 2, 3, or 4 clusters (30, 65, 5, 0 respectively in this case). Comparing between test cases allowed us to see how these parameters favored certain types of solutions.





ip obj	14.40	
opt found=1	16.1	-
# clusters=1	15.0	30
# clusters=2	15.0	65
# clusters=3	15.0	5
# clusters=4	20.4	-
% from opt.	21.0	4.2%
# opt found	15.0	-
# below opt	15.0	-
pow_bid	15.0	2.00
random_accept	15.0	40
random_bid	15.0	2

Figure B-1. Sample test case.

Then, the percent difference between the best solution found by the algorithm and the integer program is listed (4.2% in Figure B-1). The number of times (out of the ten recorded solutions) that the algorithm found the optimal solution, and the number of times that the algorithm improves on a solution found by the integer program, are calculated from the data (*# opt found* = 0 and *# below opt* = 0 in the example). Finally, the values of parameters *pow_bid*, *random_accept*, and *random_bid* are included (2.00, 40, and 2 respectively). The value of these parameters was held constant to capture the ten algorithm runs, and the range of cluster sizes in a hundred runs.

For all seventy cases, the value of *pow_cc* was fixed at 2.0. The value of parameters *times* and *stable_limit* was either 1 or 2, and is noted in the next pages. Finally, some comments were made in some of the cases; these are above the cost obtained by the integer program (no comment in Figure B-1).

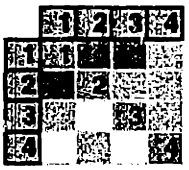
Test Cases

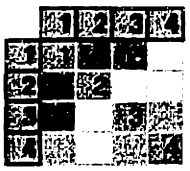
The seventy cases are shown in the following order:

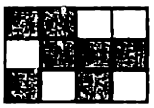
1. 10 distinct matrices of size 4 with 10 cases having 3 clusters, 10 having 2 clusters, and 10 having 1 cluster, for a total of 30 cases.
2. 10 distinct matrices of size 5 with 10 cases having 3 clusters, and 10 having 2 clusters, for a total of 20 cases.
3. 10 distinct matrices of size 6 with 2 clusters, and 10 distinct matrices of size 7 with 2 clusters, for a total of 20 cases.

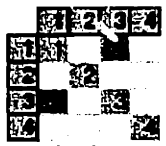
Within each category, the cases are listed in the same order as they were reported in Figure 4-2 in Chapter 4.

from here down used
times=1 and stable_limit=1
unless otherwise stated



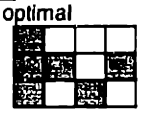


ip obj		14.40		11.60	
opt found=1	16.1	-		11.6	1
# clusters=1	15.0	30		11.6	22
# clusters=2	15.0	65		11.6	71
# clusters=3	15.0	5		11.6	7
# clusters=4	20.4	-		11.6	-
% from opt.	21.0	4.2%		11.6	0.0%
# opt found	15.0	-		11.6	10
# below opt	15.0	-		11.6	-
pow_bid	15.0	2.00		11.6	2.00
random_accept	15.0	40		11.6	40
random_bid	15.0	2		11.6	5
ip obj		18.6		15.2	
opt found=1	19.6	1		19.1	1
# clusters=1	18.6	40		21.6	25
# clusters=2	22.2	60		15.2	75
# clusters=3	22.2	-		31.1	-
# clusters=4	18.6	-		17.1	-
% from opt.	18.6	0.0%		15.2	0.0%
# opt found	20.5	6		21.6	4
# below opt	19.8	-		15.2	-
pow_bid	18.6	2.00		17.1	2.00
random_accept	18.6	10		21.6	5
random_bid	18.6	5		15.2	5
ip obj		25.8		25.2	
opt found=1	33.4	1		29.4	1
# clusters=1	25.8	28		25.2	46
# clusters=2	38.4	72		25.2	54
# clusters=3	38.4	-		39.2	-
# clusters=4	38.4	-		25.2	-
% from opt.	38.4	0.0%		25.2	0.0%
# opt found	25.8	4		39.2	7
# below opt	25.8	-		25.2	-
pow_bid	38.4	2.00		25.2	2.00
random_accept	25.8	5		39.2	5
random_bid	38.4	5		25.2	1



counts as optimal

4.0

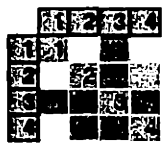


1

0.0%

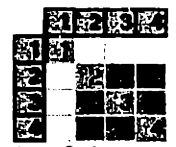
10

-



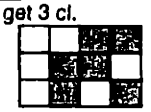
22.40

22.4	1
22.4	11
22.4	55
22.4	34
22.4	-
22.4	0.0%
22.4	10
22.4	-
22.4	2.00
22.4	10
22.4	5



could not get 3 cl.

