INTEGER PROGRAMMING FOR DISCRETE OPTIMIZATION OF THE AGILE SUPPLY CHAIN CONFIGURATION PROBLEM

THESIS

Presented to the Graduate Council of
Texas State University-San Marcos
in Partial Fulfillment
of the Requirements

for the Degree

Master of SCIENCE

by

Hayden D. Beauchamp, B.S.

San Marcos, Texas
August 2013

INTEGER PROGRAMMING FOR DISCRETE OPTIMIZATION

OF THE AGILE SUPPLY CHAIN CONFIGURATION

PROBLEM

Committee Members Approved:

_____
Farhad Ameri, Chair

_____
Clara Novoa

_____
Vedaraman Sriraman

Approved:

_____
J. Michael Willoughby
Dean of the Graduate College

FAIR USE AND AUTHOR'S PERMISSION STATEMENT

Fair Use

Duplication Permission

ACKNOWLEDGEMENTS

I would like to thank Dr. Ameri for his guidance and patience during the planning and development of this research work, in addition to providing me with all of the resources I needed to succeed.

I would like to express my very great appreciation to Dr. Novoa for the collaboration and long hours. This work would not have been possible without the direction and help she provided.

I would also like to thank Dr. Batey for his robust instruction in research methodology and his continuous support through the duration of the graduate program.

This manuscript was submitted on May 7th, 2013.

TABLE OF CONTENTS

Page

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

ABSTRACT

INTEGER PROGRAMMING FOR DISCRETE OPTIMIZATION

OF THE AGILE SUPPLY CHAIN CONFIGURATION

PROBLEM

by

Hayden Dan Beauchamp, B.S.

Texas State University-San Marcos

August 2013

SUPERVISING PROFESSOR: FARHAD AMERI

In order to keep manufacturing operations in lockstep with current market trends, businesses must continue to incorporate agility into their supply chains. This includes the ability to assess and select new suppliers quickly. The Digital Manufacturing Market (DMM) and Manufacturing Service Description Language (MSDL) have been devised previously as the necessary IT components for improving the intelligence of the supply chain configuration process. The objective of this research is to enhance the performance of the DMM's search engine by incorporating combinatorial optimization techniques. In particular, this research is aimed at creating an integer programming formulation to efficiently and effectively solve the supply chain configuration problem by maximizing the technological competencies of the assigned suppliers, while meeting capacity and distance constraints. The column generation approach is adopted to resolve the issue of limited scalability of the traditional LP formulation. Vendor cuts are proposed as a

method to constrain distance inside the supply chain network. The proposed column generation formulation successfully enables transition from a computationally prohibitive methodology to a fully scalable model that maintains functionality at very large sizes. The results also show that it is possible to achieve an economy of distance with little effect on match compatibility.

I. INTRODUCTION

Manufacturing companies are progressively improving the responsiveness, flexibility, and agility of their supply chains in order to maintain the capability to meet market demand and react promptly to unforeseen changes. To obtain agility and responsiveness, manufacturers should be able to adjust their capacity and capability in a timely manner through identification of the right partners. This research deals with the agile supply chain configuration problem. The objective is to improve the effectiveness of the connection process between buyers and sellers of manufacturing services in a distributed environment. New solutions are called for to accommodate the specific needs of agile supply chains, such as increased demand for speed and accuracy while making sourcing decisions at the strategic and tactical levels. Deployment of an agile supply chain can be difficult because it requires rapid identification and evaluation of suppliers that have the necessary technical capabilities and operational capacity to complete the requested services from an increasingly vast pool of potential suppliers.

One promising solution for improving the efficiency and effectiveness of the agile supply chain deployment process is to adopt the paradigm of e-commerce for trading *manufacturing services.* Electronic marketplaces (e-markets) for manufacturing services currently exist for certain industries such as contract manufacturing (MFG.com, Inc., 2013), fashion apparel (Messe Frankfurt Group, 2013) and transportation (Cayot, 2007). A web-based framework allows for interaction with a far greater number of potential suppliers, thus providing the customer with a wide array of options in terms of

manufacturing capabilities, costs, etc. Moreover, the automation capabilities offered by web-based solutions improve the efficacy of various computational tasks required for supply chain management, including supplier identification and evaluation. Despite their numerous advantages, electronic marketplaces currently fail in building accurate connections between buyers and sellers of manufacturing services mainly due to the syntactic (keyword-based) nature of the search process. To address the need for enhancing the performance of search engines in electronic marketplaces Ameri and Dutta proposed a semantic approach to supplier discovery through developing a market framework, called Digital Manufacturing Market (DMM), based on the Semantic Web (SW) technology (Ameri, 2008). In DMM buyers and sellers of manufacturing services describe their capabilities and needs using a formal ontology called Manufacturing Service Description Language (MSDL) (Ameri, 2006). By using the ontological representation of supply and demand entities, semantic search engines can *quantify* the similarities of service provisions (advertisements) and service requests (queries). For each advertisement-query pair, the semantic search engine returns a numeric similarity score between 0 (completely dissimilar) and 1 (completely similar). Finding the semantic similarities of supply and demand on a one-to-one basis can adequately address the needs of the supply chain configuration problem when the supply chain is small in size and composed of few suppliers. However, as supply chain size increases, the optimal configuration of the supply chain becomes more challenging. The optimal supply chain in DMM is the one with the maximum similarity score. There exist, however, multiple types of constraints (e.g. capacity and distance constraints) which complicate the optimization problem and render it difficult to solve when dealing with a large number of potential

suppliers. Because DMM is a web-based platform, it is envisioned that the supply and demand pools will be fairly large; each composed of thousands of businesses distributed globally. Therefore, a methodology needs to be developed for optimization of supply chains configured for the DMM. One research outcome which would fulfill the needs of this market transaction phase is an optimization model that can be applied to improve the efficacy of the buyer/seller connection process.

II. PROBLEM STATEMENT

Supply chain configuration is a multi-criteria problem in which multiple factors such as technological competency, geographic location, capacity, quality, delivery time, and cost should be taken into consideration. The use of a web-based platform for sourcing allows for access to large pools of suppliers for constructing agile supply chains. Large-scale optimization can be used to design optimal supply chains. However, the size of the solution space of the optimization model is directly correlated to the number of supplier agents. In other words, computational complexity increases as the supplier pool grows. In order to arrive at an optimal solution with modest computational resources, a mathematical methodology suitable for large-scale problems is needed.

Research Objective

The objective of this study is to define the agile supply chain configuration problem and identify the characteristics of a suitable analytical tool to supplement the search algorithms of the DMM scoring agent. The primary function of this tool is to generate optimal supply chains with respect to the degree of similarity between the aggregate manufacturing capabilities of the supply chain and the manufacturing needs of the target work order. The validity of the resulting supply chains, form the technological capability point of view, is already verified by the semantic search engine within the DMM platform.

The explicit objective of this research, as will be demonstrated below, is to establish a **discrete integer programming formulation** to efficiently and effectively

solve the supply chain deployment problem by maximizing the semantic similarity score between the proposed supply chain and the query, while meeting the capacity and distance constraints.

Background

Supply Chain Management

Management is practiced through operations incorporating the acquisition and allocation of resources to realize organizational goals. Those who process or distribute products or services practice supply chain management (Shapiro, 2006). While effective business management involves an integrated approach, truly efficient supply chains are achieved through integrated business planning. Indeed, some businesses advertise the economic benefits that customers experience due to advanced logistics. Integrated planning is primarily possible because of the recent development of robust information technology (IT) systems. Still, extensive and advanced IT systems are of little use without the proper analytical tools. The integration of supply chain activities is achieved through the implementation of enterprise resource planning (ERP) systems or the like. Although transactional IT systems have proven valuable towards integration, flexibility seems to be a common issue (Shapiro, 2006). In order to realize improvements in decision making, manufacturing and distribution companies are emphasizing analytical IT. Diagnostic systems are primarily driven by optimization models, which can be net-revenue maximizing or cost-minimizing. With access to large transactional databases they can fully quantify the trade-offs between cost, time, and quality of service or product. These models can be used to gain control of the costs and risks associated with

vehicle routing, non-value-adding supply chain costs, process throughput and virtually endless components of ever more elaborate supply chains.

As we have seen, modern supply chain management is practiced through the effective use of data, models and modeling systems. The purpose of this research is to identify an adequate modeling system and to develop functional models to solve the agile supply chain deployment problem.

Supply Chain Networks

A supply chain is composed of facilities, internal or third-party, which modify or handle a company's products (Shapiro, 2006). Transportation links identify the route that products flow through, from outside the network as raw materials, between factories and distribution centers, and finally to distributed markets. At minimum, supply chain management addresses functional integration, i.e., consideration of manufacturing, transportation, warehousing, purchasing and inventory management operations during decision making. This is accomplished through the integration of specialized forecasting and simulation models which optimize each component.

Network Flow Models

The agile supply chain configuration problem requires that we address transportation and manufacturing. The classic transportation problem is a special case of the network flow model, which is not applicable, while the general network flow problem resembles some aspects of our problem. The penultimate goal of a model which optimizes supply chains designed in DMM is to reduce transportation costs while providing compatible matches. By simplifying transportation costs the problem can be partially represented with a minimum cost network flow model. The ultimate goal of the

model is to produce accurate matches while controlling cost. By adding the certain elements of the network flow model to the primary formulation, we can limit distance while designing effective supply chains.

Automated Matching System

Automation and intelligence are the two most important requirements for virtual supply chain deployment (Ameri & McArthur, 2011). Traditional methods for matching supply and demand in a virtual marketplace include keyword search, directory search and database search. By using an ontological approach, the matching process can be performed more accurately (Farhad Ameri & Dutta, 2008). The semantic supplier discovery process can improve both the intelligence and the automation of pairing by reducing errors and human resource requirements.

In previous research, an agent-based model was developed for supply chain configuration. In the proposed model, the buyers and sellers are represented by blackboard and yellowpage agents respectively (Ameri & Patil, 2012). The middle agent conducts the search and matching process. The seller subscribes to a yellowpage agent to offer services, while the buyer submits a request-for-quote (RFQ) to a blackboard agent. Customers and suppliers each formulate a query to advertise their preferred match. Each query uses MSDL to outline the service, actor and part for consideration by the middle agents. Ideally, patrons of the DMM will be able to choose from multiple middle agents to align their needs with certain industries or markets. Matches are made when each actor queries a complimentary agent. A customer agent would describe the nature of the manufacturing service required, the relevant part specifications and the distinctions that an ideal supplier would have. Then, a search engine would match the buyers query to

information provided by suppliers looking to perform similar work. Conversely, supplier agents would advertise their general manufacturing capabilities, as well as the specific manufacturing processes they are willing to perform. When a customer queries the yellow page agent the search algorithm of the machine agent matches the request with sellers based on the current directory information. The search algorithms, and the overall methodology of the search engine, determine how accurately the proposed match reflects the query. The search engine tests potential matches, then scores and ranks them. However, no optimization is performed to improve the overall similarity score of the resulting supply chain. This work is aimed to enhance the performance of the supply chain configuration process by utilizing optimization techniques.

Operations Research

Operations research is the scientific and technological methodology of decision making (Shapiro, 2006). Decision making is improved by constructing mathematical models that represent real world problems. A model is populated by data to create numerical systems that can be optimized by through the application of an *algorithm*. Technology primarily serves as the computational resource that applies the mathematical method. Together, they form a decision-making engine.

Mathematical Programming

The area of applied mathematics called mathematical programming involves the optimization of a criterion function subject to constraints. By formulating a deterministic optimization model that accurately represents a problem, a solution to that problem can be found with the appropriate mathematical tools. An optimization model is essentially a set of equations and inequalities. Solving the system of equations produces feasible

solutions. This cannot be done algebraically, as practical problems are too complex to be manually solved. Some problems can employ a simpler methodology through the application of *heuristics*. That is to say, models can be solved with a rules-based approach. In other words, prescriptive analysis can yield feasible solutions with limited computation. However, heuristics solutions are often limited to approximations, and can confer performance benefits when used to supplement more effective algorithms. Thus, results from purely heuristic approaches are usually *per contra* to expectations. Computers are required to apply known methods to produce final solutions. First, solution algorithms are used to construction a computer program (solver). Then, solvers are collected to form an optimizer, the brain of an analytic IT system. An operations research study is performed to model a system. Exact optimal solutions can be found through the application of the appropriate solvers to a well-formulated optimization model. When these decisions are implemented they can lead to impressive improvements in revenue, lead time, risk, etc.

Linear Programming

A special case in mathematical programming exists when the objective value is directly proportional to the inclusion of its variables. In other words, the program is driven and constrained by linear equations. This, then, is called a linear program (LP). Because so many practical problems can be modeled as linear programs, established techniques and tools are available to solve them.

Discrete Optimization

In some cases, certain variables must assume whole number values, and are classified as discrete optimization models. Furthermore, integer programs (IPs) are those

that contain decision variables with integrality conditions. When such a model retains variables which are free to assume real values it is classified as a mixed-integer program (MIP). Integer linear programs (ILPs) then, are discrete LPs. In this case the conventional practice of linear programming does not apply. For this reason these problems are much harder to solve. However, improvements in methodology and technology have made the solution of complex integer programs more feasible.

Mosel Programming Language

The FICO™ Xpress Optimization Suite is used to model the problem the computer. Algorithms are constructed using the IVE console where the appropriate solvers can be applied. XpressMP uses Mosel because it is a modeling language that also functions as a programming language. Therefore, modeling statements call forth the required procedures needed to solve the problem. In this way, modeling statements and solving statements a can be intelligently designed to explicitly define solution algorithms for large scale optimization. Different types of problems employ different functions, and therefore require different solvers. In order to maintain operational efficiency the Mosel system allows the user to assign particular modules to interface only with the required solvers. The distinct lexis of each module defines a novel set of functions and procedures, extending the capacity of the dynamic modeling environment as needed.

