Projects With Sequential Iteration: Models and Complexity

Arunava Banerjee
Department of Computer and Information Science and Engineering
University of Florida
CSE Building, Room E301
P.O. Box 116120
Gainesville, FL 32611-6120
(352) 392-1476
arunava@cise.ufl.edu

Janice E. Carrillo
Department of Decision and Information Sciences
Warrington College of Business
University of Florida
355E Stuzin Hall
P.O. Box 117169
Gainesville, FL 32611-7169
(352) 392-5858
janice.carrillo@cba.ufl.edu

Anand Paul
Department of Decision and Information Sciences
Warrington College of Business
University of Florida
356 Stuzin Hall
P.O. Box 117169
Gainesville, FL 32611-7169
(352) 846-1239
paulaa@ufl.edu

ABSTRACT

Many new product development (NPD) projects are inherently complex, making effective management of the tasks, resources, and teams necessary to bring new products to market problematic. Frequently, managers of such NPD projects are overwhelmed by complicating factors such as stochastic task times, ill-defined specifications, complex interrelationships between tasks, and information dependencies. Recently, several researchers have developed an alternative project management tool called the Design Structure Matrix (DSM) that explicitly takes into account the iterative nature of new product development projects. In this paper, we first introduce a mixed integer linear programming formulation for the numerical DSM. Next, we analyze the numerical DSM and establish the complexity of this class of problems. Finally, numerical analysis of the DSM problem and heuristic approaches shows that relatively good solutions can be easily obtained, thereby offering managers efficient alternative solution approach to the original DSM problem.

1. Introduction

Many new product development (NPD) projects are inherently complex, making effective management of the tasks, resources, and teams necessary to bring new products to market problematic. Typically, product development concerns all activities which facilitate the transformation of a market opportunity into a product available for sale, (Krishnan and Ulrich (1997)). Frequently, managers of such NPD projects are overwhelmed by complicating factors such as stochastic task times, ill-defined specifications, complex interrelationships between tasks, and information dependencies.

Traditional project management methodologies including PERT and CPM are useful tools for handling some of the problems associated with NPD projects. For example, given a list of project tasks with corresponding estimates of completion times and precedence relationships, the critical path listing those tasks crucial in determining the overall project completion time can be easily determined. However, these methodologies fail to take into account many of the complicating factors inherent in managing NPD projects. GERT (Graphical Evaluation and Review Technique), a transform based method of describing networks algebraically, overcomes some of the limitations of PERT analysis. In principle, GERT analysis can characterize the distribution of project completion time taking into account activity iteration and probabilistic looping. However, it becomes necessary to resort to simulation for all but simple networks.

In their study of several companies, Adler et al (1996) discuss the failure of these traditional project management methodologies in aiding firms to speed new products to market. Two of the key factors they identify confounding successful project management of new product development processes were ineffective measures of resource utilization, and the iterative nature of new product development processes. To illustrate, they discuss the experiences of a major computer equipment manufacturer that created cross-functional concurrent teams in an attempt to minimize the number of iterations (or re-work cycles) needed to bring new products to market. While such a strategy may increase the communication between team members, thereby decreasing the chances of re-work, they found that the team itself then became a bottleneck resource which actually limited company's development speed. Similar experiences were reported at several companies, including Motorola, Hewlett Packard, General Electric, AT&T, and Ford.

Over the past decade, several researchers have developed an alternative project management tool that explicitly takes into account the iterative nature of new product development projects; this is the Design Structure Matrix (DSM) approach. While DSM methodologies encompass the iterative nature of design projects, exact analysis of the computational difficulty of problems of this nature has remained relatively unexplored. In this paper, we first provide a mixed integer linear programming formulation of the numerical DSM problem and highlight the computational difficulties with the approach. We then analyze the numerical DSM method and formally establish its complexity. Moreover, we explicitly address the situation where the task times are stochastic within the DSM framework. Finally, we address the computational aspect of the problem by solving several problem instances using the CPLEX integer programming software. We also examine the performance of some intuitively appealing heuristics, thereby offering managers efficient alternative solution approaches to the original DSM problem.

The remainder of the paper is organized as follows. In Section 2, we provide an overview of the basic DSM problem and summarize the existing literature on this problem. The basic DSM problem and a related problem are described and analyzed in Section 3. In Section 4, computational issues are described. Numerical analysis of these models and other sample heuristics are also shown in Section 4. Finally, the conclusions offer an overview of the implications of this analysis as well as suggestions concerning future directions for research.

2. Overview of the DSM Problem and Existing Literature

Steward (1981) introduced the notion of a Design Structure Matrix (DSM) which listed the project tasks and showed an explicit relationship between two of the project tasks by placing an 'X' in the matrix. Such a method is similar to PERT/CPM techniques which specify precedence relationships, in that entries in the lower diagonal portion of the DSM matrix represent a precedent task relationship. However, an entry in the upper diagonal portion of the DSM matrix denotes an iterative process whereby the outcome of a particular activity can impact directly on a previously performed task. Smith and Eppinger (1997) further refined the DSM concept to include the stochastic nature of task iteration by adding probabilities to the off-diagonal entries, and deterministic task completion times to the diagonal entries of the matrix. The probabilities reflect the chance of having to repeat a task given the information obtained during the completion of another task. To illustrate, in a typical numerical DSM the task times t_i are listed in the diagonal entries and the probability p_{ij} that a certain task i will have to be iterated again after completion of activity j are listed in the non-diagonal entries. The authors discuss ways to identify the best task ordering utilizing a reward Markov chain approach and note that “no simple closed-form relationship has been found that can determine the minimum length ordering of a 3x3 or larger matrix.”

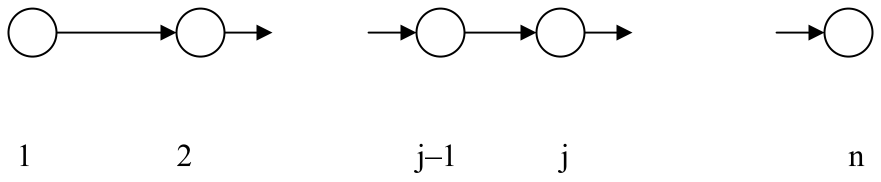
Denker et al (2001) and Eppinger (2001) both offer simple tutorials describing the basic mechanics of DSM’s. For example, focusing on a particular row of the DSM shows all of the information inputs necessary to complete a task, while focusing on a particular column of the DSM shows all of the information outputs that one provides to other tasks. Similarly, managers should first attempt to determine an appropriate task sequence which creates a lower diagonal matrix, thereby minimizing the task dependencies. If this is not possible, then these authors advocate that managers attempt to limit the scope of the iteration by arranging all of the dependencies in the top of the matrix as close to the diagonal as possible.