12.0



24.0

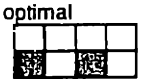
100.0%

-

-

counts as optimal

4.00



1

0.0%

10

-

27.9

31.8	1
32.4	13
36.6	76
36.6	11
27.9	-
27.9	0.0%
27.9	-
27.9	-
27.9	2.00
36.6	10
36.6	10

24.0

24.0	1
24.0	14
24.0	88
24.0	-
24.0	-
24.0	0.0%
24.0	10
24.0	-
24.0	2.00
24.0	10
24.0	5

it always gets there

4.0

4.0	1
4.0	
4.0	
4.0	
4.0	
4.0	0.0%
4.0	10
4.0	-
4.0	2.00
4.0	5
4.0	5

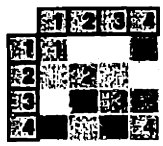
pow_bid=1 to get there easily!

39.9

47.7	1
39.9	28
58.4	51
58.4	21
39.9	-
58.4	0.0%
39.9	5
58.4	-
39.9	1.00
44.4	5
39.9	2

27.0

27.0	1
27.0	41
27.0	58
27.0	-
27.0	-
27.0	0.0%
27.0	10
27.0	-
27.0	2.00
27.0	10
27.0	5



15.40

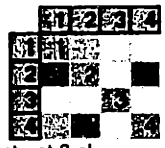
15.4	1
15.4	-
15.4	58
15.4	42
15.4	-
15.4	0.0%
15.4	10
15.4	-
15.4	2.00
15.4	30
15.4	20

22.6

23.8	1
25.4	10
22.6	75
22.6	15
22.6	-
25.4	0.0%
25.4	5
22.6	-
25.4	2.00
23.2	10
22.6	5

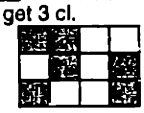
33.2

40.3	1
33.2	7
47.2	79
47.2	14
33.2	-
47.2	0.0%
33.2	1
34.6	-
33.2	2.00
47.2	5
47.2	5



could not get 3 cl.

7.2



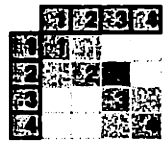
9.0	-
	25.0%
	-
	-

9.0

9.0	1
9.0	56
9.0	44
9.0	-
9.0	-
9.0	0.0%
9.0	10
9.0	-
9.0	2.00
9.0	10
9.0	5

16.2

16.2	1
16.2	58
16.2	42
16.2	-
16.2	-
16.2	0.0%
16.2	10
16.2	-
16.2	2.00
16.2	5
16.2	5



4.40

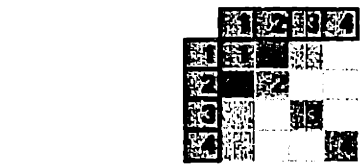
4.4	1
4.4	6
4.4	60
4.4	34
4.4	-
4.4	0.0%
4.4	10
4.4	-
4.4	2.00
4.4	20
4.4	10

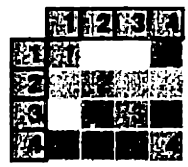
8.0

8.3	1
8.4	-
8.4	62
8.4	38
8.4	-
8.0	0.0%
8.4	3
8.0	-
8.0	2.00
8.4	10
8.4	10

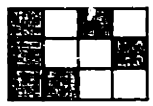
11.6

16.5	-
17.6	36
17.6	64
17.6	-
17.6	-
17.6	3.4%
17.6	-
17.6	-
12.0	2.00
12.0	1
17.6	1