To be effective in application Mosel is designed with an open architecture for use with other software. The e-commerce platform will relay query information to the manufacturing services ontology database, where it will be processed into similarity score data. The supply chain optimization model will access the database to retrieve the processed query data to perform the optimization. Although the mathematical

programming research is performed using a standalone program, the software which runs the final application will replicate its own modeling environment using Mosel libraries. In this way, a global software application can perform the modeling and solving tasks in series with the other operations necessary to provide the agile supply chain design service.

III. LITERATURE REVIEW

As a cross-functional driver of supply chain performance a firm's sourcing is integral to remaining competitive in today's rapidly changing markets (Shapiro, 2006). Businesses are continuing to incorporate agility into their supply chains in order to keep their manufacturing operations at pace with product development. Typical sourcing processes include supplier scoring and assessment, supplier selection and contract negotiation, design collaboration, procurement, and sourcing planning/analysis (Shapiro, 2006). Analytical tools are typically relied upon to make supplier selection and purchasing decisions, which are the processes that the DDM platform will serve to facilitate. Supplier assessment and evaluation is modeled as a supplier selection problem, while purchasing metrics like order quantity are determined by the order allocation problem.

Supplier Selection Problem

The underlying technical problem of this thesis can be classified under the general category of *supplier selection and order allocation* problems. Supplier selection is a multiple criteria decision-making problem (Çebi & Bayraktar, 2003). A typical optimization model in supplier selection has multiple objective functions to address multiple criteria, such as minimizing the purchasing price and manufacturing lead time, and maximizing the quality of the finished goods (Huo &Wei, 2008). Optimization models initially included up to 23 parameters for evaluating suppliers, including capacity,

delivery time and quantity-based price discounts (Dickson, 1966). Since then additional studies have identified up to 60 criteria used to assess suppliers (Roa & Kiser, 1980).

The supplier selection and order allocation problems have been studied for over fifty years. Researchers and practitioners have developed and implemented approaches to model these problems using mixed integer linear programming (MILP), multi-objective programming (MOP), goal programming (GP) and nonlinear programming (NLP) (Sanayei, Farid Mousavi, Abdi, & Mohaghar, 2008). Some examples of the methods used include IP with tabu search heuristic (Ko, Kim, & Hwang, 2001), Goal Programming (Karpak, Kumcu, & Kasuganti, 1999). Fuzzy-Multi-objective Integer Programming (Hue & Wei, 2008), Genetic Algorithms (NLP) (Ding et al., 2004), Analytical Hierarchy Process (AHP) (Nydick & Hill, 1992), Artificial Neural Network (ANN) (Wu, Zhang, Zheng, & Xi, 2010), and Analytical Network Process (ANP) (Kirrytopoulos, Leopoulos, Mavrota, & Voulgaridou, 2010), Multi-attribute Utility approach (MATU) (Min, 1994), data envelopment analysis (DEA), voting analytical hierarchy process (VAHP), total cost approach (TCA), and artificial intelligence methods. AHP can be combined with linear programming to include order allocation problems across multiple suppliers (Ghodsypour & O'brien, 1998) or with other methods. These hybrid approaches have varying degrees of success (Sanayei, Farid Mousavi, Abdi, & Mohaghar, 2008).

Chamodrakas, Batis and Martakos (2010) simplified the supplier selection problem in a business to business e-market environment by breaking it into two stages. In the initial screening stage satisficing can be used to qualify vendors. The computational complexity of the problem in the final supplier selection stage is greatly reduced in

comparison to the unpruned model, and can be solve with Fuzzy Parameter Programming (Chamodrakas et al., 2010).

IV. RESEARCH METHODOLOGY

The manufacturing services e-commerce platform requires a customized optimization technique to drive the decision-making process. The DMM framework provides a distinct advantage when solving the supplier selection problem. The degree of complexity needed to model typical supplier criteria is responsible for producing the variety of modeling systems and solution schemes described above. The DMM approach involves the decomposition of the matching process into multiple phases, namely, supplier assessment, evaluation and selection. MSDL is used as an ontology for the description of customer RFQs and supplier advertisements for generating similarity scores in the supplier assessment phase. Three principal components of semantic descriptions are stored in the DMM system. Both qualitative and quantitative descriptions of manufacturing process capabilities and related part specifications and constraints, either advertised or required, reflect manufacturing service capabilities. The third type of characteristic is supplier criteria like quality level and service level. MSDL is also used to describe both qualitative and quantitative attributes, as well as tangible and intangible criteria. The automated matching system quantifies the similarity between a query and its prospective counterparts. Possible matches are scored, likely occurring in multiple stages, similar to the approach developed by Chamodrakas et al. (2010), in order to reduce the computational complexity of the assessment process. This research deals with the supplier evaluation phase. The analytical tool proposed in this paper is intended to optimize the assessment scores of assigned suppliers while considering capacity and

distance constraints. When the problem is decomposed in this way, the burden of computation falls on the similarity assessment agent. Additional algorithms will need to be developed to quantify similarity with and expanding list of supplier criteria. The supplier selection model proposed here needs only a single objective of maximizing the aggregated similarity score. Future formulations of the supplier evaluation model can include more supplier criteria and constraints. The requirements of the analytical tool for the evaluation phase, as currently outlined, are much simpler and can be addressed with a preliminary operations research study.

The operations research (OR) study is a fairly standardized process consisting of five phases, namely, problem definition, model construction, model solution, model validation and model implementation (Taha, 2007). Because the practical purpose of an OR study is to methodically improve systems, the final phase is implementation of the solution. This academic study is theoretical. Therefore, feedback from solution implementation, such as collection and analysis of new data, is beyond the scope of this research. However, consideration regarding the possibility of the proposed decision to be implemented is addressed in the model validation phase. Fourer, Gay, & Kernighhan (2002) have decomposed the mathematical programming process into six main steps as shown in Table 1. These have been numbered such that we can use them as a reference as we progress through the MIP development.

Table 1. OR study vs. mathematical programming

Phases of an OR study	Steps of the Mathematical Programming Process	
Problem Definition	1	Outline the scope and assumptions of the proposed analytical tool. Define the variables, objectives and constraints that accurately represent the general form of the problem to be solved.
	2	Collect data from an explicit instance.
Model Construction	3	Using the preliminary formulation and data identify an objective function that characterizes the interaction of the variables which drive value to be optimized. Interpret apparent limitations in the problem which restrict the data into constraint equations.
Model Solution	4	Use a solver program to apply an algorithm and solve the problem instance. The output of the model includes the optimal values of the variables.
Model Validation	5	Analyze the results.
	6	Refine the model, input and output. Repeat as needed.
Model Implementation	Ø	Ø

The component definition is relatively straightforward, while design and refinement of the optimization algorithm will likely iterate to allow for systematic improvements to model construction and solution through various optimization techniques. Because we are not collecting data, it must be generated. Appropriate assumptions regarding data generation are crucial to the validity of the proposed model and solution scheme.

Assumptions

Before starting the mathematical programming process is will be necessary to simplify the system we are attempting to model. In order for preliminary formulation to accurately represent the real-world, certain assumptions must be made. The model can be

revised to include the phenomena outlined by the assumptions in subsequent formulations.

Assumption 1: Raw materials and finished product transportation not considered. Although distances outside the supplier network are traditionally considered in the supply chain network design process, only distances between suppliers will be modeled.

Assumption 2: **Each work order contains independent services.** Some manufacturing services are typically performed in rapid succession or in series with others as part of a larger process, or are otherwise traditionally performed by one supplier for a handful of reasons. Excluding this should have no effect on the validity of the formulation.

Assumption 3: Best match is the primary objective of the model. Here we assume that a higher aggregate score of a proposed supply chain configuration is preferred over those which contain particular suppliers with outstanding score, but may have a lower total overall. Other practical reasons exist for supplier preference. Manual supplier selection is a parameter of query specification. It is therefore a capability of the model regardless of formulation. Thus, it does not impact performance and will not be tested in this study.

Assumption 4: Range of process duration is realistic. Each service requires a certain number of hours per batch to complete, and will vary between suppliers. Each supplier will perform the service within a certain range, which will vary depending on industry and process type. The model will simulate this randomly. The functionality of the model should be the same when handling real data, in this respect.

Assumption 5: Supplier matching scores are random. The scores of the data we are using are randomized. Realistic data would show trends in scores due to the nature of manufacturing process specialization, i.e., the individual capabilities of a manufacturer are interrelated. Nevertheless, a robust model, which can handle randomized scores, is expected to perform well when applied to real samples. This is because real similarity score data for work orders will likely exhibit trends within suppliers. Some suppliers will not possess capabilities in queried areas, while those that do will show a tendency to be suited to multiple services in that work order. These practical scenarios would hypothetically require less distance than randomized problem instances where each supplier is suited to fulfill a fraction of the services in a work order.

Traditional Formulation Without Distance Constraints

Problem Definition

Components

The principal elements of the problem are the decision alternatives, restrictions and the objective criterion used to evaluate the alternatives (Taha, 2007). The *objective* of our supply chain optimization model is to fill needs by matching queried services with offered services. In other words, we are assigning members of one set to those of another. Defining the characteristics of the objective and constraint functions will illustrate the kind of problem we are attempting to solve. We know that variables will be assigned to sets by using binary coefficients, and that assignment constraints will govern the operation of the optimization model. Studying this problem classification will facilitate the construction of the model.

Set Partitioning Problem

Set packing, covering and partitioning models are those that employ binary decision variables to identify objects or agents as part of the solution (Rardin, 1998).

$$x_{jk} \triangleq \begin{cases} 1 & \text{if agent } k \text{ is selected for task } j \\ 0 & \text{otherwise} \end{cases}$$

The type of assignment problem depends on the how it is constrained. Set covering constraints require at least one member of the set of agents $\sum_{k \in K} x_{jk} \geq 1$ to be included in the solution. A notable example involves the use a modified maximal covering model to plan the layout of EMS stations in Austin, TX (Eaton, Daskin, Simmons, Bulloch, & Jansma, 1985). This set covering model was designed to determine the location of each station, and its capacity to provide varying levels of service to overlapping regions with limited equipment, human resources and funding. In other words, each area is assigned at least one agent to perform every type of service. Alternatively, set packing models involve the limitation of agent assignments to one at most. Example problems include packing a set of items into the smallest number of containers and packing the most items into a fixed number of containers. While covering and packing constraints bound assignment, *set partitioning* constraints require the assignment of exactly one agent $\sum_{k \in K} x_{jk} = 1$ to each member of the solution set. Likewise, our supply chain design problem requires that every service be performed by one supplier. Therefore, it is a set partitioning problem. Still, it can be further classified.

Generalized Assignment Problem

Because the specific objective is to maximize the aggregated score of all matches by assigning each requested service to one agent without violating its capacity, this is a generalized assignment model (Rardin, 1998). The Generalized Assignment Problem

(GAP) is significant because its definition underpins models formulated to solve problems in resource scheduling, facility location, vehicle routing and manufacturing systems (Savelsbergh, 1997). In fact, the GAP foundation has been used to support optimization in crew scheduling, stochastic batch-sizing, capacitated lot-sizing, sports league scheduling, service network design, political districting, efficient production-distribution system design, flexible manufacturing systems, vehicle routing with simultaneous distribution and collection, and much more.

Data from Problem Instance

A small example (toy problem) is used to provide a working reference for the mathematical programming formulations. The toy problem will also provide an illustration of how each model operates. The visual aids that follow will illustrate only the smallest size problem. A range of large scale optimizations will be run, measured and compared. These robust comparisons will provide the most meaning to the OR study.

An LP has been formulated to model the toy problem for this application. Microsoft Excel is used to digitally model the LP and its solver generates the optimal solution, as verified by manual computation. This will be used to validate the solution provided by XpressIVE. The basic LP generates a valid solution to the toy problem, thus confirming definition and implementation of the optimization model.

Model Construction

The linear program formulated using a traditional discrete programming approach has one objective function (i.e. the equation that quantifies the decision consequences) which aggregates individual service-supplier scores of the selected suppliers. The objective of the model is to maximize the overall score by assigning values to the binary

decision variables. In effect, it will provide high-scoring matches while satisfying all constraints.

$$\text{Max} \quad z = \sum_{i=1}^{I} \sum_{j_i=1}^{J_i} \sum_{k=1}^{K} Score_{ij_ik} * x_{ij_ik} \qquad \text{(TOF)}$$

$$\sum_{k=1}^{K} x_{ij_ik} = 1 \qquad \forall \begin{cases} i = 1, \dots, I \\ j_i = 1, \dots, j_i \end{cases} \qquad \text{(T1)}$$

$$\sum_{i=1}^{I} \sum_{j_i=1}^{J_i} d_{ij_i} * x_{ij_ik} \leq C_k \qquad \forall \quad k = 1, \dots, K \qquad \text{(T2)}$$

$$x_{ij_ik} \triangleq \begin{cases} 1 & \text{if } k \text{ is selected for service } j_i \\ 0 & \text{otherwise} \end{cases} \qquad \text{(sign constraint)}$$

The optimization model uses parameters and data provided by the toy problem to create a pool of possible solutions. If formulated correctly, the LP should provide the optimal solution. A solver will assign values to the decision variables in such a way that the highest value for the objective function is produced. The formulation uses binary decision variables, linear constraints, and a single linear objective function.

Decision Variables

The decision variables are a matrix of binary values that determine the inclusion of customer/supplier matches on a service-by-service basis.

$$\begin{bmatrix} x_{11_1,1} & \cdots & x_{11_1,K} \\ \vdots & \ddots & \vdots \\ x_{IJ_I,1} & \cdots & x_{IJ_I,K} \end{bmatrix}$$

For example, $x_{ij_ik} = 1$ if service j_i in work order *i* is performed by supplier *k*. Table 2 shows that the rows summations in the array of decision variables is constrained to one.