Other authors offer insights into the application of DSM methodologies in a variety of different ways to many diverse industries. Browning (2001) offers an excellent overview of the literature on DSM. Yassine et al (2001) offer advice on how managers can determine appropriate probability measures p_{ij}. Yassine et al (2000) introduce the notion of restructuring or inserting additional tasks to minimize task iteration. Finally, Ahmadi et al (2001) apply an alternate version of the DSM to the problem of minimizing the number of iterations necessary to complete large scale design projects.

3. Analysis of the DSM Problem

3.1 The DSM Problem Description

Consider a set of n activities to be executed sequentially. Each activity takes a certain time for its completion, and in general the completion times of the activities are random variables. We assume that the mean completion times of all the activities can be estimated a priori. There is a specific type of dependence between activity completion times and activity sequence. Let us explain the dependence structure associated with the DSM matrix with respect to the diagram below.



Activity 1 is completed after a completion time t_1, and activity 2 is begun. After activity 2 is completed, there are two possible outcomes depending on whether or not activity 1 needs to be reworked. There is a probability p_{12} that activity 1 will have to be iterated and a probability $(1-p_{12})$ that we may proceed to the next activity in sequence, activity 3. Suppose an iteration of activity 1 is required. In that case, after the iteration is completed, there are 2 possibilities: activity 2 may have to be iterated (with probability p_{21}) or we may proceed to activity 3 (with probability $1-p_{21}$).

Consider the situation when we reach activity j for the first time. After activity j is completed, there are j possibilities: any one of the (j –1) preceding activities i may have to be repeated with probability p_{ij} (i = 1,2,…, j-1), or else we proceed to activity (j+1). If one of the preceding activities (say activity k) is iterated, there are again j possibilities after activity k is completed; either one of the (j –1) activities in the set {1,2,…,k-1,k+1, …,j} is iterated, or else we proceed to activity (j+1). We define the total time that elapses between the point when we first reach activity j to the point when we first reach activity (j+1) to be the *stage time* of activity j (j = 1,2,…,n-1). The stage time of the *last* activity is the time interval between the point of arrival at the last activity and completion of the project. The stage time of the *first* activity is simply its completion time.

We proceed in this fashion until we successfully traverse the complete chain of n activities. With finite activity completion times and some mild restrictions on the iteration probabilities, the procedure will terminate in a finite amount of time. The problem is to identify the sequence that minimizes expected project completion time (the sum of the expected stage times of all the activities) for a set of activities with given activity completion times and iteration probabilities.

We note that the problem of minimizing the expected completion time of a project with stochastic task times is equivalent to minimizing the expected completion time of the same project with each completion time random variable replaced by its mean. This follows from two facts: first, the completion time of the project is a linear function of the activity completion times; second, expectation is a linear operator. Hence, in the remainder of the paper we shall use the terms "activity completion time" and "expected activity completion time" interchangeably. It is interesting to note that we do not even need the activity time random variables to be independent.

3.2 Integer Programming Formulation of the DSM

Smith and Eppinger (1997) utilize a reward Markov chain approach to determine the expected completion time for a particular sequence of activities when analyzing the DSM problem. Furthermore, these authors state that "no simple closed-form relationship has been found that can determine the minimum length ordering of a 3x3 or larger matrix." We propose two alternate integer programming formulations of the DSM problem which can be solved utilizing standard integer programming software packages. Let the variable $x_{ij}=1$ if task i is assigned to the jth position in the sequence. Also r_{ij} is an intermediate variable which takes on the value of the stage time once the task has been scheduled in the sequence. More specifically, if $x_{ij}=1$, then $r_{ik}=0$ for all $k<j$. For all $k\geq j$, let r_{ik} be the expected stage time for task i if it is assigned to the kth position in the sequence, given a particular sequence of tasks assigned to the previous positions. The stage time variables can also be interpreted as the expected time required to complete task i and all subsequent tasks encountered in stage j, given that one does task i first in stage j. For a DSM matrix of dimension nxn with expected task times in the diagonals and probabilities in the off-diagonals, a quadratic programming formulation of the DSM problem is shown below.

Quadratic Integer Programming Formulation (DSM):

$$\text{Min } \sum_{i=1}^{n}\sum_{j=1}^{n} r_{ij}x_{ij}$$

St:

$$\sum_{i=1}^{n} x_{ij} = 1 \quad \forall j$$

$$\sum_{j=1}^{n} x_{ij} = 1 \quad \forall i$$

$$r_{ij} = \left(\sum_{k=1}^{j} x_{ik}\right)\left(t_i + \sum_{\substack{l=1 \\ l\neq i}}^{n} p_{li}r_{lj}\right) \quad \forall i \forall j$$

x_{ij} binary

We shall show that the quadratic integer problem above can be converted into a *linear mixed integer* problem. First, the linearization can be achieved by noting that every quadratic term in the original problem can be written as a product of a continuous variable and a binary variable. Next, let r be a positive valued continuous variable and let x be a binary variable corresponding to an arbitrary quadratic term 'rx' in the above formulation. We create a new variable w and substitute it for 'rx'. Let M be an upper bound for all the positive valued continuous variables in the quadratic programming problem above. To ensure that w is identical to 'rx', we add the following linear constraints:

$w \geq 0$ (C-1)

$w \leq Mx$ (C-2)

$w \leq r$ (C-3)

$w + M \geq r + Mx$ (C-4)

If x = 0, it follows from (C-1) and (C-2) that w = 0. If x = 1, it follows from (C-3) and (C-4) that w = r. Hence the four constraints together ensure that w is identically equal to 'rx'. To convert the quadratic programming formulation for the DSM into a linear MIP, we replace each quadratic term by a new variable and add constraints of the form (C-1) to (C-4) above. This completes the linearization.