5.80



22.80

5.8	1
5.8	68
5.8	22
5.8	10
5.8	-
5.8	0.0%
5.8	10
5.8	-
5.8	2.00
5.8	20
5.8	2

23.4	-
23.4	1
23.4	62
23.4	37
23.4	-
23.4	2.6%
23.4	-
23.4	-
23.4	2.00
23.4	30
23.4	5

7.6

27.1

9.7	1
9.4	45
8.1	41
7.6	14
7.6	-
17.8	0.0%
7.6	6
7.6	-
7.6	2.00
15.8	10
7.6	2

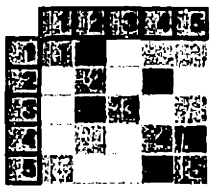
29.9	1
33.4	10
27.1	74
27.1	16
27.1	-
27.1	0.0%
34.8	6
27.1	-
34.8	2.00
33.4	10
27.1	5

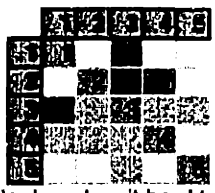
11.2

39.1

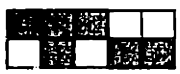
14.4	1
14.1	57
14.1	43
14.1	-
23.2	-
11.2	0.0%
11.2	3
15.2	-
15.2	2.00
14.1	20
11.2	2

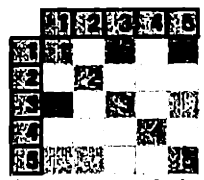
48.0	1
41.2	28
41.2	51
41.2	21
55.2	-
55.2	0.0%
41.2	1
55.2	-
55.2	2.00
55.2	5
39.1	2





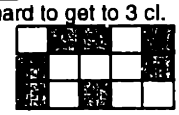
incr. times or stable_l. makes it hard to get 3 cl.

ip obj		28.4		23.3	
opt found=1	33.5	1		30.1	-
# clusters=1	33.9	-		28.0	20
# clusters=2	37.3	37		28.7	54
# clusters=3	30.5	63		26.4	26
# clusters=4	33.9	-		35.3	-
% from opt.	33.9	0.0%		35.3	0.4%
# opt found	28.4	1		28.0	-
# below opt	36.2	-		35.3	-
pow_bid	33.9	2.00		26.4	2.00
random_accept	33.9	10		34.1	10
random_bid	33.9	2		23.4	5
ip obj		38.9		29.7	
opt found=1	43.7	1		36.5	1
# clusters=1	38.9	5		29.9	5
# clusters=2	39.7	57		36.2	78
# clusters=3	42.1	38		41.8	17
# clusters=4	44.4	-		41.6	-
% from opt.	46.9	0.0%		41.6	0.0%
# opt found	38.9	2		36.2	1
# below opt	57.1	-		29.9	-
pow_bid	44.3	2.00		29.7	2.00
random_accept	41.0	5		41.6	10
random_bid	44.3	10		36.2	2



hard to get to 3 cl.

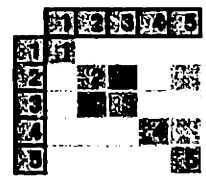
9.3



11.0	-
11.0	35
11.0	53
11.0	12
11.0	0.0%
11.0	17.7%
11.0	-
11.0	-
11.0	2.00
11.0	100
11.0	1

14.1

16.5	1
16.8	33
15.0	63
16.8	4
14.1	-
16.8	0.0%
16.8	3
14.1	-
14.1	2.00
16.8	10
23.4	5



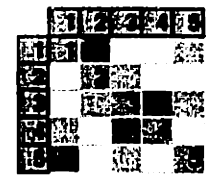
5.2

5.2	1
5.2	88
5.2	12
5.2	0.0%
5.2	10
5.2	-
5.2	2.00
5.2	10
5.2	5

used times 2 and stable 2

6.7

8.5	1
8.4	91
8.4	9
6.7	-
8.4	0.0%
11.0	2
8.4	-
8.4	2.00
11.0	5
6.7	5

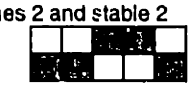


23.5

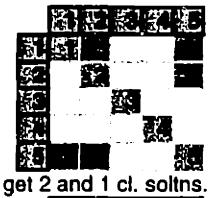
27.5	1
23.5	67
26.8	31
33.1	2
27.4	0.0%
27.4	1
27.4	-
27.4	2.00
27.6	10
26.8	5