Table 2. Toy problem decision variable matrix

	Supplier 1	Supplier 2	Supplier 3	Supplier 4	Supplier 5	
WO1						
S_{11}	X_{111}	X_{112}	·	·	X_{115}	=1
S_{21}	X_{121}				·	=1
S_{31}	·				·	=1
WO2						
S_{12}	·				·	=1
S_{22}	·				·	=1
S_{23}	·				·	=1
WO3						
S_{13}	·				·	=1
S_{23}	·				·	=1
S_{33}	X_{331}	X_{332}	·	·	X_{335}	=1
	$<=C_1$	$<=C_2$	$<=C_3$	$<=C_4$	$<=C_5$	

After the decision variables (x_{ij_ik} inside the table are either 1 or 0) are chosen, the scores ($score_{ij_ik}$) associated with those pairs are aggregated to produce the objective value. The Boolean inclusion of the individual scores is affected by multiplying an array of scores, of identical dimensions, by this table. The overall score is quantified by summing the score of each assignment.

Parameters/Input Data

The toy problem is used to simulate practical application of the model by providing parameters of a workable size.

$Dmax_i$ is the pre-defined threshold for total distance between suppliers in work order i

d_{ij_i} is the duration of/capacity required by service j_i in work order i

C_k is the total available capacity of supplier k

$score_{ij_ik}$ is the matching score of assigning supplier k to service j_i in work order i

Table 3 illustrates how these input parameters are input in spreadsheet form to the Microsoft Excel solver.

Table 3. Similarity score input to Excel solver

	Supplier 1	Supplier 2	Supplier 3	Supplier 4	Supplier 5	
WO1						Duration
S_{11}	0.64	0.32	0.50	0.43	0.71	3 h
S_{21}	0.23	0.63	0.95	0.58	0.01	7 h
S_{31}	0.15	0.56	0.40	0.42	0.54	9 h
WO2						
S_{12}	0.11	0.32	0.38	0.67	0.89	11 h
S_{22}	0.55	0.67	0.34	0.78	0.29	10 h
S_{23}	0.00	0.00	0.00	0.00	0.00	0 h
WO3						
S_{13}	0.39	0.58	0.22	0.33	0.24	5 h
S_{23}	0.41	0.08	0.92	0.62	0.78	8 h
S_{33}	0.29	0.45	0.34	0.07	0.11	3 h
Capacity	17 h	13 h	12 h	19 h	17 h	

The score provides a virtual quantification of similarity between services demanded and processes supplied in the digital manufacturing market. The LP matches individual service requests with suppliers based on the score and constraints. The objective value is driven by the objective function coefficients (score) and the Boolean assignment of score inclusion via corresponding binary decision variable.

Objective Function

The objective function is used to rate decisions. The objective function of this model is a linear equation that contains all of the decision variables. The Traditional Objective function (TOF) is used to maximize the matching scores while assigning all services requested in work orders to the available suppliers.

$$\text{Max } z = \sum_{i=1}^{I} \sum_{j_i=1}^{J_i} \sum_{k=1}^{K} Score_{ij_ik} * x_{ij_ik} \qquad (TOF)$$

The objective function is essentially a weighted sum, where the coefficients that value each decision variable are similarity scores associated with each possible match between service and supplier. The objective of the model is to maximize the total score while satisfying any constraints.

This LP has a solution space which is unimodal. The linear constraints form a convex set, and the objective function is concave. The purpose of the objective function is to rank the values of the decision variables in such a way that, when any three are modeled in a solution space, the coordinates which represent the intersection of a point in the feasible region of the multi-dimensional objective function curve generated produce the highest score aggregate and therefore the optimal solution.

Constraints

Some simple constraints can be applied to define boundaries in the solution space. All constraints in the LP are equations or inequalities that serve to define the practical validity of possible solutions, based on limitations of the systems modeled in the toy problem.

$$\sum_{k=1}^{K} x_{ij_ik} = 1 \qquad \forall \begin{cases} i = 1, \dots, I \\ j_i = 1, \dots, j_i \end{cases} \qquad \text{(T1)}$$

Constraint T1 requires each service, in each work order, to be assigned to a single supplier.

$$\sum_{i=1}^{I} \sum_{j_i=1}^{J_i} d_{ij_i} * x_{ij_ik} \leq C_k \qquad \forall\ k = 1, \dots, K \qquad \text{(T2)}$$

Constraint T2 prevents the optimizer from creating solutions which would require suppliers to exceed their capacity in order to fulfill the work orders requested.

Model Solution

The basic LP model generates a valid solution to the toy problem as supported by the Excel solver results, thus confirming the definition of the problem in Mosel and implementation of the optimization model in XpressMP.

Model Scalability

The original toy problem contained 5 suppliers and 45 decision variables. The analytical tool that is need for this application will ultimately handle hundreds of thousands of suppliers. However, the computation resources available for this study will only be able to process a fraction of that number. The preliminary formulation must first be failure tested to establish a range of feasibility for this machine (Intel® Core™ i7 2600 CPU @ 3.4 GHz, 16.00 GB of RAM, Microsoft Windows 7, 64-bit Operating System). Problem size is a product of number of total services and suppliers. Problems with 25, 50, 75, 100, 125, 150, and 175 suppliers will each have two levels of services. Five replicates will be tested for each size. The runs are randomized and run serially by a master program. The results of each run are transmitted automatically to an Excel database.

The original formulation assumed for simplicity that each supplier consumes identical capacity resource to perform each service. In addition to being unrealistic, it has also been proven to make very large problems more difficult to solve because of a phenomenon called symmetry (Savelsbergh, 2002). Therefore, the service duration parameter is reformulated to an array with 3 dimensions.

Distance Constraints

The general formulation of the GAP is insufficiently bounded for our application. The traditional LP model has been constructed to deliver optimum query/provision matches regardless of transportation costs. There are multiple ways to constrain the model to address these costs. Each approach will observe the following: Distance constraints in this study are intentionally oversimplified in order to easily assess risk and performance metrics like transportation costs, transportation risk, delay risk and transit time. These factors impact operational costs and drive sourcing decisions in supply chain management. This thesis will deal with these drivers in a general way. While some of these costs are indirect, all are directly correlated with distance. Therefore, the magnitude of a transportation link is a general predictor of its associated costs. By reducing cost input to arc distance, the network flow problem can be optimized using a minimum cost flow model. Future models can include more specific parameters and constraint functions in order to approach optimization in a more realistic manner.

Traditional Formulation with Distance Constraints

Nonlinear Programming

Distance threshold (D_max) values are used to limit the total distance traveled in each work orders in a solution in order to meet the transportation cost and time constraints specified by the customer. Therefore, the virtual constraint *total distance traveled in work order* $i \leq D\_max_i$ is to be applied in the algorithm. The traditional mathematical programming approach involves the inclusion of an explicit distance constraint which defines the relationship between work order distances and threshold parameters as a set of inequalities. The T3 constraints below are similar to those used to

define the objective function of a quadratic assignment model. They sums the distance of all transportation links between suppliers, shown in Table 4.

In Table 8 the distance matrix contains all distance between suppliers.

Table 4. Distances between any two suppliers k and l (mi) D_{kl}

	Sup_1	Sup_2	Sup_3	Sup_4	Sup_5
Sup_1	0	34	45	87	22
Sup_2	34	0	89	69	23
Sup_3	45	89	0	13	35
Sup_4	87	69	13	0	18
Sup_5	22	23	35	18	0

While most decision variables will become zeros, the assignment of a supplier to a service is represented by a one. The double decision variable coefficients in these inequalities produce non-zero values only when a pair of decision variables is quantified. In this way, only the distances of the chosen transportation links are aggregated.

$$\sum_{j_i=1}^{J_i-1} \sum_{k \in \text{suppliers}} \sum_{l \in \text{suppliers}} D_{k,l} * x_{ij_ik} * x_{i(j_i+1)l} \leq D_{max_i} \quad \forall \; i = 1, \dots, I \quad \text{(T3)}$$

While the T1 and T2 constraint families are arrays of linear constraints, T3 are dynamic sets of quadratic constraints. Therefore, the model becomes a nonlinear program (NLP), specifically a nonlinear integer program. NLPs require a different approach, and the combinatorial nature of NLIP are especially difficult to solve. The Xpress SLP solver is used to make successive linear approximations of the mixed integer linear formulation. Nevertheless, nonlinear integer programs are difficult to solve and may not be appropriate for the size aspirations of the agile supply chain design platform.

Traditional Formulation with Cuts Added Dynamically

Our model is deterministic because all of the parameters are knowable. Dynamic programming involves the decomposition of a problem in to smaller components to be solved recursively. Deterministic dynamic programming is used to solve shortest path problems and could be used to model distance in the agile supply chain design problem. However, solving models recursively means that the final solution depends on the initial solution. Dynamic programming finds local optima and would not allow us to find exact optimal solutions. Nevertheless, it may still be advantageous to break down the model into smaller parts. We will explore this concept later with an alternative formulation called column generation. First, the concept of dynamic constraint declaration will be considered as a causal solution to the distance problem.

Enforcement of the distance thresholds, as we have seen, cannot be addressed with a global constraint without reducing the tractability of the model. A simpler approach called work order assignment cuts can theoretically accomplish the same end via different means. The model can iteratively introduce case-specific constraints, which together bound the solution to distance thresholds.

Dynamic Constraint Introduction

In order to produce feasible solutions without enumerating all possible permutations of transportation links before running the program, individual constraints can be introduced on an ad hoc basis. The model will produce the solution with the highest objective value the first time the LP is solved. The model does not terminate after the first solution is generated because it may be infeasible with respect to distance constraints. The distances between suppliers assigned in each work order are summed.

The total distance cannot exceed the arbitrary threshold value (T3). For each work order which fails the distance threshold test, the program will automatically add a constraint which prohibits that particular assignment of supplier matches.

$$\sum_{j_i=1}^{J_i}\sum_{k=1}^{K} x_{ij_ik} < |J_i| \qquad \forall \ \ i \text{ which fail distance test} \qquad (T3\hat{\imath})$$

After the new vendor cuts constraints are added for iteration $\hat{\imath}$ the optimizer is prompted to resolve the problem. This process repeats until all distance constraints are satisfied. Table 5 shows an example where the optimizer chooses suppliers 2, 1, and 4 for work order one. It is important to note that while the the services in each work order are performed sequentially, a certain combination of suppliers has a specific total distance, indifferent to order.

Table 5. Example solution for one work order

		Sup_1	Sup_2	Sup_3	Sup_4	Sup_5	RHS
WO_1	S_{11}	0	1	0	0	0	=1
	S_{21}	1	0	0	0	0	=1
	S_{31}	0	0	0	1	0	=1
	Order	2nd	1st	-	3rd	-	

The scores associated with these choices are high. However, total distance in work order 1 exceeds the distance threshold. Therefore, the constraint $x_{112} + x_{121} + x_{134} \leq 2$ will be added to the model to prevent this particular combination of suppliers from appearing in future runs. Once this is done for each disqualified work order, the LP is solved again. In this way, invalid solutions can be individually identified and eliminated. The objective value will decrease each time the model is solved. After the model has eliminated enough invalid solutions, it will arrive at highest valid objective value. This, then, is the optimal solution of the vendor cuts method.

The dynamic addition of cuts appears to be the most computationally efficient feasible way to constrain distances inside the network. An alternative method would require distance summations for every combination of suppliers to be computed prior to running the model. As problem size grows, the time require for this operation explodes. Massive computational power would be needed to solve real world problems of even modest size. Still, issues associated with computing speed will naturally decrease over time. It is for the reasons previously discussed, then, that causal constraint definition fits our problem better than global declaration.

After the traditional model is developed it will be used to support the validity of solutions provided by a more advanced LP model called column generation (Desaulniers, Desrosiers & Marius M., 2005). The Branch and Price technique can also be used to solve these problems. It is, however, a nuanced methodology. Due to time constraints, only a preliminary investigation was included in the scope of this study.

Survey of Problem Size

Multiple sizes of toy problem will be run in order to identify the effect of experimental variable *number of decision variables* on the dependent variables: total number of runs, CPU runtime using XpressIVE, etc. The number of decision variables is dictated by number of suppliers and total number of services. If the toy problem has I = 3 work orders, J = 9 services and K = 5 suppliers, then there are 45 decision variables. As we scale up the toy problem, computation becomes cumbersome. One of our hypotheses is that the traditional approach is impractical for solution of realistically-sized problems.

Experimental Design

The traditional formulation will be tested in different sizes to observe the behavior of the model as problem size increases. A few experiments will test the predictions concerning the effect that each type of variable has on the performance of the optimization model. These hypotheses are outlined below.

DOE 1: Solving the LP to optimality will become computationally prohibitive

H_0: The solver will continue to be efficient as problem size grows

H_1: The solver will not be able to explicitly generate columns for large sizes

DOE 2: Increase in supplier-to-service ratio will improve tractability

H_0: No change will be observed between cases with disparate m/n ratios

H_1: Increases in available suppliers will cause a decrease in computation time

DOE 3: Scarcity of supplier capacity will negatively impact performance

H_0: Changes in supplier capacity levels will not affect the CPU solve time

H_1: Optimization will take longer for cases with less available capacity

The performance can be improved by solving these problems using a column generation approach. Decomposing the problem will produce fewer decision variables than the traditional formulation, and maintain optimality at a much higher range than the explicit formulation.

Column Generation

When linear programs have huge numbers of variables, it is sometimes not possible to explicitly generate all columns of the decision variable matrix. Column generation algorithms are employed to systematically redact and solve sets of columns and add those that improve the current solution to the problem matrix. Using this iterative

process the optimal solution can be reached through successive identification of best partial solutions.

Column generation tractability depends on the number of alternatives b_k generated for each supplier *k.* For example, if there are 5 suppliers and b_k is 3 for supplier 1, 2 for supplier 2, 2 for supplier 3, 2 for supplier 4, and 1 for supplier 5, the formulation will generate a table with 10 columns. This table has 5 sets of 1 to 3 complimentary. The decision to choose which 4 will supplement any 1 column will result in one of 24 alternatives. This third iteration of the decomposed problem has 10 decision variables, rather than the 45.

Problem Definition

Decision Variables

In the disaggregated formulation possible assignments for each supplier are generated by a sub-problem. Assignments included in the solution are determined by a decision variable which weights each column. The master problem decision variables $y_{k^{(b_k)}}$ are used to weight columns. $y_{k^{(b_k)}} = 1$ if column *b* of supplier *k* is selected to satisfy any services.