In general, finding a mixed integer linear formulation for a combinatorial optimization problem would give some hope of finding a computationally efficient solution. In this case, unfortunately, the linearization does not pave the way for an efficient solution. The crux of the problem is this: *the LP relaxation of the MIP obtained by the procedure described above will inevitably give a poor lower bound to the optimal integer solution.* This makes it impossible to run an efficient branch and bound procedure for solving the MIP. While we discovered the fact that the big-M method gives poor lower bounds empirically, the logic behind it can be explained with the help of the following simple example.

Example

Consider the following 2 activity DSM:

Activities	**1**	**2**
1	3	0.6
2	0.4	4

The QIP formulation for the problem is as follows:

Minimize $r_{11} y_{11} + r_{21} y_{21} + r_{12} (y_{12} - y_{11}) + r_{22} (y_{22} - y_{21})$

Subject to

$r_{11} = (3 + 0.4\, r_{21})\, y_{11}$

$r_{21} = (4 + 0.6\, r_{11})\, y_{21}$

$r_{12} = (3 + 0.4\, r_{22})\, y_{12}$

$r_{22} = (4 + 0.6\, r_{12})\, y_{22}$

$y_{11} + y_{21} = 1$

$y_{12} = y_{22} = 1$

$y_{11}, y_{21}, y_{12}, y_{22}$ binary.

To see that the above formulation is a special case of the generic QIP previously discussed, note that we made the substitution y_{ij} for $\sum_{k=1}^{j} x_{ik}$ in the original formulation. The objective function takes the above form because $x_{ij} = y_{ij} - y_{i,j-1}$.

Consider what happens when the substitution $ry = w$ is made. (For simplicity, we drop the subscripts on r and y.) Choose $M = 100$ without loss of generality. The linearization requires the following constraints to be added:

$w \leq 100y$

$w \leq r$

$w \geq 0$

$w \geq 100y + r - 100$.

Substituting an arbitrary value of r in the system of inequalities above will give us a clear picture of the situation. By setting $r = 10$, the inequalities reduce to:

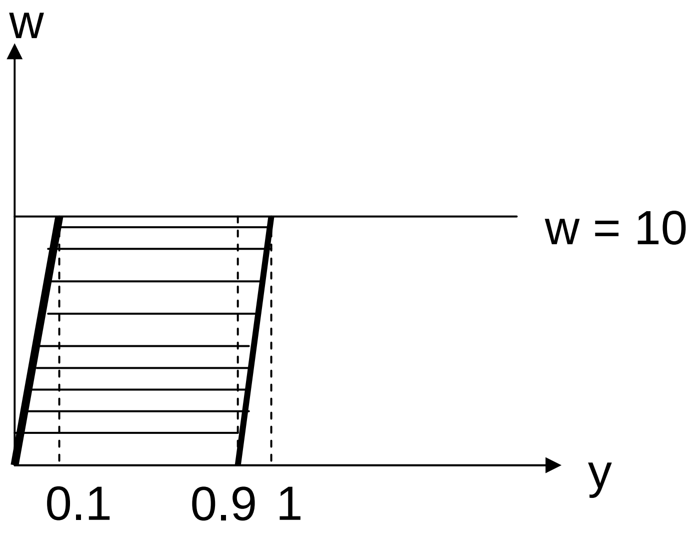
$w \leq 100y$

$w \leq 10$

$w \geq 0$

$w \geq 100y - 90$

The two dimensional slice of the feasible region at $r = 10$ when y is relaxed is represented by the shaded parallelogram in the graph on the next page. As shown in the graph, a larger value of M corresponds to a larger feasible region. Consequently, the constraints for the relaxed problem are not as tight. In fact, given any positive fraction e (however small), there is a value of M such that w lies in the range [0,r] for all y in [e, 1-e]. The point is that the more leeway w is given, the smaller the value returned by the objective function and the poorer the lower bound obtained. On the other hand, trying to control the value of M used in a particular instance by setting M close to r is not an option because we do not have a good upper bound on r in the first place. Ultimately, finding a good upper bound is as hard a problem as the one we seek to solve.



In the relaxed version of the MILP just described, if we substitute each occurrence of $r_{ij}y_{kl}$ with w_{ijkl}, the objective becomes the following:

Minimize $w_{1111} + w_{2121} + w_{1212} – w_{1211} + w_{2222} – w_{2221}$.

Since $y_{12} = y_{22} = 1$, the constraints on w_{1212} and w_{2222} are tight; hence $w_{1212} = r_{12}$ and $w_{2222} = r_{22}$. However, note that if $y_{11} = 0.1$ and $y_{21} = 0.9$, then on the strength of the above observations, w_{1211} can be set as high as r_{12} and w_{2221} can be set as high as r_{22}. In effect, the part of the objective which includes $w_{1212} - w_{1211} + w_{2222} – w_{2221}$ can be driven to zero. So the optimal value of the objective function of the relaxed MILP is close to a relaxation of the expected stage time after the first job is scheduled. Succeeding stage times contribute very little to the objective. This is clearly a very poor lower bound on the objective function of the original QIP.

3.3 Complexity of the DSM problem

The following theorem addresses the complexity of the DSM problem.

Theorem 1: *The DSM problem is NP-hard.*

Proof: First, we note that the problem can be described efficiently by means of a directed graph. Given a set of n jobs with expected job completion times t_i, we construct a directed graph as follows. We associate job i with node i (i = 1 to n). Each node is therefore associated with a positive number t_i, the completion time of the job corresponding to the node. If p_{ij} is strictly greater than zero, we draw an arc from node j to node i with the number p_{ij} on the arc. We assume without loss of generality that this directed graph is strongly connected. Otherwise it can be broken into its strongly connected components and our analysis repeated for each such component. (Note that the nodes in 2 strongly connected components cannot lie on a cycle.)

Given a strongly connected directed graph G, we first formulate a <u>modified version</u> of the DSM problem that simplifies the structure of the iterations and fixes the task times and iteration probabilities via a simple rule. The task time T for all the nodes is defined to be the maximum out-degree of G and $p_{ij} = \frac{1}{T}$ is the probability attached to the edge from node j to node i , if such an edge exists, and $p_{ij} = 0$ otherwise. Consider the situation when we reach job j for the first time (j > 2). After job j is completed, there are j possibilities: any one of the (j –1) preceding jobs i may have to be repeated with probability p_{ij} (i = 1,2,…, j-1), or else we proceed directly to job (j+1). If one of the preceding jobs (say job k) is iterated, *we complete that job (consuming time t_k) and then proceed to job (j+1) directly after.* (Note that this iteration rule is in stark contrast to that in the DSM model, where the iteration of job k may be followed by further iterations of other jobs before returning to job (j+1) in the main sequence). We note the following fact about the expected completion time in the modified DSM model. The claim follows immediately from the definition of the task times and iteration probabilities at every node.