used times 2 and stable 2

33.1



34.4	1
33.1	1
35.8	78
33.1	21
33.1	0.0%
33.1	2
33.3	-
43.5	2.00
33.1	5
33.1	5



could get 2 and 1 cl. soltns. only

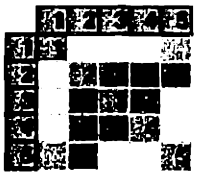
10.0

-

20.5

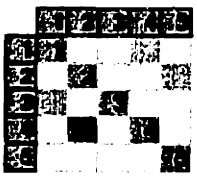
105.0%

-



32.2

44.2	1
48.2	-
46.2	75
48.2	25
32.2	-
46.2	0.0%
48.2	5
46.2	-
32.2	2.00
48.2	20
46.2	5



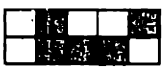
5.3

7.5	-
7.0	-
7.1	59
7.0	41
7.0	-
7.0	31.1%
12.3	-
7.0	-
7.0	2.00
7.0	20
7.1	5

20.5

20.5	1
20.5	65
20.5	35
20.5	-
20.5	-
20.5	0.0%
20.5	10
20.5	-
20.5	2.00
20.5	10
20.5	5

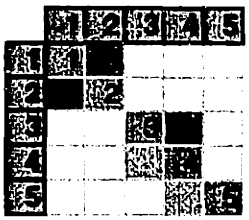
38.5



43.9	1
39.5	1
38.7	82
38.5	17
38.5	-
38.7	0.0%
38.5	6
38.5	-
65.7	2.00
38.5	5
65.2	5

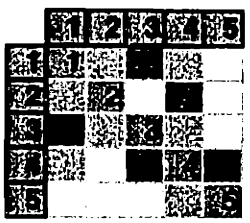
8.5

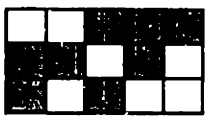
9.8	1
13.4	-
10.1	91
8.5	9
10.1	-
10.2	0.0%
8.6	2
10.2	-
8.5	2.00
10.1	5
8.6	5



7.2	1
7.2	-
7.2	84
7.2	16
7.2	-
7.2	0.0%
7.2	10
7.2	-
7.2	2.00
7.2	20
7.2	5

10.6	1
10.4	-
10.4	79
10.4	21
10.4	-
11.2	0.0%
11.2	7
10.4	-
10.4	2.00
11.2	10
10.4	5

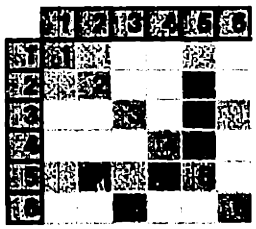


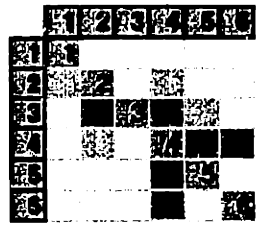


35.2	-
33.4	5
35.7	52
36.5	33
33.4	10
37.6	11.9%
39.4	-
36.5	-
33.4	2.00
33.3	10
32.9	5

43.5	1
43.9	-
43.9	65
39.2	35
49.0	-
39.2	0.0%
39.2	6
39.2	-
63.2	2.00
39.2	5
39.2	5

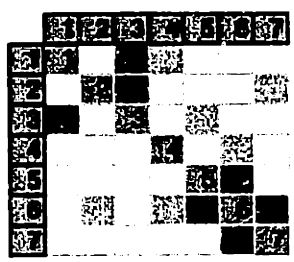
from here down used
times=2 and stable_limit=2

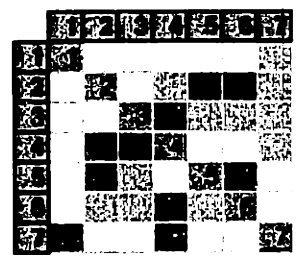




		node
ip obj		130.8
CI		2
CIS		4
opt found=1	74.4	-
# clusters=1	67.6	2
# clusters=2	83.9	47
# clusters=3	78.4	50
# clusters=4	83.9	1
% from opt.	67.6	49.8%
# opt found	67.6	-
# below opt	67.6	10
pow_bid	65.6	2.00
random_accept	83.9	5
random_bid	78.4	3