The $r_{ij_ik^{(b_k)}}$ matrix is composed of input data from the solution of sub-problems which are generated at each iteration. These values substitute for the decision variables of the traditional formulation. An $r_{ij_ik^{(b_k)}}$ variable is equal to 1 if service j_i in work order *i* is supplied in column l of supplier *k* and 0 otherwise. An initial subset of columns is provided for the master problem to solve. The matrix will expand as it is populated by new columns between iterations. The sub-problems will continue to generate novel ways

until no more exclusive columns can be generated or the solver can verify that a solution is optimum.

Parameters

$Dmax_i$ is the pre-defined threshold distance for suppliers included in work order *i*

d_{ij_ik} is the capacity consumed by service j_i in work order *i* by supplier *k*

C_k is the total available capacity of supplier *k*

$score_{ij_ik}$ is the matching score of assigning supplier *k* to service j_i in work order *i*

$Total\_Score_{k^{(b_k)}}$ is the total matching score of all assignments in column *b*

$$Total\_Score_{k^{(b_k)}} = \sum_{i=1}^{I} \sum_{j_i=1}^{J_i} score_{ij_i\langle k\rangle} * r_{ij_ik^{(b_k)}} \qquad \forall \begin{cases} k = 1, \dots K \\ b_k = 1..B_k \end{cases}$$

Model Construction

Objective Function

$$\text{Max } Grand\_Score_{colgen} = \sum_{k=1}^{K} \sum_{b_k=1}^{B_k} Tot\_Score_{k^{(b_k)}} * y_{k^{(b_k)}} \qquad (CGOF)$$

Constraints

Assignment constraints CG1 supervene over the columns of the disaggregated formulation basis. CG1 are row constraints and will remain in the master problem. Alternatively, supplier-specific constraints from the original model can be declared in the sub-problems.

$$\sum_{k=1}^{K} \sum_{b_k=1}^{B_k} r_{ij_ik^{(b_k)}} * y_{k^{(b_k)}} = 1 \qquad \forall \begin{cases} i = 1, \dots, \mathrm{I} \\ j_i = 1, \dots, J_i \end{cases} \qquad (\mathrm{CG1})$$

Constraint CG1 is essentially the same as TF1 which states that each service in each work order must be assigned to a single supplier. Dual variables from the

Lagrangian relaxation are used to determine the assignments in the new columns that will be generated by the sub-problem. Constraint CG2 is a constraint on the columns of a particular supplier in the master problem.

$$\sum_{b_k}^{B_k} y_{k(b_k)} = 1 \qquad \forall \quad k = 1, \dots, K \quad \text{(CG2)}$$

The model is capable of solving larger problem sizes because the decomposition scheme involves the delegation of computation to a separate sub-model for each supplier. A sub-model will generate a column of possible assignments for one supplier. The CG2 constraints state that only one set of possible assignments is chosen. The Lagrangian duals from these convexity constraints are used to price the columns such that a stopping point for the iterative process can be identified. The CG2 constraints also help to maintain feasibility of integer solutions. In order to exploit duality we must allow the master problem to assume a strictly linear formulation. Unlike the traditional formulation, there are no integrality constraints. Methods like branch-and-price are devised to resolve difficulties with discrete solution generation. Nevertheless, integrality is not an issue when working with problem instances where capacity is abundant, like in the agile supply chain configuration problem. CG3 is a column constraint, and can be included in the sub-problem.

$$\sum_{i=1}^{I} \sum_{j_i=1}^{J_i} r_{ij_i(b_k)} * d_{ij_ik} \leq C_k \qquad \forall \begin{cases} k = 1, \dots, K \\ b_k = 1, \dots, B_k \end{cases} \quad \text{(CG3)}$$

Each supplier utilizes a different capacity c_{ij_ik} in order to satisfy service j_i. Constraint CG3 prevents assignments from exceeding the available supplier capacity.

$$y_{k(b_k)} \triangleq \begin{cases} 1 & \text{if column } b \text{ selected for supplier } k \\ 0 & \text{otherwise} \end{cases} \quad \forall \begin{cases} k \in \text{suppliers} \\ b_k = 1, \dots, B_k \end{cases} \text{(Sign constraints)}$$

The sign constraints of the sub-problems ensure that the assignments in each column will be integer and binary. Each column represents a partial assignment of services to suppliers. The objective CGOF is to maximize the resulting total score of selected columns such that assignments are given for all work orders considered.

Column Generation Master Problem

The formula for $Total\_Score_{k^{(b_k)}}$ is integrated into the objective function as the coefficient of the decision variables.

$$\text{Max } Grand\_Score_{colgen} = \sum_{k=1}^{K} \sum_{b_k=1}^{B_k} score_{ij_i\langle k\rangle} * r_{ij_ik^{(b_k)}} * y_{k^{(b_k)}} \quad (CGOF)$$

st.

$$\sum_{k=1}^{K} \sum_{b_k=1}^{B_k} r_{ij_ik^{(b_k)}} * y_{k^{(b_k)}} = 1 \qquad \forall \begin{cases} i = 1, \dots, \mathrm{I} \\ j_i = 1, \dots, J_i \end{cases} \qquad (\mathrm{CG1})$$

$$\sum_{b_k}^{B_k} y_{k^{(b_k)}} = 1 \qquad \forall \quad k = 1, \dots, K \qquad (\mathrm{CG2})$$

Note the removal of the integrality conditions for the decision variable $y_{k^{(b_k)}}$. Iteration in the master problem is assumed continuous and thus, no sign constraint is introduced. This allows the optimizer to solve the LP relaxation and find the dual prices.

Sub-Problem

The following integer knapsack problem is solved for each $k \in$ suppliers.

$$\text{Max } price_k = \sum_{i=1}^{I} \sum_{j_i=1}^{J_i} \left[\left(score_{ij_ik} - \pi_{ij_i}\right) * r_{ij_i}\right] - \delta_k$$

st.

$$\sum_{i=1}^{I} \sum_{j_i=1}^{J_i} r_{ij_i\langle k\rangle} * d_{ij_i\langle k\rangle} \leq C_{\langle k\rangle} \qquad (\mathrm{CG3})$$

$$r_{ij_i\langle k\rangle} \triangleq \begin{cases} 1 & \text{if service } j_i \text{ is assigned to } \langle k\rangle \\ 0 & \text{otherwise} \end{cases} \quad \forall \begin{cases} i = 1, \dots, \mathrm{I} \\ j_i = 1, \dots, J_i \end{cases} \quad \text{(sign constraint)}$$

The $r_{ij_i\langle k\rangle}$ decision variables enter the basis of the master problem as $r_{ij_ik^{(b_k)}}$ for iteration b_k of supplier k if the price is positive. The columns of the sub-problems are priced using the dual variables $\boldsymbol{\pi_{ij_i}}$ and δ_k. A nonnegative price indicates a column of positive reduced cost which should enter the basis. The iterative process stops for each supplier which has exhausted all such columns. The column generation process is over when all beneficial columns have been found. The final basis is solved as an integer program. Figure 1 is a flow chart of the column generation algorithm.

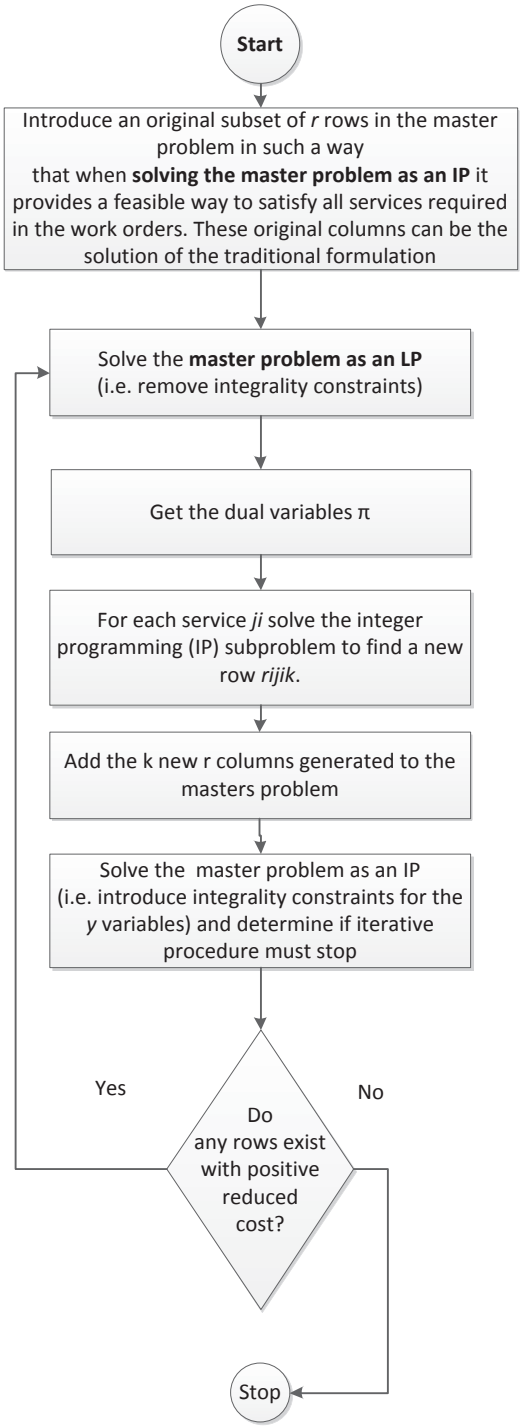

Figure 1. Flowchart of the column generation process.

The Table 6 contains the decision variables (yellow) and the total cost parameters. The multiplication of these two vectors will provide the objective function value named $Grand\_Score_{colgen}$.

Table 6. Components for objective function value computation

Supplier 1			Supplier 2		Supplier 3	
Way 1	Way 2	Way 3	Way 1	Way 2	Way 1	Way 2
Y_{11}=	Y_{12}=	Y_{13}=	Y_{21}=	Y_{22}=	Y_{31}=	Y_{32}=
$Tot\_Score_{1b_1}$	$Tot\_Score_{1b_2}$	$Tot\_Score_{1b_3}$	$Tot\_Score_{2b_1}$	$Tot\_Score_{2b_2}$	$Tot\_Score_{3b_1}$	$Tot\_Score_{3b_2}$

Table 7 shows the possible assignments for each supplier as multiple columns representing different partial solutions. Row constraints ensure that all the row sum of all chosen column in is equal to the right hand side (1).

Table 7. $r_{ij_ik^{(b_k)}}$ rows and right hand side

	Supplier 1			Supplier 2		⋯	Supplier *k*		
	Way_1	⋯	Way_b	Way_1	Way_2	⋯	Way_1	Way_2	
WO1									
S_{11}									=1
S_{21}									=1
S_{31}									=1
WO2									
S_{12}									=1
S_{22}									=1
WO3									
S_{13}									=1
S_{23}									=1
S_{33}									=1
	<=C_1	⋯	<=C_1	<=C_2	<=C_2	⋯	<=C_3	<=C_3	

At each iteration, after solving the master problem as an IP, the total distance traveled between suppliers selected for the services in each work order is calculated (i.e. distance traveled in work order $i = D_i$ assuming a serial execution of services in the work order in ascending order). The distance traveled computation will be very similar for the column generation and for the traditional formulation.

If the distance traveled in a work order *i* exceeds D_max_i the columns selected will not be chosen again simultaneously.

Example: Column 1 in supplier 1 and Column 1 in supplier 2 were selected ($Y_{11} = 1 \; and \; Y_{21} = 1$). If the distance traveled is violated (> D_max_i) in any work order *i* then a cut must be introduced to prevent these columns from being selected simultaneously. The cut for this case is $Y_{11} + Y_{21} \leq 1$.

Table 8 shows the time required (in hours) $\boldsymbol{c_{ij_i}}$ for the different services of the given work orders. These values will act as coefficients to the column assignments to be bound by the vector in Table 9.

Table 8. Expanded d_{ij_ik} matrix

		Full Service Duration Vector				
		Toy Problem	Sup 2	Sup 3	Sup 4	Sup 5
WO_1	S_{11}	$d_{11} = 3$	d_{112}	·	·	d_{115}
	S_{21}	$d_{12} = 7$	d_{212}	·	·	d_{215}
	S_{31}	$d_{13} = 9$	d_{132}	·	·	d_{135}
WO_2	S_{12}	$d_{21} = 11$	d_{212}	·	·	d_{215}
	S_{22}	$d_{22} = 10$	d_{222}	·	·	d_{225}
WO_3	S_{13}	$d_{31} = 5$	d_{312}	·	·	d_{315}
	S_{23}	$d_{32} = 8$	d_{322}	·	·	d_{325}
	S_{33}	$d_{33} = 3$	d_{332}	·	·	d_{335}

There is now a supplier specific value for service duration. This will aid the algorithm in evaluating individual assignments, based on reduced cost and the capacity expenditure, by reducing symmetry.

Table 9. Available capacity of the suppliers (hr)

Supplier 1	Supplier 2	Supplier 3	Supplier 4	Supplier 5
C_1=17	C_2=13	C_3=12	C_4=19	C_5=17

Row Generation Decomposition Scheme

Model Construction

The column generation approach was originally intended to be applied to a problem matrix with many rows and few columns. Thus, the disaggregated formulation of the LP conferred modest performance benefits when solving medium-sized problems with a supplier-to-service ratio less than 1:1. The quantity of columns that must be generated when problems instances have a high supplier-to-service ratio is computationally prohibitive. However, the decomposition scheme can be transposed to generate rows instead of columns. In this way, the computational complexity is greatly reduced and should be able to handle very large problem instances.

Objective Function

$$\text{Max } Grand\_Score_{rowgen} = \sum_{i=1}^{I} \sum_{j_i=1}^{J_i} \sum_{b=1}^{B} Total\_Score_{i_{j_i}}^{(b_{ij_i})} * Z_{i_{j_i}}^{(b_{ij_i})} \qquad (RGOF)$$

The objective function totals the score of the assignments in the rows that are chosen by the decision variable $Z_{i_{j_i}}^{(b_{ij_i})}$.