Fact 1: *The expected completion time of every sequence, according to the rules described above for the modified DSM model, is an integer.*

The "minimum feedback arc set" problem on a digraph, which is known to be NP-hard (Karp, 1972), is trivially reducible to the decision version of the modified DSM problem; hence the decision version of the modified DSM problem is NP-hard. Now we shall reduce an arbitrary instance of the modified DSM problem to an instance of the DSM problem, and show a one-to-one correspondence between the optimal sequences for the two problem instances.

Suppose we are given a graph G for the modified DSM problem. We construct a new directed graph H for the DSM problem as follows. For each node i in G, we create 2 nodes 'out_i" and "in_i" in H. For each edge (i,j) from node i to node j in G, we assign edge (out_i,in_j) from node out_i to node in_j in H and associate it with probability $p_{ji} = \frac{1}{T}$. We also create edges from "in_i" to "out_i" for every i and associate each such edge with probability Q (to be defined later). We assign task time $\frac{T}{Q}$ to all nodes in_i and task time 0 to all nodes out_i.

Given any sequence (not necessarily an optimal sequence) in G for the modified DSM problem, we define the corresponding sequence in H for the DSM problem as follows. Let the sequence in G be $\sigma(1)\sigma(2)...\sigma(n)$ where $\sigma(i)$ is the element of {1,2,…,n} to which the permutation σ maps i. Consider an arbitrary permutation $\tilde{\sigma}$.

Definition: For any sequence $\sigma(1)\sigma(2)...\sigma(n)$ in G, we define the sequence $out_{\tilde{\sigma}(1)}out_{\tilde{\sigma}(2)}...out_{\tilde{\sigma}(n)}in_{\sigma(1)}in_{\sigma(2)}...in_{\sigma(n)}$ to be a *corresponding sequence* in H.

So a corresponding sequence in H is the sequence of "out" nodes in any order followed by the sequence of "in" nodes in the same order as the sequence in G.

Fix a sequence σ in G. We define the *rework time* for node i to be the stage time for node i minus the task time t_i. The rework time for a sequence is the sum of the rework times over all the nodes of the sequence.

Lemma 1

The rework time for a corresponding sequence in H is bounded above by the rework time for σ in G plus a positive fraction and bounded below by the rework time for σ.

Lemma 2

An optimal sequence in H consists of the "out" nodes in some order followed by the "in" nodes in some order.

Lemma 3

An optimal sequence in H is a corresponding sequence that corresponds to an optimal sequence in G.

It follows from Lemma 1 and Fact 1 that if the optimal sequence in H consists of a batch of "out" nodes followed by a batch of "in" nodes, then the optimal sequence in G can be immediately inferred from the optimal sequence in H (Lemma 3 spells out the inference). Lemma 2 guarantees that the optimal sequence in H must indeed consist of a batch of "out" nodes followed by a batch of "in" nodes. Hence, the lemmas, together with Fact 1, establish that the DSM problem is NP-hard. The proofs for Lemma 1 and Lemma 2 are included in the Appendix. Lemma 3 is an immediate consequence.

3.5 A Simplified Model of Iteration

In view of the complexity of the structure of iterations in the DSM model, we introduce a simplified model of iterations in which we limit the number of activities that can be iterated during each stage. Consider the point at which we reach activity j for the first time. After the activity is completed, there are j possibilities of iteration. With probability $p_{kj,}$ activities j and k have to be resolved together, a procedure that takes time $(k \oplus j)$ (k = 1,2,…,j-1), where we define $(k \oplus j)$ to be the expected rework time in a 2-activity DSM with activities k and j when k is sequenced before j. Furthermore, we assume that once the j and k pair of activities is resolved, further iterations cannot occur with alternate activities, and the stage is completed. The remaining possibility is that no iteration is required at this stage; in this case, we proceed to activity (j+1) directly after activity j is completed. We call this model the Simple DSM Model. It follows from our proof of Theorem 1 that the Simple DSM Model is NP-hard. But at this point the plot takes an interesting turn: if the probability matrix is symmetric, it turns out that the Simple DSM Model is polynomially solvable. This follows from Theorem 2 below.

Theorem 2: *If the iteration probabilities are symmetric ($p_{jk} = p_{kj}$), then sequencing the activities in SPT order minimizes expected project completion time in the Simple DSM Model.*

Proof: We proceed via an adjacent pairwise interchange argument. Let T_i and t_i denote the stage time and completion time, respectively, of activity i. Write $(j \oplus i)$ for the time required to resolve an iteration between activities j and i initiated at i. The expected time at stage i is

$$T_i = t_i + \sum_{j=1}^{i-1} (j \oplus i) p_{ji} \ .$$

Now compare the sum of the stage times of all n activities resulting from the sequence 1,2,…,i-1,i,i+1,…,n with the sum of the stage times resulting from the sequence 1,2,…, i-1,i+1,i…,n. After cancellation, we are left with the terms p_{ij} $(i \oplus j)$ and p_{ji} $(j \oplus i)$. Since $(i \oplus j) = \frac{t_j + t_i p_{ji}}{1 - p_{ij} p_{ji}}$ (see Smith and Eppinger (1997), page 1111), if $p_{ij} = p_{ji}$ for all i,j, then p_{ij} $(i \oplus j) <$ p_{ji} $(j \oplus i)$ if and only if $t_i < t_j$. Hence, sequencing in ascending order of activity completion times is optimal. **Q.E.D.**

4 Computational Issues

4.1 Heuristics

Next, we propose other *heuristic methods* that greatly reduce the computational burden associated with solving the DSM problem. The first heuristic is to ignore the probability of task iteration and simply schedule the tasks according to the shortest expected task (processing) time (SEPT). The second heuristic creates a ratio for each activity based on a combination of the activity's time and probability of repeating other tasks after this activity. This heuristic method is akin to the SEPT method, but directly takes into account the impact of iteration on completion times. The Shortest Expected Processing Time Ratio (SEPTR) heuristic is summarized in the following steps.

1. Initialize k such that k = n.
2. Calculate a ratio h_i for each activity i (i = 1 to k) as follows: $h_i = \dfrac{t_i}{\sum_{\substack{j=1 \\ j \neq i}}^{n} p_{ij} / (1 - p_{ij})}$.