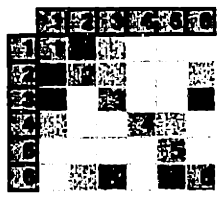
ip obj		40.5
CI		2
CIS		5
opt found=1	59.5	1
# clusters=1	40.5	3
# clusters=2	59.1	67
# clusters=3	59.1	30
# clusters=4	48.6	
% from opt.	59.1	0.0%
# opt found	58.1	1
# below opt	57.8	-
pow_bid	84.7	2.00
random_accept	65.6	5
random_bid	62.6	3





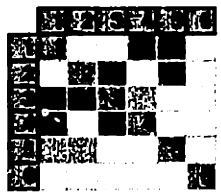
ip obj		75.6
opt found=1	87.3	1
# clusters=1	85.3	5
# clusters=2	96.5	51
# clusters=3	85.3	39
# clusters=4	96.5	5
% from opt.	79.7	0.0%
# opt found	93.2	3
# below opt	75.6	-
pow_bid	75.6	2.00
random_accept	110.0	3
random_bid	75.6	1

ip obj		131.2
opt found=1	163.8	1
# clusters=1	148.1	10
# clusters=2	131.2	48
# clusters=3	153.0	37
# clusters=4	153.0	5
% from opt.	195.1	0.0%
# opt found	173.3	1
# below opt	200.0	-
pow_bid	175.4	2.00
random_accept	155.9	3
random_bid	153.0	1



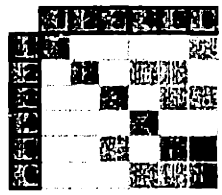
51.3

2
5
69.2 1
67.9 1
87.1 35
62.4 55
60.9 9
87.8 0.0%
76.4 3
51.3 -
51.3 2.00
96.0 5
51.3 3



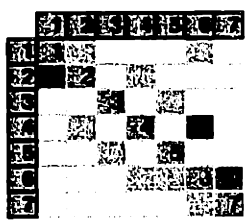
57.6

2
4
68.9 1
59.7 15
59.7 56
81.6 23
69.7 6
57.6 0.0%
59.7 2
59.7 -
88.7 2.00
57.6 5
95.2 3



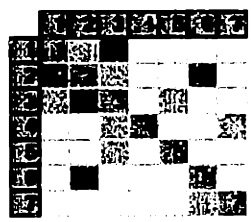
node
27.9

2
4
36.6 -
32.6 24
58.8 60
41.0 16
31.4 6
33.2 4.3%
43.0 -
41.0 2
26.7 2.00
31.4 5
26.7 3



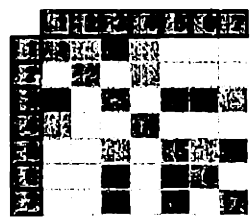
node
67.1

68.2 -
68.6 1
68.6 19
77.6 79
68.6 1
64.1 11.0%
68.4 -
68.6 2
68.4 2.00
69.6 3
59.7 1



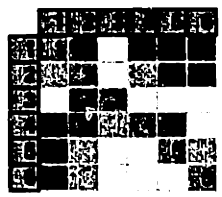
57.3

98.4 -
80.8 -
140.0 28
96.6 57
64.2 15
132.5 12.0%
91.3 -
88.5 -
132.5 2.00
80.8 3
77.2 1



88.8

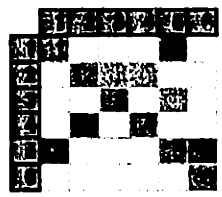
101.3 1
129.2 2
88.8 33
128.2 58
94.1 6
88.8 0.0%
88.8 0
88.8 -
88.8 2.00
88.8 3
129.2 1



node

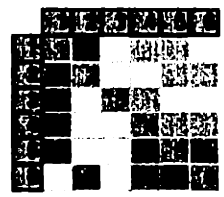
114.7

	2
	4
151.3	-
128.4	24
193.1	80
118.8	16
174.9	-
195.5	3.6%
124.9	-
163.7	-
163.7	2.00
124.9	5
124.9	3