Constraints

The dualized convexity constraints RG1 will provide a pricing index, rather than the capacity constraints. Alternatively, services-specific assignment will be declared in the sub-problems.

$$\sum_{i=1}^{I} \sum_{J_i=1}^{J_i} r_{i_{j_i}}^{(b_{ij_i})} * Z_{i_{j_i}}^{(b_{ij_i})} = 1 \qquad \forall \begin{cases} i = 1, \dots, \text{I} \\ j_i = 1, \dots, J_i \end{cases} \qquad \text{(RG1)}$$

Constraints CG2 are constraints on the columns of a particular supplier and will remain in the master problem.

$$\sum_{i=1}^{I}\sum_{j_i=1}^{J_i} r_{i_{j_i}k}^{(b_{ij_i})} * d_{ij_ik} * Z_{i_{j_i}}^{(b_{ij_i})} \leq C_k \qquad \forall \quad k = 1, \ldots . K \qquad \text{(RG2)}$$

By reducing the number of columns that need to be generated (rows), the model is much more efficient. The constraints interact in a similar way as the previous formulation, but produce much more rapid changes in the objective value.

$$\sum_{k}^{K} r_{i_{j_i}k}^{(b_{ij_i})} * Z_{i_{j_i}}^{(b_{ij_i})} = 1 \qquad \forall \begin{cases} i = 1, \ldots, \mathrm{I} \\ j_i = 1, \ldots, J_i \end{cases} \qquad \text{(RG3)}$$

The assignment constraints in each sub-problem allow the row to enter the basis of the master problem with only one assignment. Therefore, there are not incompatible combinations of columns, as in the previous formulation.

$$Z_{i_{j_i}}^{(b_{ij_i})} \triangleq \begin{cases} 1 \text{ if assignment } b \text{ is selected for } j_i \\ 0 \text{ otherwise} \end{cases} \qquad \forall \begin{cases} k = 1, \ldots, K \\ b_k = 1, \ldots, B_k \end{cases} \text{ (sign constraint)}$$

Each row represents an individual assignment. The objective RGOF is to maximize the resulting total score of selected rows such that capacity constraints are not violated.

Master Problem

$$\boldsymbol{Max\ Grand\_Score_{rowgen}} = \sum_{i=1}^{I}\sum_{j_i=1}^{J_i}\sum_{b=1}^{B}\sum_{k=1}^{k} \boldsymbol{score\_matrix_{ij_ik} * r_{i_{j_i}k}^{(b_{ij_i})} * Z_{i_{j_i}}^{(b_{ij_i})}} \quad (\boldsymbol{RGOF})$$

st.

$$\sum_{b_{ij_i}=1}^{B_{ij_i}} \boldsymbol{Z_{i_{j_i}}^{(b_{ij_i})} = 1} \qquad \forall \begin{cases} \boldsymbol{i = 1, \ldots, I} \\ \boldsymbol{j_i = 1, \ldots, J_i} \end{cases} \qquad (\boldsymbol{RG1})$$

$$\sum_{i=1}^{I}\sum_{j_i=1}^{J_i}\sum_{b_{ij_i}=1}^{B_{ij_i}} \boldsymbol{r_{i_{j_i}k}^{(b_{ij_i})} * d_{ij_ik} * Z_{i_{j_i}}^{(b_{ij_i})} \leq C_k} \qquad \forall \quad \boldsymbol{k = 1, \ldots . K} \quad (\boldsymbol{RG2})$$

Note that the assignment constraints and capacity constraints are switched as compared to the original column generation formulation. Therefore, the pricing scheme of the sub-problem is also inverted. Dual variables $\boldsymbol{\pi_{ij_i}}$ come from the convexity constraints (RG1) and are used to price the sub-problem solutions. Dual variables δ_k are now associated with the capacity constraints rather than the assignment constraints.

Sub-Problem

$$\text{Max } Total\_value_{ij_i} = \sum_{i=1}^{I} \sum_{j_i=1}^{J_i} [(score\_matrix_k - \delta_k) * r_k] - \pi_{ij_i} \quad \text{(RGOF)}$$

st.

$$\sum_{\boldsymbol{k}}^{\boldsymbol{K}} \boldsymbol{r_{\langle ij_i \rangle k}} = \mathbf{1} \qquad (\boldsymbol{RG3})$$

$$r_k \triangleq \begin{cases} 1 \text{ if supplier } k \text{ is assigned service } j_i \\ 0 \text{ otherwise} \end{cases} \quad \forall \ \ k = 1, \dots, K \qquad \text{(sign constraints)}$$

Table 10 shows the row generation scheme in the master problem.

Table 10. Transposed disaggregation scheme

Row	WO1	Serv	Sup 1	Sup 2	Sup 3	Sup 4	Sup 5	RHS
$y_{11}^{(1)}$	WO_1	S_{11}						=1
$y_{11}^{(2)}$		S_{11}						=1
.		.						.
$y_{11}^{(B_1)}$		S_{11}						=1
$y_{12}^{(2)}$		S_{12}						=1
$y_{13}^{(1)}$		S_{13}						=1
$y_{21}^{(1)}$	WO_2	S_{21}						=1
$y_{22}^{(1)}$		S_{22}						=1
$y_{31}^{(1)}$	WO_3	S_{31}						=1
$y_{32}^{(1)}$		S_{32}						=1
$y_{133}^{(1)}$		S_{33}						=1
			<=C_1	<=C_2	<=C_3	<=C_4	<=C_5	

IV. ANALYSIS OF RESULTS

Instance Data

This study differs from the typical OR study in that the model parameters are not samples collected from a population, but rather are randomly generated. In order to develop an analytical tool for a process which does not yet exist, samples must be simulated to reflect the best prediction of characteristics a real population is expected to have. Information will be collected from soliciting suppliers describing their manufacturing service capabilities. We will make assumptions based on general knowledge of manufacturing industries. It is only necessary for the data to exhibit broad tendencies of the population that will ultimately be sampled. Realistically, a properly formulated and programmed optimization model is expected to handle both real and fabricated data in much the same way. The information collected at this phase of research will illuminate areas of interest for future work that can be studied with descriptive models and optimized with analytical models.

Analysis of the traditional LP model results shows that the formulation is consistent in its processing of the five replicates of each size of the problem. Figure 2 illustrates that while larger problems take longer to compute, replicates tend to take the same amount of time.

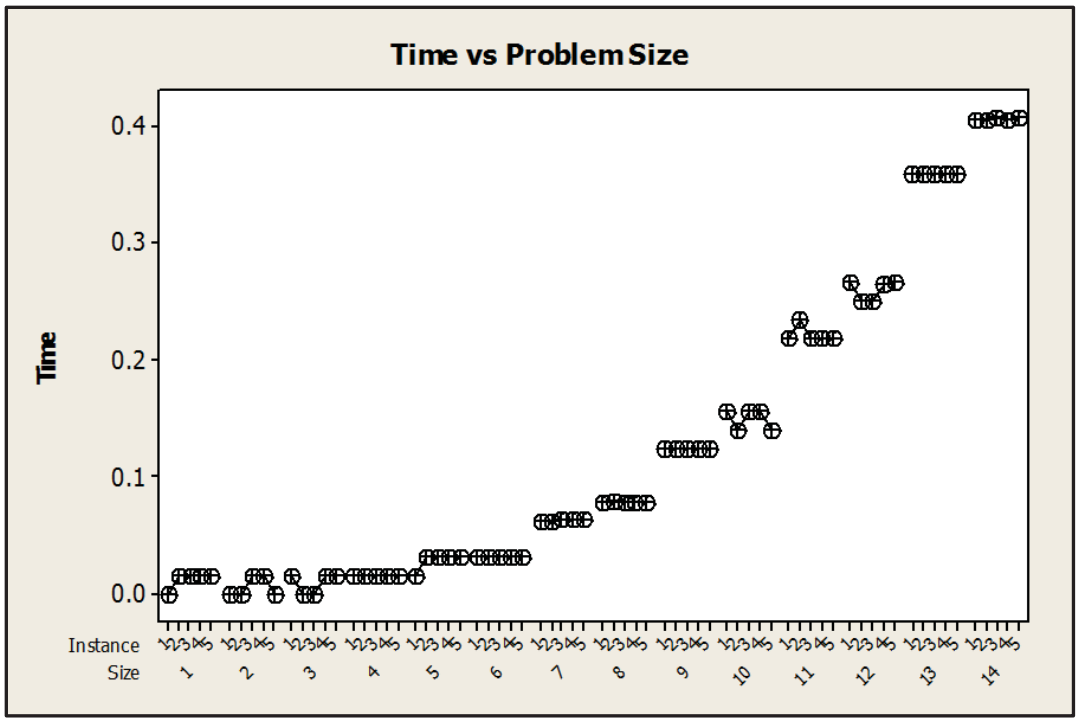

Figure 2. Boxplot of time vs. problem size vs. instance.

In table 11 a p-value of 0.488 for problem instance leads us to fail to reject the hypothesis that different problems of a similar size will take the same amount of time to compute solutions for. Each instance has a different number of active services, and different data populating the input matrices.

Table 11. ANOVA results for traditional formulation

Two-way ANOVA: Time versus Size, Instance					
Source	DF	SS	MS	F	P
Size	13	1.19214	0.0917030	2922.78	0.000
Instance	4	0.00011	0.0000273	0.87	0.488
Error	52	0.00163	0.0000314		
Total	69	1.19388			
S = 0.005601		R-Sq = 99.86%		R-Sq(adj) = 99.82%	

This result allows us to conclude that the computation method used in the tradition formulation, namely the simplex algorithm, is consistent. Each replicate instance

is also repeated three times. Each time a problem is rerun the solution agrees with the established global optimum.

Similarity Score

Although the data is randomly generated, only one variable is truly random. The similarity score is a random number between 0 and 1. In practice, the optimization model will receive scores from an automated matching system which reflect the similarity between a service request and a service solicitation, described in MSDL, and defined by the matching algorithms. A low matching score means that the subjects are dissimilar, while a high score means that a match is possible. In order to maintain solution validity, the matching algorithm must be able to distinguish possible matches from the rest. The similarity score data should be post-processed in such a way that incompatible matches are identified such that the distinction is passed to the optimization model. It is envisioned that the agile supply chain design platform will be able model and control certain costs. By constraining the model to yield a solution with lower costs, the total score is also reduced. It is necessary then that the optimization model be prohibited from providing inexecutable/infeasible assignments. This could be accomplished via threshold data or by assigning zero values to impossible matches.

Real samples of suppliers will have tools and processes that are interrelated. This is because multiple tools and processes are usually required to produce a single part or product. For these same reasons work order queries will also demonstrate the similar patterns. This means that if a supplier can provide one service in a work order, they are likely to be capable of performing another. These distinctions vary greatly over manufacturing industry, technology requirement, resource consumption and many other

variables. This area of study is highly nuanced, and will be recommended as future work. This study will allow the scores to be completely random. If an optimization model can produce high-scoring supply chain with random data, it will likely perform better with real data.

Supplier Capacity

Capacity and service duration are generic units. They may take on any value that fits the description of a service provision or work order, e.g., unit/s, part/min, hr/ batch, days/build. The supplier pool is expected to be so large that there is enough excess capacity to fulfill work order requests. Optimization model performance is tested at multiple levels of capacity scarcity. Individual supplier capacity varys over 25% and 50% in excess of the total capacity required. Otherwise, the solution would be infeasible without adequate capacity provision.

Service Duration

The amount of time required to perform each service will realistically vary depending on which supplier performs it. The simulated data represents up to a 50% difference between suppliers. There are too many similarities in process requirements to allow for a larger disparity. Notwithstanding the proposition of a general range of deviation for this metric, certain processes will surely display disparities in resource requirement characteristic of their practice or practitioner. For example, an automated manufacturing system can inspect parts thousands of times faster than a human. It is often a matter of technology, and such distinction would be incorporated into the MSDL descriptions. A customer expressing a need for automated inspecting capabilities would not be matched with a supplier without one. In this respect, the degree of variation

between processing times of truly similar services will be minimal. We will allow service duration to take on any number between 1 and 13. This limited range will allow us to observe the effect of capacity scarcity on model performance. Each service is assigned an average duration. Each supplier will perform the service within ±25% of this norm.

The service duration input was changed from two dimensional to three dimensional to reduce symmetry and maintain tractability for large scale optimization. Table 12 shows the comparison of both methods of input and insignificantly different results.

Table 12. ANOVA 2D vs. 3D duration array

Two-way ANOVA: Time versus Duration dimension, Size

Source	DF	SS	MS	F	P
Duration dimension	1	0.00009	0.000091	1.05	0.307
Problem Size	13	2.55298	0.196383	2263.32	0.00
Interaction	13	0.00084	0.000064	0.74	0.719
Error	112	0.00972	0.000087		
Total	139	2.56363			

S = 0.009315 R-Sq = 99.62% R-Sq(adj) = 99.53%

With a p-value of 0.307 it is inferred that these relatively small sizes are not large enough to exhibit the detrimental effects of symmetry on computation time.

Services per Work Order

Each work order will have a different number of services. The maximum number of services per work order is dictated by problem size. The number of services is allowed to take on any number between 66% of the maximum and the maximum. This prevents the overlap of problem size and allows the effects of interaction with other performance metrics to be more easily discerned.

Distance Threshold

Each work order will be subjected to a maximum distance threshold. Realistic definition of threshold values would not be entirely arbitrary. The purpose of the thresholds is to reduce distance traveled in each work order. Arbitrary values for each work order based on a certain percentage of maximum possible distance based on the number of services. However, this approach would not allow for direct observation of the interaction between distance constraints and performance. Therefore, a descriptive model can has been programmed to reduce the total distance to a predetermined percentage by incrementally lowering *D_max* values and resolving the problem until it is reached, while only allowing total distance to vary a little between work orders. The problems are then solved with the new *D_max* values. This ensures a more equitable assignment of distance threshold values. Moreover, multiple levels per problem size can be recorded and tested again, to observe the distinction between a global implicit distance constraint and multiple explicit work order distance constraints.