3. Schedule the activity associated with the largest ratio in the kth position in the sequence, and remove it from the activity list. Decrement k such that k=k-1. Repeat steps 2 and 3 until there are no more activities to schedule (i.e. when k=0).

4.2 Numerical Results

A series of numerical experiments were developed to explore the efficacy of the heuristic introduced relative to traditional branch-and-bound methodologies. First, we generated a series of DSM's with varying expected task time and column probability parameters. The dimension of the DSM's that we tested were n=6 and n=9 total tasks. The expected processing times for each task were randomly drawn from a uniform distribution with a mean of μ and a spread of L, such that $t_i \in [\mu - L/2, \mu + L/2]$. Two levels for each of these parameters were investigated, with μ= 4 or 10, and L=2 or 8.

To generate the iteration probabilities in the matrix, we adjusted both the density of the matrix and the total probability of an iteration for each column. The density variable α effectively controls what percentage of the tasks within a column will have a positive probability of iteration. Three levels of density were investigated with α=33%, 67% or 100%. To illustrate, when n=6 and α=33%, then 2 tasks in each column have a positive probability of iteration. A uniformly distributed random number between zero and 1 was generated for each off-diagonal element of the matrix to determine which tasks within a given column would have a positive probability of iteration. When the density variable α was set to 100%, this corresponds to the situation where there is a positive probability that all prior tasks would need to be iterated after the completion of a task in the series. For this situation, we also assumed that the probabilities of iteration were equivalent for all of the tasks in a column. Consequently, we can investigate the scenario where the probability of task iteration is equal over all previously completed tasks. This situation may occur when managers are uncertain about the actual iteration probabilities, but estimate that there is roughly an equal chance of iterating among all previously completed tasks. It may also occur when there is very little *a priori* information about iteration probabilities.

Finally, the parameter β controls the magnitude of the sum of the column probabilities. Recall that a prerequisite for feasibility of the numerical DSM problem is that the sum of the column probabilities cannot exceed 1. The actual column probability used for each DSM was randomly generated from a uniform distribution over the interval [0, β]. We investigated two levels of this parameter with β=.5 or .9. Then, uniform [0,1] randomly generated numbers were generated for each positive non-diagonal element to determine the fraction of the total column probability that each element should receive.

A total of 480 DSM's were generated, reflecting 10 trials of each of the 48 different parameter combinations. The DSM matrices were generated using C++ on a dual-processor Intel Xeon 3.4 GHz Dell workstation with 2G of memory. The CPLEX software was utilized to solve the linear MIP formulation of each DSM problem instance and identify the optimal solution, while Matlab software was used to generate the heuristic solutions. The computational results for each set of trials is shown in Tables 1 and 2 in the Appendix. The impact of the individual parameters for each size matrix is shown in Table 3.

Overall, the SEPTR heuristic performed quite well relative to the optimal solution. The average SEPTR gap expressed in the percentage of the average completion time for the optimal solution was 1%, with a maximum of 29%. The DSM which yielded the gap of 29% appears to be somewhat of an outlier, as the next highest SEPTR gap was 13%. Upon further investigation, the outlier was a 6x6 DSM where the sum of the column probabilities for 4 out of 6 of the columns was greater than or equal to .78. The SEPTR heuristic also clearly outperforms the SEPT heuristic. The average SEPT gap was 8%, with a maximum gap of 48%. Finally, while the 6x6 DSMs could all be solved in less than a minute, the mean CPU time for the 9x9 matrices was 37 minutes and the maximum CPU time was 103 minutes. Because the average solution time of the SEPTR heuristic was .06 seconds, the SEPTR heuristic clearly gives a high quality solution in a minimal amount of time.

The impact of various task time and iteration parameters on the solution quality is shown in Table 3. In general, it appears that the average task time parameters do not have much of an influence on the heuristic performance. In contrast, the parameters controlling the iteration probabilities appear to influence the heuristic quality. One surprising result is that the heuristic gives better results when there are more tasks with a positive probability of iteration. One explanation for this result is that, because the sum of the tasks must be less than 1, increasing the number of tasks with a positive probability of iteration necessarily distributes the total column probability over a larger number of tasks. Consequently, the ratio calculated for the SEPTR heuristic is averaged over a larger number of tasks.

To facilitate the experimental design and to be able to report our computational findings within a reasonable time frame, we tested only 6x6 and 9x9 matrices. Indeed, the average CPU Time to solve the 9x9 matrices using CPLEX was 37 minutes, with the maximum CPU Time of 103 minutes. The typical size of a DSM, however, is much larger than those shown here. For example, Eppinger (2001) describes a DSM used by Intel to plan for its development process for new semiconductors which contains 60 tasks. Similarly, Yassine et al (2001) discusses the design process for a hood of a new car at Ford which contains 43 tasks. While further analysis is warranted to address these larger DSM's, we anticipate that the SEPTR heuristic will perform reasonably well for larger projects.

5. Conclusions

The possibility of iteration frequently occurs in several types of project management networks. The Design Structure Matrix (DSM) was developed to develop insights into managing iteration in new product development projects, specifically in situations where new information or task incompatibility necessitates the repetition of previously performed tasks. Product development projects of this type include those where considerable technical or market uncertainty exists (such as R& D projects). To illustrate, DSM techniques have been used for applications in the semiconductor, automobile, and aerospace industries. However, task iteration is not unique only to product development projects. Large scale construction projects also suffer from the consequences of task iteration. Within the job-shop scheduling domain, task iteration in the form of rework is a realistic consideration. Utilization of such DSM techniques is appropriate in these alternate environments, to the extent that the probabilities of rework can be estimated in planning an appropriate schedule.

In this paper, we establish the complexity of the DSM problem with stochastic task times to be NP-hard. In contrast to previous authors, we also explicitly consider task time uncertainty in our formulation. Because of the complexity of the original DSM model, we develop computational and heuristic methods for solving the DSM problem. First, we include a mixed integer linear programming formulation of the DSM problem, although this approach leads to limited computational success. Second, we develop and analyse different heuristic methods. Numerical analysis of 480 test cases shows that one of these, called the *Shortest Expected Processing Time Ratio* (SEPTR) heuristic method, is quite promising, in that it yields near optimal solutions with minimal computational effort. Moreover, these techniques can be applied to large scale networks that were previously difficult to solve.