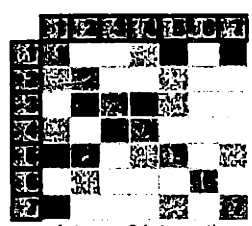
26.1

	2
	5
30.2	1
26.1	6
26.9	83
33.6	17
33.6	-
33.6	0.0%
28.7	3
33.6	-
26.1	2.00
26.1	5
33.6	3



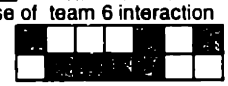
101.3

	2
	4
120.3	-
105.3	6
109.7	48
153.9	44
109.7	-
109.8	4.0%
143.4	-
109.7	-
109.7	2.00
146.1	5
105.3	3

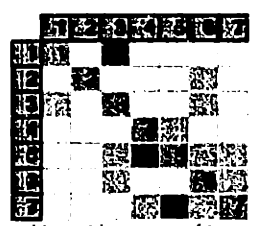


hard to get because of team 6 interaction

89.3

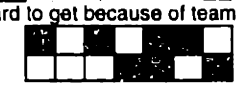


	-
121.6	-
137.1	2
105.4	24
136.7	66
157.3	10
105.4	11.8%
105.4	-
117.0	-
99.9	2.00
114.9	3
137.1	1

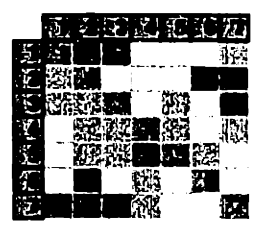


node (hard to get because of team 2 interaction)

55.1

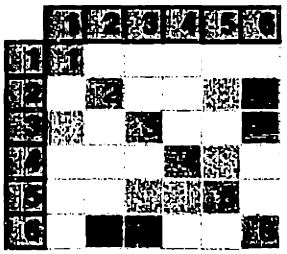


	-
82.4	-
107.6	2
107.6	17
108.6	49
55.4	34
60.1	0.5%
115.0	-
76.2	-
67.2	2.00
71.3	3
55.4	1



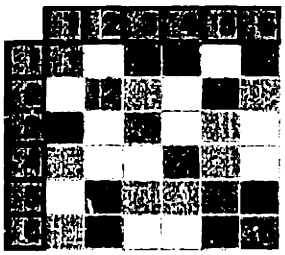
138.4

	1
178.3	1
187.5	2
174.6	30
138.4	66
179.3	2
140.9	0.0%
238.3	1
208.2	-
196.3	2.00
151.6	3
168.2	1



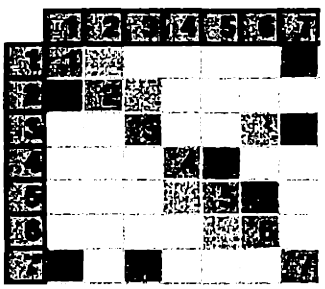
32.9

	2
	4
49.6	1
43.9	
62.5	46
62.5	48
43.9	6
32.9	0.0%
43.4	1
62.5	-
62.5	2.00
46.4	5
35.4	3

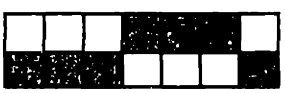


87.7

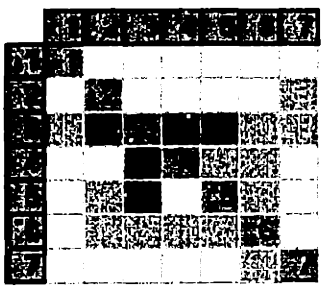
	2
	5
94.3	1
88.7	
87.7	84
89.1	16
89.1	-
137.3	0.0%
88.7	3
97.7	-
87.7	2.00
89.1	5
87.7	3



63.9



	1
77.9	
63.9	-
85.5	51
68.0	44
141.3	5
63.9	0.0%
63.9	5
63.9	-
97.0	2.00
68.0	3
63.9	1



85.4

	1
120.4	
89.6	9
113.7	60
130.9	28
166.6	3
110.7	0.0%
134.5	1
162.4	-
112.0	2.00
98.4	10
85.4	2