Suppler/Service Ratio

The observed ratio of suppliers to services is kept a 10:1 to realize the full benefit of column generation. We will treat this as a minimum since one can easily imagine cases of common service requests which could be fulfilled by a large pool of suppliers. A branch-and-price model has been developed for cases where only a very small number of suppliers are eligible at a certain stage in the matchmaking process.

Traditional Formulation Without Distance Constraints

To confirm the validity of the traditional LP formulation the small toy problem was modeled in Microsoft Excel. The Excel solver is proven to be effective for small,

simple problems. This toy problem with 125 decision variables is well within it capability. Table 13 shows the problem input table in excel.

Table 13. Parameters provided to Excel spreadsheet

hr/service required	Scores for service/supplier matches						
			S1	S2	S3	S4	S5
3	Work order 1	Service 1	0.64	0.32	0.50	0.43	0.71
7		Service 2	0.23	0.63	0.95	0.58	0.01
9		Service 3	0.15	0.56	0.40	0.42	0.54
11	Work order 2	Service 1	0.11	0.32	0.38	0.67	0.89
10		Service 2	0.55	0.67	0.34	0.78	0.29
0		Service 3	0.00	0.00	0.00	0.00	0.00
5	Work order 3	Service 1	0.39	0.58	0.22	0.33	0.24
8		Service 2	0.41	0.08	0.92	0.62	0.78
3		Service 3	0.29	0.45	0.34	0.07	0.11

The yellow area in Table 14 shows the decision variable matrix after the solver has been run. The blue numbers indicate variables which are constrained.

Table 14. Solution provided by Excel Solver Solution

Assignment Matrix - Optimal Answer								
		S1	S2	S3	S4	S5	Total	Required
Work order 1	Service 1	0.00	0.00	0.00	0.00	1.00	1.00	1
	Service 2	0.00	0.00	1.00	0.00	0.00	1.00	1
	Service 3	0.00	1.00	0.00	0.00	0.00	1.00	1
Work order 2	Service 1	0.00	0.00	0.00	0.00	1.00	1.00	1
	Service 2	0.00	0.00	0.00	1.00	0.00	1.00	1
	Service 3	0.00	0.00	0.00	0.00	0.00	0.00	0
Work order 3	Service 1	1.00	0.00	0.00	0.00	0.00	1.00	1
	Service 2	0.00	0.00	0.00	1.00	0.00	1.00	1
	Service 3	0.00	1.00	0.00	0.00	0.00	1.00	1
Total capacity (hr)		5.00	12.00	7.00	18.00	14.00	Total_score	
Supplier capacity		17.00	13.00	12.00	19.00	17.00	$5.35	

The formulas within certain cells represent the linear equations that make up the objective function, parameters and constraints. The Excel solver uses linear programming to solve the system of equations by assigning a binary value to each decision variable.

Appendix shows the input/output method used for the XpressIVE model. Table 15 from the Xpress output shows the same solution provided by Excel solver.

Table 15. Solution provided by XpressIVE

		SUPPLIER				
		s1	s2	s3	s4	s5
WO1	sv1	0	0	0	0	1
WO1	sv2	0	0	1	0	0
WO1	sv3	0	1	0	0	0
WO2	sv1	0	0	0	0	1
WO2	sv2	0	0	0	1	0
WO2	sv3	0	0	0	0	0
WO3	sv1	1	0	0	0	0
WO3	sv2	0	0	0	1	0
WO3	sv3	0	1	0	0	0

Parity is maintained as problems of different sizes are compared. Therefore, we concluded that the LP Mosel formulation accurately reflects the theoretical model.

Traditional Formulation with Distance Constraints

The quadratically constrained NLP model initially outperforms the dynamic programming model with respect to computation time. The NLP model produces exact optimal solutions up to 75 suppliers and 22,500 decision variables. However, the problem becomes intractable when service durations are supplier specific. The increase in variables with the addition of the 3rd dimension causes the global search to identify an optimal solution at the upper bound without satisfying the all distance constraints. The objective value of the infeasible solution is often quite close (as small as 2.7/1000 of 1%) to that obtained by performing cuts. The NLP model has been programmed to produce pseudo-optimal solutions by dynamically reducing the D_max thresholds until the overall distance savings exceeds a certain percentage. However, this general approach is not aligned with future objectives of an explicit formulation of the agile supply chain design

optimization model which considers actual distance cost. Still, the nonlinear distance constraint may be fruitful. The Xpress Optimization Suite is equipped with a solver this specific purpose. Successive Linear Programming (SLP) is a process of making linear approximations of the original problem and solving the approximations. SLP is a complicated programming methodology which needs to be studied in depth, and is beyond the scope of this research.

The results confirm that the quadratic formulation of the global distance constraint functions as intended. This constraint can be reformulated and incorporated into the column generation model. The decomposition of the problem should allow the constraint to maintain functionality on a much larger scale.

Traditional Formulation with Cuts

Constraining for distance is an effort to control costs. The more the model exhibit lower performance the more it is constrained. Because computational resources will become more abundant as the analytic tool matures, we are primarily concerned with the behavior of the MSDL similarity scores of the assignments.

Toy Problem

The toy problem was directed to cut 50% of the work order distance. Table 16 shows that the cut solution is only 0.16 less than the optimal solution.

Table 16. XpressIVE results for toy problem cuts

```
                SUPPLIER

           1     2     3     4     5

1 sv1      0     0     0     0     1
1 sv2      0     0     1     0     0
1 sv3      0     0     0     1     0
2 sv1      0     0     0     0     1
2 sv2      0     0     0     1     0
2 sv3      0     0     0     0     0
3 sv1      0     1     0     0     0
3 sv2      1     0     0     0     0
3 sv3      0     1     0     0     0

(0.341 sec) Optimal solution: 5.19
beginning distance: 298
ending distance:   134
distance reduction: 55.0336%
```

Descriptive Statistics

A descriptive LP model was initially programmed to incrementally reduce the distance threshold of each work order until the total distance reaches a desired level of reduction. Six levels of distance reduction at 5%, 10%, 25%, 37.5%, 50% and 75% of total distance in the optimum solution are tested. Table 17 shows the details of the problem sizes.

Table 17. Problem instances

Problem Size	Work Orders	Services	Suppliers	Decision Variables
1	5	5	25	625
2	5	10	25	1250
3	10	10	50	5000
4	10	15	50	7500
5	15	15	75	16875
6	15	20	75	22500
7	20	20	100	40000
8	20	25	100	50000
9	25	25	125	78125
10	25	30	125	93750
11	30	30	150	135000
12	30	35	150	157500
13	35	35	175	214375
14	35	40	175	245000

Figure 3 shows the effect that the distance constraints have on average score.

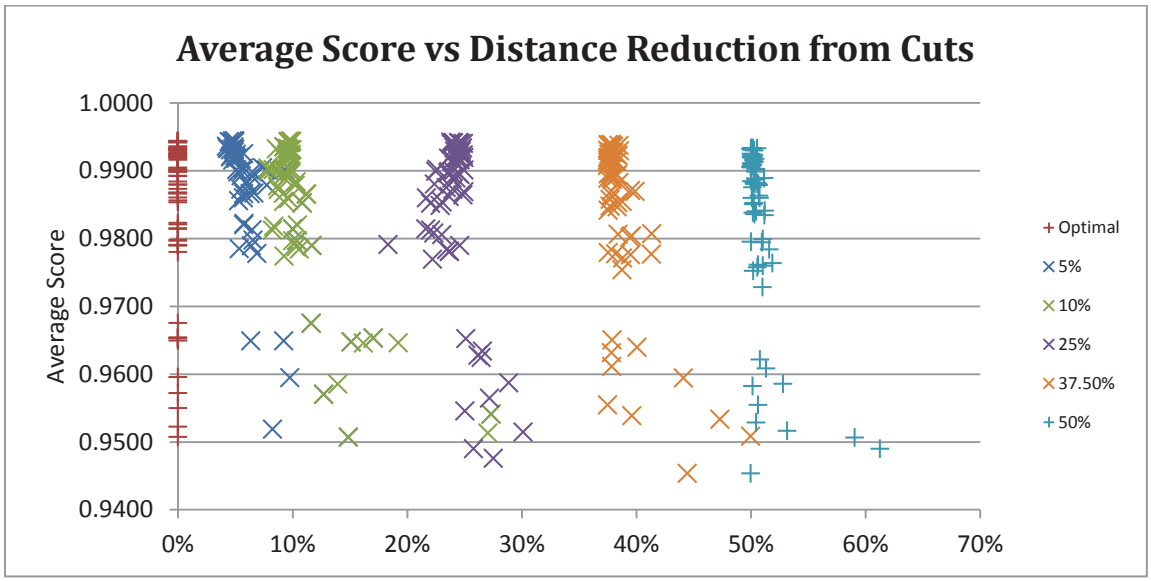

Figure 3. The six cut levels and their effect on average score.

After work order distance thresholds are identified that meet each goal, the models are run again, in random order, with the explicit D_max values and solutions are

quickly identified after the necessary cuts are applied. No solutions subject to 75% distance reduction are feasible due to insufficient capacity. Figure 4 shows the same data broken down into the seven levels of suppliers. We can clearly see that the poor data on the bottom half of the chart is made up of only the smallest size of 25 suppliers. This is much smaller than is expected in the web-based platform application. We can therefore conclude that results from supplier pools lager than 175 are expected to contain assignments of with at least a 0.99 similarity score when subject to cuts of up to 50% of optimum distance.

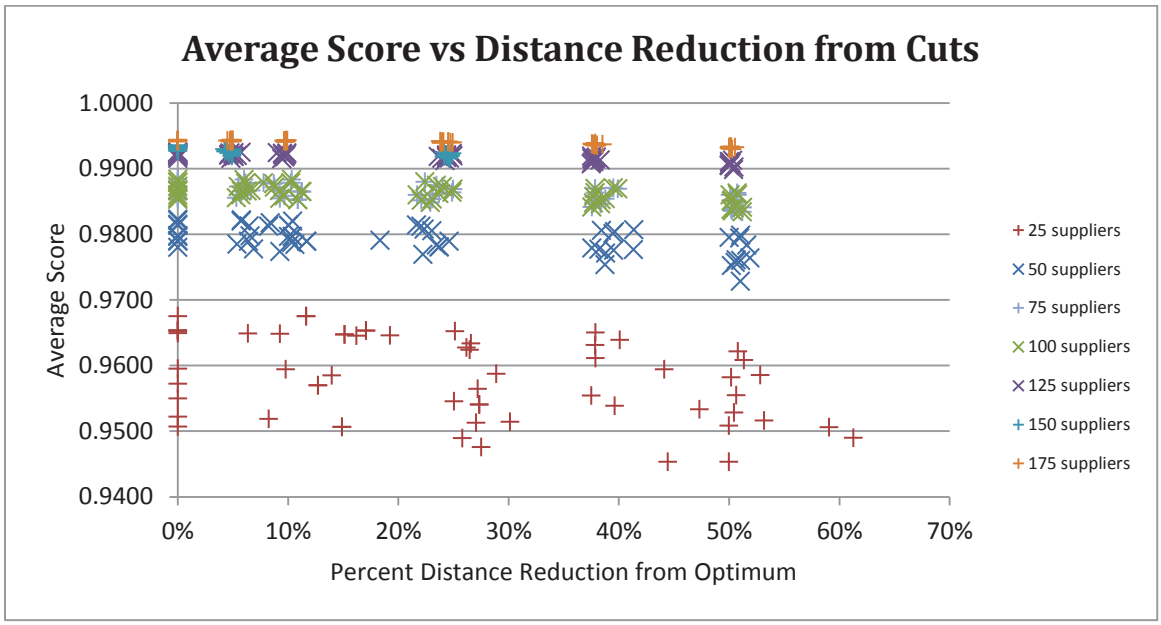

Figure 4. Categorization of cuts values by number of suppliers.

This data can yet be further analyzed. It is clear from Figure 5 that if we compare the scores of the cut to the original optima, that the scores converge as the problem size increases. We can conclude that the model can provide better scores when drawing from a larger pool of suppliers.

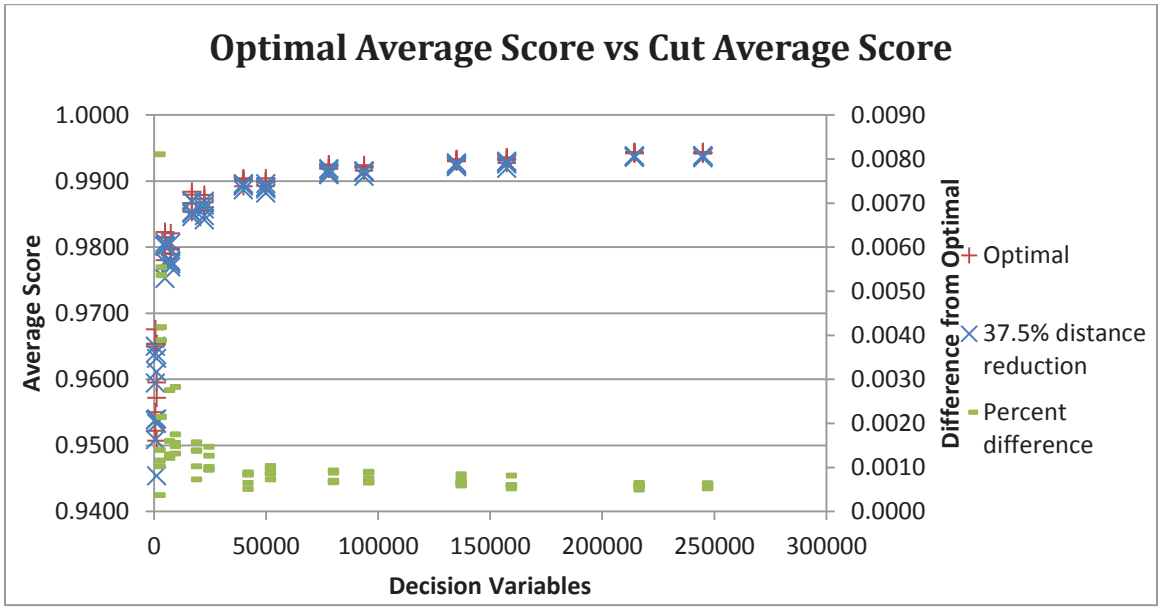

Figure 5. Average optimum and cut score (top left) and difference (bottom right).