Finally, we suggest some interesting avenues for future research. First, it would be useful to investigate the impact of sequencing on the variance of project completion time. For instance, we could explore a PERT-like approach, whereby we first identify the sequence that minimizes expected completion time, and then compute the variance of the project completion time associated with this sequence. The problem of finding the sequence that minimizes the metric 'mean + a fixed multiple of the standard deviation of the project completion time' may be managerially interesting. A second substantial problem for future research is that of developing good lower and upper bounds for the DSM problem. The SEPT rule may be a step in the direction of a good upper bound. For finding non-trivial lower bounds, one approach would be to curtail the number of iterations allowed at every stage and solve the modified problem to optimality. We believe that there are substantial and interesting challenges involved in pursuing these suggestions.

Acknowledgements:

The authors would like to thank Michelle Hanna for her assistance with the computational experiments.

6. References

Adler, P.S., Mandelbaum A., Nguyen V., Schwerer, E. (1996) Getting the Most Out of Your Product Development Process. *Harv. Bus. Rev.,* 74, 134-147.

Ahmadi, R., Roemer, T.A., Wang, R.H. (2001) Structuring Product Development Processes. *Eur. Jl. of Oper. Res.,* 130, 539-558.

Balas, E., Saltzman, M. (1991) An Algorithm for the Three-Index Assignment Problem. *Oper. Res.*, 39, Jan. - Feb, 150-161.

Browning, T.R. (2001) Applying the Design Structure Matrix to System Decomposition and Integration Problems: A Review and New Directions. *IEEE Trans. on Eng. Mgt.*, 48, 292-306.

Denker, S., Steward, D.V., Browning, T.R. (2001) Planning Concurrency and Managing Iteration in Projects. *Proj. Mgt. Jl.*, 32, 31-38.

Eppinger, S.D. (2001) Innovation at the Speed of Information. *Har. Bus. Rev.*, Jan., 3-11.

Karp, R.M. (1972) Reducibility Among Combinatorial Problems. *Complexity of Computer Computations, Proc. Sympos.* IBM Thomas J. Watson Res. Center, Yorktown Heights, N.Y. New York: Plenum, p.85-103.

Krishnan, V., K. Ulrich (1997) Product Development Deisions: A Review of the Literature. Mgt. Sci., 47, 1-21.

Smith, R.P., Eppinger, S.D. (1997) A Predictive Model of Sequential Iteration in Engineering Design. *Mgt. Sci.*, 43, 1104-1120.

Steward, D.V. (1981) The Design Structure System: A Method for Managing the Design of Complex Systems. *IEEE Trans. on Eng. Mgt.*, 28, 71-74.

Yassine, A.A., Whitney, D.E., Lavine, J., Zambito, T. (2000) Do-it-Right-First-Time (DRFT) Approach to DSM Restructuring, *Proceedings of the DETC 00: ASME 2000 Int. Design Engineering Technical Conferences,* Baltimore MD.

Yassine, A.A., Whitney, D.E., Zambito, T. (2001) Assessment of Rework Probabilities for Simulating Product Development Processes Using the Design Structure Matrix (DSM), *Proceedings of the DETC 01: ASME 2001 International Design Engineering Technical Conferences,* Pittsburgh PA.

Appendix 1: Proofs from Lemma 1 and 2

Proof of Lemma 1

Consider any sequence $\sigma(1)\sigma(2)...\sigma(n)$ in G and a corresponding sequence in H. Without loss of generality, we re-label the nodes to obtain the sequence 1, 2, 3,…, n in G (and a corresponding sequence in H). Now consider the point at which nodes 1 to m-1 have already been sequenced and node in_m is to be sequenced next in H. Let r_m denote the stage time for in_m and s_m denote the stage time for out_m. We have (noting that $p_{ii} = 0$ for all i)

$$r_m = \frac{T}{Q} + Qs_m \qquad (1)$$

$$s_m = \sum_{j=1}^{m} p_{jm} r_j \tag{2}$$

Solving them simultaneously, we get

$$r_m = \frac{T}{Q} + Q\sum_{j=1}^{m} p_{jm} r_j \,. \tag{3}$$

Let $r_j = \frac{T}{Q} + \Delta_j$ (j = 1 to m). Then

$$r_m = \frac{T}{Q} + Q\sum_{j=1}^{m} p_{jm} (\frac{T}{Q} + \Delta_j) = \frac{T}{Q} + \sum_{j=1}^{m} p_{jm} T + Q\sum_{j=1}^{m} p_{jm} \Delta_j \,. \tag{4}$$

Since the rework time in G at node m is $\sum_{j=1}^{m} p_{jm} T$, the rework time at node out$_m$ is, from (4) above, greater than the rework time in G at node m by $Q\sum_{j=1}^{m} p_{jm} \Delta_j$. We shall bound $Q\sum_{j=1}^{m} p_{jm} \Delta_j$. We need to use the fact that $p_{ij} > 0$ and $\Delta_i > 0$ throughout the analysis that follows. Note first that $Q\sum_{j=1}^{m} p_{jm} \Delta_j \le Q\sum_{j=1}^{m} p_{jm} \sum_{j=1}^{m} \Delta_j \le Q\sum_{j=1}^{m} \Delta_j$ (the second inequality follows since $\sum_{j=1}^{m} p_{jm} \le 1$). So we need to bound $Q\sum_{j=1}^{m} \Delta_j$.

In a similar fashion to equation (3) for the stage time of in$_m$, the following equations hold for iterations at nodes in$_1$, …, in$_{m-1}$ at that stage:

$$r_1 = \frac{T}{Q} + Q\sum_{j=1}^{m} p_{j1} r_j$$

$$r_2 = \frac{T}{Q} + Q\sum_{j=1}^{m} p_{j2} r_j$$

.

.

.

$$r_{m-1} = \frac{T}{Q} + Q\sum_{j=1}^{m} p_{j,m-1} r_j$$

Substituting $\frac{T}{Q} + \Delta_j$ for r_j (j= 1 to m) inside the summation sign in each of the preceding equations, as well as in equation (3), and simplifying, we get

$$\Delta_1 = T\sum_{j=1}^{m} p_{j1} + Q\sum_{j=1}^{m} p_{j1}\Delta_j$$

$$\Delta_2 = T\sum_{j=1}^{m} p_{j2} + Q\sum_{j=1}^{m} p_{j2}\Delta_j$$

.

.

$$\Delta_m = T\sum_{j=1}^{m} p_{jm} + Q\sum_{j=1}^{m} p_{jm}\Delta_j$$

Now noting that $\sum_{j=1}^{m} p_{ji} \le 1$ (i = 1 to m) and $\sum_{j=1}^{m} p_{ji}\Delta_j \le \sum_{j=1}^{m} p_{ji} \sum_{j=1}^{m} \Delta_j \le \sum_{j=1}^{m} \Delta_j$ we get

$$\Delta_1 \le T + Q\sum_{j=1}^{m} \Delta_j$$

$$\Delta_2 \le T + Q\sum_{j=1}^{m} \Delta_j$$

.

.

.

$$\Delta_m \le T + Q\sum_{j=1}^{m} \Delta_j$$

Adding up, we get $\sum_{j=1}^{m} \Delta_j \le mT + Qm\sum_{j=1}^{m}\Delta_j$. This simplifies to

$\sum_{j=1}^{m}\Delta_j \le \frac{mT}{1-Qm}$.

Hence $Q\sum_{j=1}^{m}\Delta_j \le \frac{QmT}{1-Qm} \le \frac{QnT}{1-Qn}$. So $\frac{QnT}{1-Qn}$ is an upper bound for the difference in expected rework times in G and H when any single activity is sequenced. Hence the difference in expected rework times when all n activities have been sequenced is bounded above by $\frac{Qn^2T}{1-Qn}$. Choose Q such that $\frac{Qn^2T}{1-Qn} < 1$. We have now bounded the expected rework time of a corresponding sequence in H to within 1 time unit of the expected rework time of the sequence in G. Q.E.D.

Proof of Lemma 2

We shall prove that the optimal sequence in H consists of the "out" nodes in some order followed by the "in" nodes in some order. Suppose this is not the case. Then the optimal sequence in H is a mixed sequence of "out" nodes and "in" nodes. We shall show that such a sequence can be reordered to form a corresponding sequence without adding to the total expected completion time.

Claim1: If out$_i$ is scheduled before in$_i$ for any i, out$_i$ can be moved to the front of the sequence without increasing the completion time.

Proof of Claim 1: Note that moving out$_i$ ahead of its current predecessors does not change the stage times of any of the current predecessors because the only incoming edge into out$_i$ is from in$_i$ and in$_i$ is sequenced after out$_i$. The stage time of in$_i$ is unchanged because it sees the same set of predecessor jobs whether out$_i$ is moved to the first position in the sequence or not. The stage time of out$_i$ can only decrease when it is moved ahead.

Claim 2: If out$_i$ is sequenced after in$_i$ for any i, in$_i$ can be sequenced to be the immediate predecessor of out$_i$ without increasing the completion time.

Proof of Claim 2: The stage times of in$_i$ and out$_i$ do not change after the move; the stage times of the other nodes can only decrease.

It follows from the preceding claims that any optimal schedule can be preprocessed in accordance with the moves described in Claim 1 and Claim 2 without adding to the expected completion time, and the resulting sequence can be partitioned into two blocks: an intial block $out_{\sigma(1)} out_{\sigma(2)} ... out_{\sigma(m)}$ followed by a second block. The second block contains the nodes $in_{\sigma(1)}, in_{\sigma(2)}, ..., in_{\sigma(m)}$ in that order *interspersed with the following pairs of nodes*: $in_{\sigma(m+1)} out_{\sigma(m+1)}, in_{\sigma(m+2)} out_{\sigma(m+2)}, ..., in_{\sigma(n)} out_{\sigma(n)}$. Now consider any consecutive pair $in_{\sigma(k)} out_{\sigma(k)}$ in the preprocessed sequence. When $out_{\sigma(k)}$ is sequenced, it is not possible that any in_j is sequenced before it if $p_{j,\sigma(k)} \neq 0$. This follows because the expected rework time for $out_{\sigma(k)}$ can then be made arbitrarily large by making Q arbitrarily small (the expected rework time is greater than $p_{j,\sigma(k)}\frac{T}{Q}$). So it must be the case that $p_{j,\sigma(k)} = 0$. It follows that interchanging $in_{\sigma(k)}$ and $out_{\sigma(k)}$ in the sequence does not change the stage times of either of the interchanged nodes; consequently, the interchange does not change the total expected completion time of the sequence. Now invoking Claim 1, we infer that we may move $out_{\sigma(k)}$ to the head of the sequence. We repeat this procedure until we finish with a sequence of the corresponding type; this sequence is optimal since we started with an optimal sequence that was, by hypothesis, not a corresponding sequence and permuted it into a corresponding sequence without adding to the expected completion time.   Q.E.D.

Appendix 2: Computational Results

Table 1: Computational Results for 6x6 DSM

Parameter Values					CPLEX MIP Solver				Matlab			
					Optimal Values		Optimal Soln Time (min)		SEPT Gap		SEPTR Gap	
6x6 Experiments (10 Trials Each)	Density	Iteration Prob	Mean	Spread	Avg	Max	Avg	Max	Avg	Max	Avg	Max
	0.33	0.5	4	2	24.846	26.285	0.026	0.033	7.33%	14.62%	0.76%	1.91%
				8	24.911	29.882	0.026	0.032	5.78%	16.60%	0.78%	2.94%
			10	2	62.940	66.950	0.030	0.040	10.14%	18.82%	1.12%	4.38%
				8	62.040	69.373	0.034	0.042	6.29%	9.66%	0.79%	1.77%
		0.9	4	2	27.614	31.944	0.035	0.047	21.79%	45.46%	1.28%	7.18%
				8	29.358	48.784	0.035	0.044	14.30%	35.81%	6.09%	29.88%
			10	2	67.667	73.389	0.033	0.046	15.39%	25.91%	1.35%	4.85%
				8	66.400	76.186	0.030	0.045	15.79%	24.47%	1.73%	8.04%
	0.67	0.5	4	2	26.789	29.767	0.046	0.063	5.43%	10.19%	0.92%	3.77%
				8	25.012	34.737	0.033	0.046	2.56%	5.25%	0.42%	2.03%
			10	2	65.388	68.595	0.042	0.051	7.18%	12.96%	0.63%	3.01%
				8	62.260	69.217	0.041	0.059	4.51%	6.80%	0.62%	1.66%
		0.9	4	2	26.963	35.287	0.045	0.059	6.24%	12.39%	0.79%	2.12%
				8	26.763	35.310	0.043	0.073	6.58%	13.73%	0.88%	3.76%
			10	2	70.701	80.738	0.053	0.071	11.32%	23.56%	1.61%	7.84%
				8	73.181	95.588	0.054	0.077	11.01%	27.60%	1.28%	5.17%
	1.00	0.5	4	2	26.313	28.056	0.049	0.072	3.48%	6.67%	0.00%	0.00%
				8	25.119	32.453	0.045	0.057	2.08%	5.24%	0.00%	0.00%
			10	2	66.568	74.084	0.053	0.081	4.19%	6.42%	0.00%	0.00%
				8	67.805	76.161	0.057	0.089	4.01%	6.92%	0.00%	0.00%
		0.9	4	2	27.021	31.566	0.049	0.069	7.73%	14.60%	0.00%	0.03%
				8	25.427	32.474	0.055	0.070	2.96%	6.21%	0.00%	0.02%
			10	2	74.392	97.669	0.063	0.082	9.05%	16.11%	0.00%	0.00%
				8	74.605	89.096	0.065	0.095	7.43%	11.07%	0.01%	0.07%