Finally, when the average score values are grouped by number of suppliers, a clearer picture begins to immerge. If we assume a hypothetical situation with real data, choosing from at least 100 suppliers, which is a reasonably safe assumption, an assumed cut of 37.5% distance can be made while suffering only a tenth of point penalty. Figure 6 shows that sizes of 175 suppliers or more can make the same cut for half that cost in score reduction.

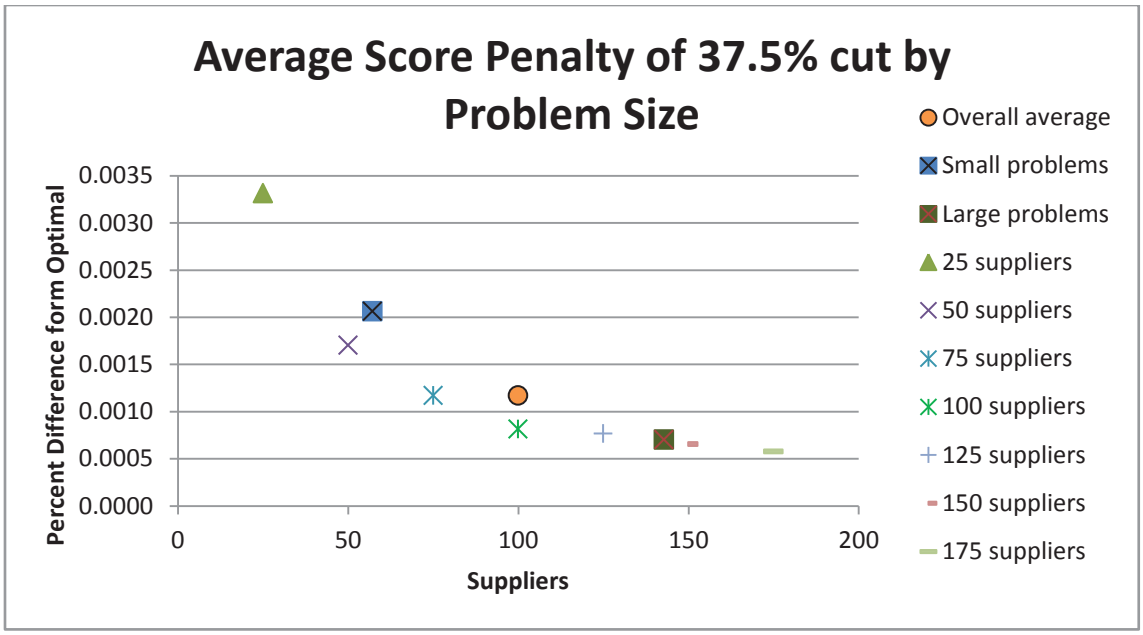

Figure 6. Average penalties of 37.5% cuts grouped by number of suppliers.

Again, in figure 7 we see a more severe penalty exacted for cutting up to 50% of maximum distance. This still only amounts to five points. This number will be lower when real data is handled because of manufacturer specialization.

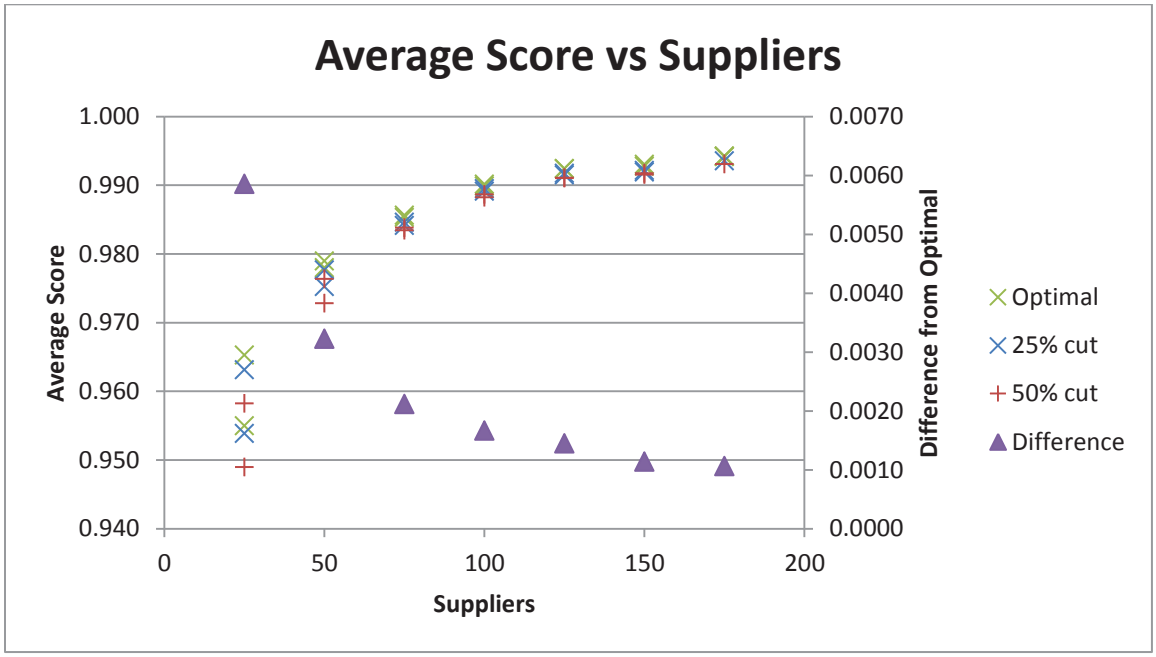

Figure 7. Two levels of cuts and optimal solutions.

In Figure 8 the cuts begin to multiply the required time by a factor of 10.

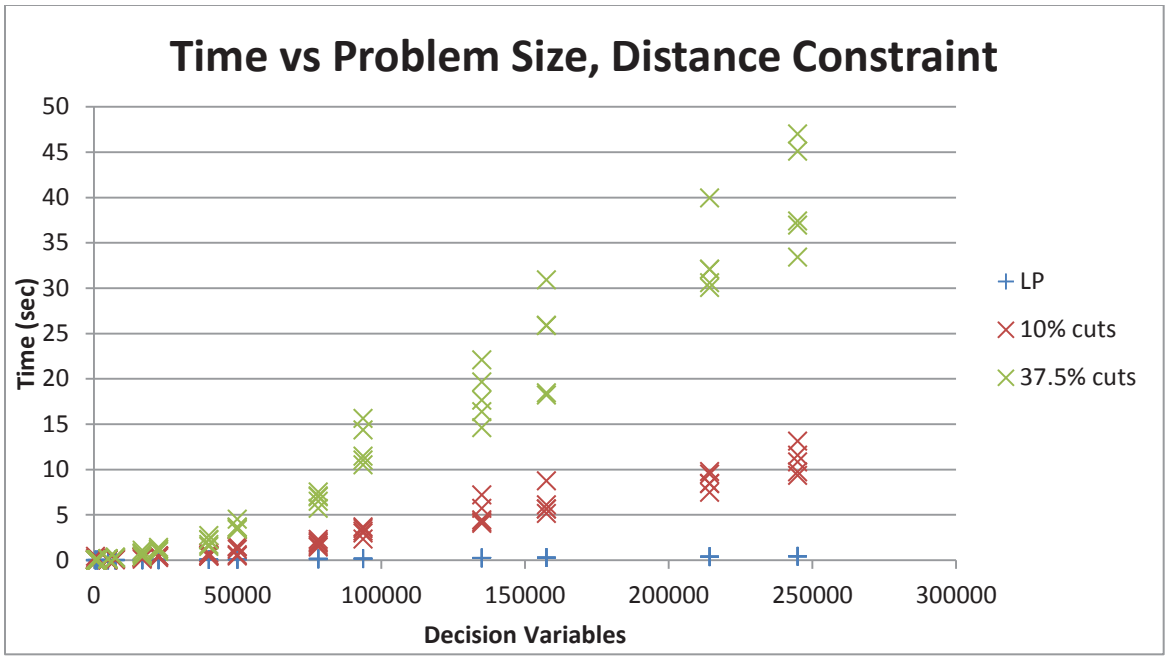

Figure 8. Computation time required to perform cuts at two levels.

These results are not too bad for the small samples. However, performing cuts becomes cumbersome when tens of thousands are required, as they are for the largest size problem. Cuts in the column generation formulation are expected to perform better because there are less columns combinations to cut than supplier combinations.

Larger ratios of suppliers to work orders were also tested. The results showed an improvement in processing times of all models. Similar results were seen when the supplier capacity was doubled and quadrupled. This is congruent with the theoretical predictions of abundant capacity. Alternatively, scarcity of capacity makes the problem much more difficult to solve. Therefore, these are the results that illustrate the efficacy of our models.

Column Generation

The generalized assignment problem is NP-hard, and many approaches have been developed to solve it either approximately or exactly. Yet the difficulty in modeling the general form the GAP is not experienced in the agile supply chain design problem. Typical academic work modeling the GAP assumes scarce capacity. The larger the competition of tasks/services for the limited resource, the more difficult it is to solve. This degree of paucity cannot be experienced in a marketplace environment. Even if individual suppliers were very conservative with their solicitation, the collective capacity of the entire system would be abundant. This parameter allows us to exploit the simple avenues of solving the problem. The simplex method is a powerful algorithm that is able to solve this problem very quickly.

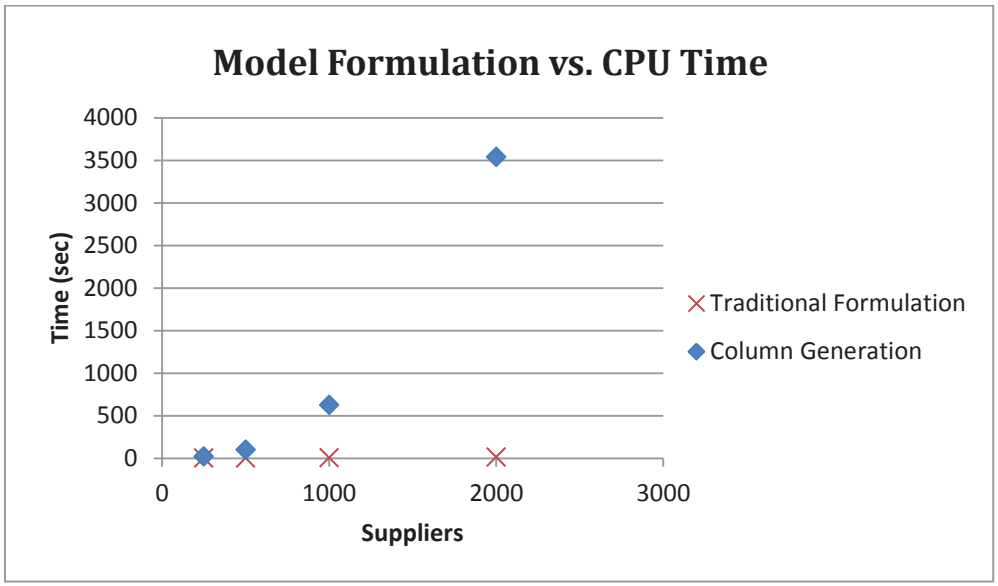

Figure 9. Column generation performance before reformulation.

Despite the known advantages of column generation, the traditional model exhibits exceptional performance up to limited size (5,000 suppliers in this case). These results are indicative of the weak nature of the dual variables associated with the LP

relaxation. An upper bound is essentially constructed, in a maximization problem, by dualizing the constraints to be used in the objective function. A heuristic was also used to speed up this process by providing an approximation, preventing the solver from resorting to lengthy computations, unaided by confounded dual variables. Heuristic approximations allow the optimal solution to be found in a shorter amount of time.

The typical formulation of a column generation model for the generalized assignment problem has suppliers in columns and services in rows. This approach is most efficient when the service to supplier ratio (*n/m*) is 10:1 or larger, when the problem matrix has more rows than columns (Ogtildeuz, 2002). The Lagrangian duals values are less confounded and more effective when there are many assignment constraints containing fewer column elements in the LP relaxation. Consider the four problem sizes below. By the same reasoning, the typical column generation approach is computationally prohibitive when choosing from a large pool of suppliers and the *n/m* ratio is very small. For problem size 1 the model will have to generate multiple columns for each of the 250 suppliers. It would be more efficient to generate rows for each of the 25 services. Table 18 shows the performance improvements associated with transposing the decomposition scheme.

Table 18. Column Generation Traditional and Transposed

	Column Generation Data			CPU Time	
Problem Size	WOs	Services	Suppliers	Original	Transposed
1	5	5	250	18.597	0.312
2	10	5	500	98.483	0.656
3	10	10	1000	625.105	2.067
4	15	10	2000	3538.32	6.436

Figure 10 shows that the comparison between the decomposition schemes resembles the gap between the performance of the traditional model and the first column generation formulation.