Table 2: Computational Results for 9x9 DSM

Parameter Values					CPLEX MIP Solver				Matlab			
					Optimal Values		Optimal Soln Time (min)		SEPT Gap		SEPTR Gap	
	Density	Iteration Prob	Mean	Spread	Avg	Max	Avg	Max	Avg	Max	Avg	Max
9x9 Experiments (10 Trials Each)	0.33	0.5	4	2	38.510	41.214	19.512	26.773	9.25%	16.59%	1.85%	4.89%
				8	44.646	57.574	20.070	27.091	4.79%	8.96%	2.20%	3.95%
			10	2	94.923	100.088	17.651	25.152	9.30%	14.76%	1.71%	6.00%
				8	94.338	108.067	17.282	24.476	6.30%	11.70%	1.14%	2.20%
		0.9	4	2	42.231	48.235	21.339	26.830	15.30%	25.16%	3.29%	10.35%
				8	44.893	57.300	18.819	25.104	11.72%	16.76%	3.49%	6.77%
			10	2	104.280	115.925	18.859	32.508	15.31%	22.07%	5.71%	13.13%
				8	101.005	111.269	17.560	24.546	14.26%	27.29%	3.41%	8.02%
	0.67	0.5	4	2	38.646	41.528	38.354	57.617	5.75%	12.25%	0.83%	1.55%
				8	40.425	51.473	32.328	51.068	3.80%	7.83%	1.11%	3.14%
			10	2	95.752	102.058	35.027	56.317	5.26%	7.88%	0.78%	3.82%
				8	100.163	112.424	34.793	61.767	5.93%	9.42%	1.14%	4.01%
		0.9	4	2	42.819	47.783	39.479	57.628	12.33%	19.45%	1.97%	4.08%
				8	42.275	56.475	38.373	57.272	7.43%	15.04%	2.95%	6.63%
			10	2	107.112	114.783	30.981	45.901	14.01%	19.07%	1.94%	4.60%
				8	109.220	129.331	38.412	75.980	10.30%	17.84%	1.49%	2.75%
	1.00	0.5	4	2	38.987	41.595	51.643	73.431	4.01%	6.26%	0.00%	0.00%
				8	43.990	53.520	48.797	80.320	2.03%	4.13%	0.00%	0.00%
			10	2	99.797	106.633	61.014	90.205	3.83%	5.89%	0.00%	0.00%
				8	101.317	109.801	63.007	103.499	3.88%	6.52%	0.00%	0.00%
		0.9	4	2	45.477	53.494	59.533	86.580	8.36%	15.26%	0.00%	0.00%
				8	36.387	56.566	44.776	67.104	4.05%	6.70%	0.00%	0.03%
			10	2	113.446	126.626	63.469	91.811	7.80%	11.14%	0.00%	0.00%
				8	106.883	118.040	60.362	88.985	5.30%	11.50%	0.00%	0.00%

Table 3: Impact of Numerical Parameters on Computational Results

Parameter Values		CPLEX MIP Solver				Matlab			
		Optimal Values		Optimal Soln Time (min)		SEPT Gap		SEPTR Gap	
6x6 Experiments (10 Trials Each)	Density	Avg	Max	Avg	Max	Avg	Max	Avg	Max
	0.33	45.722	76.186	0.031	0.047	12.10%	45.46%	1.74%	29.88%
	0.67	47.132	95.588	0.044	0.077	6.85%	27.60%	0.89%	7.84%
	1.00	48.406	97.669	0.055	0.095	5.12%	16.11%	0.00%	0.07%
	Iteration Prob								
	0.5	44.999	76.161	0.040	0.089	5.25%	18.82%	0.50%	4.38%
	0.9	49.175	97.669	0.047	0.095	10.80%	45.46%	1.25%	29.88%
	Mean								
	4	26.345	48.784	0.041	0.073	7.19%	45.46%	0.99%	29.88%
	10.00	67.829	97.669	0.046	0.095	8.86%	27.60%	0.76%	8.04%
	Spread								
	2	47.267	97.669	0.044	0.082	9.11%	45.46%	0.71%	7.84%
	8	46.907	95.588	0.043	0.095	6.94%	35.81%	1.05%	29.88%
9x9 Experiments (10 Trials Each)	Density								
	0.33	70.603	115.925	18.887	32.508	10.78%	27.29%	2.85%	13.13%
	0.67	72.051	129.331	35.968	75.980	8.10%	19.45%	1.53%	6.63%
	1.00	73.285	126.626	56.575	103.499	4.91%	15.26%	0.00%	0.03%
	Iteration Prob								
	0.5	69.291	112.424	36.623	103.499	5.35%	16.59%	0.90%	6.00%
	0.9	74.669	129.331	37.663	91.811	10.51%	27.29%	2.02%	13.13%
	Mean								
	4	41.607	57.574	36.085	86.580	7.40%	25.16%	1.47%	10.35%
	10.00	102.353	129.331	38.201	103.499	8.46%	27.29%	1.44%	13.13%
	Spread								
	2	71.832	126.626	38.072	91.811	9.21%	25.16%	1.51%	13.13%
	8	72.128	129.331	36.215	103.499	6.65%	27.29%	1.41%	8.02%