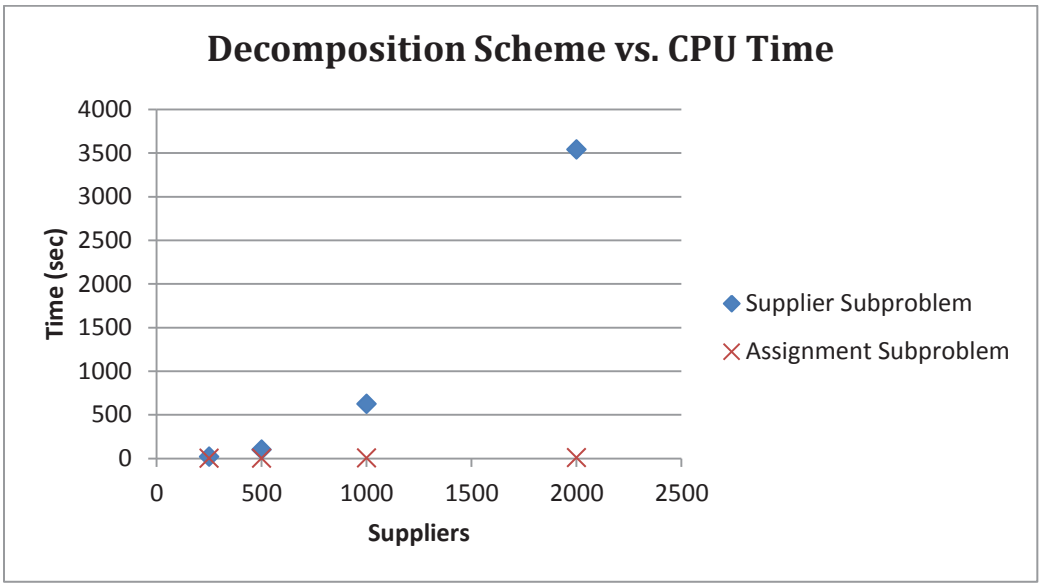

Figure 10. Computation reduction from decomposition reformulation.

The Dantzig–Wolfe decomposition is meant to break down a problem into components that model the problem in a sensible way. In the generalized assignment problem the columns represent individual supplier assignments. Therefore, the capacity constraints are modeled in the knapsack problem and any columns that are generated to enter the master problem are feasible in that sense. Although the assignment sub-problem of the new formulation is not technically a knapsack problem because the capacity constraints remains in the master problem, it is still influenced by dual variables from the capacity constraints and the rows entering the basis are priced using the convexity constraints.

While the simplex method had proven to be very efficient, column generation with the assignment sub-problem is still faster. It is unclear if exact heuristics would be

faster, but they may allow for larger problems. Look at the tradeoff between the traditional formulation and the new column generation formulation.

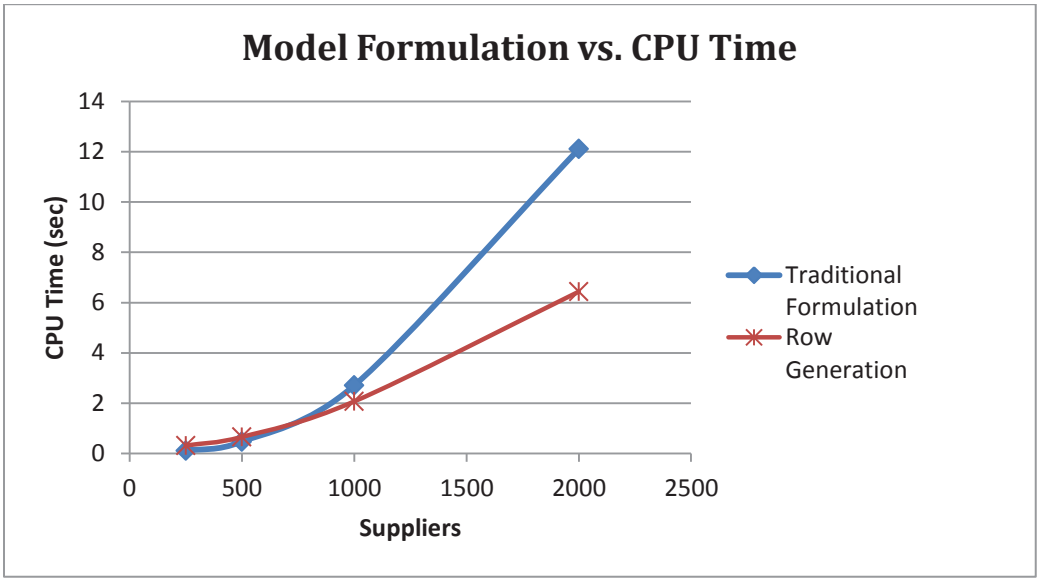

Figure 11. Changes observed in performance between models.

Instances with different levels of maximum services in work orders have been tested. Table 19 shows ten problem instances with similar numbers of services. Each problem size has five replicates. All runs were randomized in blocks and each replicate was repeated three times.

Table 19. Instances with similar work order size

Problem Size	Work Orders	Services	Suppliers	Assignment Variables
1	5	10	500	25000
2	10	10	1000	100000
3	15	10	1500	225000
4	20	10	2000	400000
5	25	10	2500	625000
6	30	10	3000	900000
7	35	10	3500	1225000
8	40	10	4000	1600000
9	45	10	4500	2025000
10	50	10	5000	2500000

The decomposed row generation formulation produces the same solutions as the traditional model in less time. Figure 12 shows the gap between the two models grows quickly as problem size increases.

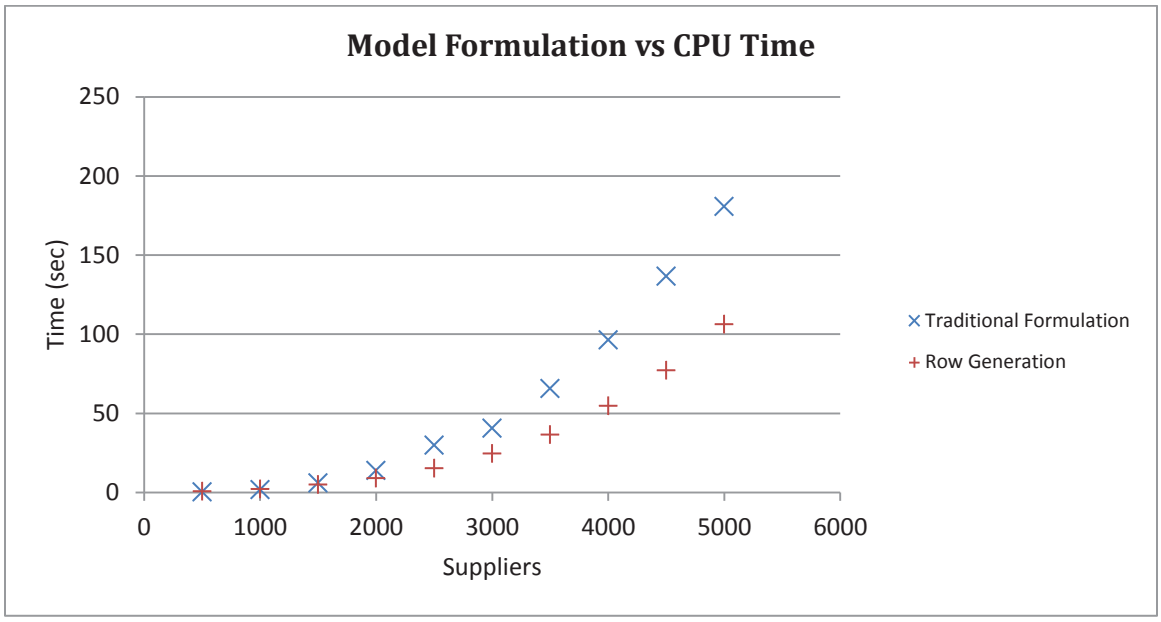

Figure 12. Computation time for ten sizes of each model formulation.

While these results do not appear to be a significant improvement, it is important to note that the linear program is limited to problems of 5,000 suppliers and 2.5 mil assignment variables with these computational resources. A problem with 100 work orders of 10 services each, and 10,000 suppliers can be solved in 17 min 14 sec. A problem with 12,500 suppliers will take about 26 min to solve. The computer lacks sufficient memory to generate larger data sets than this. The performance can be improved by running the sub-models concurrently.

The full details of the instances are displayed in Table 20. 10 sizes of 5 replicates each were run in randomized order.

Table 20. Raw Data from traditional model and row generation

INPUT						OUTPUT			
		Set Size			Prob Size	Traditional Formulation		Row Generation	
Size	Instance	WO	Services	Suppliers	No. vars	Time (sec)	objective	Time (sec)	objective
1	1	5	10	500	25000	0.249	38.93323	0.655	38.93323
	2				25000	0.256	39.9319	0.499	39.9319
	3				25000	0.312	45.90898	0.624	45.90898
	4				25000	0.479	38.93323	0.515	38.93323
	5				25000	0.504	42.92369	0.686	42.92369
2	1	10	10	1000	100000	1.718	78.92174	2.105	78.92174
	2				100000	1.673	76.92379	1.809	76.92379
	3				100000	1.78	82.91943	1.902	82.91943
	4				100000	1.828	83.91909	2.043	83.91909
	5				100000	2.807	82.91943	1.934	82.91943
3	1	15	10	1500	225000	5.858	125.9108	4.944	125.9108
	2				225000	5.329	119.9137	4.303	119.9137
	3				225000	5.689	121.9132	3.945	121.9132
	4				225000	5.797	127.9081	4.57	127.9081
	5				225000	5.435	123.9121	4.383	123.9121
4	1	20	10	2000	400000	13.768	169.9145	8.876	169.9145
	2				400000	13.392	165.9163	8.469	165.9163
	3				400000	14.121	171.9138	8.781	171.9138
	4				400000	12.849	159.92	8.579	159.92
	5				400000	12.971	164.9178	8.486	164.9178
5	1	25	10	2500	625000	29.749	204.9168	15.272	204.9168
	2				625000	24.306	211.9142	15.553	211.9142
	3				625000	25.227	210.9146	15.35	210.9146
	4				625000	23.67	204.9168	14.929	204.9168
	5				625000	24.113	209.9148	15.349	209.9148
6	1	30	10	3000	900000	40.462	245.9219	24.617	245.9219
	2				900000	48.79	240.924	24.039	240.924
	3				900000	41.883	242.9225	24.211	242.9225
	4				900000	50.111	252.9204	24.819	252.9204
	5				900000	41.688	252.9204	25.037	252.9204
7	1	35	10	3500	1225000	65.648	276.9141	36.534	276.9141
	2				1225000	64.727	296.9077	38.906	296.9077
	3				1225000	77.34	288.9103	37.751	288.9103
	4				1225000	64.867	297.9066	38.766	297.9066
	5				1225000	67.637	286.9106	37.564	286.9106

Table 20 continued

INPUT						OUTPUT			
		Set Size			Prob Size	Traditional Formulation		Row Generation	
Size	Instance	WO	Services	Suppliers	No. vars	Time (sec)	objective	Time (sec)	objective
	1				1600000	96.241	327.9195	54.678	327.9195
	2				1600000	104.588	341.9178	56.69	341.9178
8	3	40	10	4000	1600000	109.395	325.9207	54.304	325.9207
	4				1600000	98.84	331.9191	54.787	331.9191
	5				1600000	101.914	335.9188	55.723	335.9188
	1				2025000	136.57	373.9103	77.142	373.9103
	2				2025000	135.382	360.9132	75.114	360.9132
9	3	45	10	4500	2025000	134.741	370.9108	76.612	370.9108
	4				2025000	141.341	375.9098	77.813	375.9098
	5				2025000	138.193	374.9099	77.734	374.9099
	1				2500000	180.727	418.9161	106.314	418.9161
	2				2500000	177.513	401.919	101.244	401.919
10	3	50	10	5000	2500000	184.104	412.9168	104.301	412.9168
	4				2500000	188.176	410.9173	103.787	410.9173
	5				2500000	193.575	421.9157	105.924	421.9157

V. CONCLUSION

This study represents the preliminary phase of an entire line of research that will move forward based on the descriptive information that can be gleaned from these results. The technical requirements of the optimization models developed here are directly related to the descriptive characteristics of the instance data. The conclusions that can be drawn from these results regarding the effectiveness of the models when processing different data/ real data are limited. The formulas for generating similarity score, service duration and supplier capacity were constructed with simple assumptions. The optimization models handled the resulting data quite well. We know when the supplier-to-service ratio is low, as typically modeled in the generalized assignment problem, that the problem would be very difficult to solve. For example, if the supplier-to-service ratio was 1:10 instead of 10:1, the computational resources needed to solve a problem instance with 1,000 suppliers with the traditional linear program would exceed those used in this research. A traditional branch-and-price model was developed to handle special cases of the problem where the supplier-to-service ratio is small. The results resembled those shown in the literature. However, branch-and-price and other methods for solving the GAP are devised to address issues that do not exist when the problem is formulated for the DMM platform.

By decomposing the LP formulation and taking a column generation approach, a transition from a computationally prohibitive methodology to a fully scalable model that maintains functionality at very large sizes, we have fulfilled the principal objective of this

study. Although the traditional formulation and column generation models do not directly control distance, they are effective nonetheless. The abundance of computational resources in the near future will extend the practical efficacy of the models, even if they are not reformulated.

In order to fully understand the nature of the supplier selection problem as formulated specifically for the DMM platform, a number of descriptive models were needed. To conduct the study, functional programs were also developed to interact with the data generation, descriptive and optimization models. It is clear that the IT requirements of the OR branch of the DMM are much more than optimization models.

The most significant result that has been observed is that when problem size is expanded to reflect the true size of the manufacturing services industry, it can be heavily constrained while still producing good results. If we consider the trend expected to appear in the real samples of suppliers with multiple interrelated scores, it should allow for even greater control over distance while maintaining the collective value of assignments.

This study has addressed three concerns regarding the unique formulation of the supplier selection problem for DMM. First, the column generation approach resolved the issue of limited scalability of the traditional LP formulation. Second, distance inside the supply chain can be reduced without significantly impacting the objective score. Finally, these results show that a properly devised analytical model can perform supplier evaluation and selection based on similarity score data derived from MSDL descriptions, allowing it to provide solutions that neatly fulfill the needs of the agile supply chain deployment problem.

Future Work

Simulated Data

The characteristics of the actual environment intended to be modeled should be researched in depth. Collecting real data samples was beyond the scope of this research and was not possible in some senses. In the future, data should be collected when possible. Otherwise, formulas for generating the data need to be improved.

Descriptive models

Properly formulated model assumptions and accurate data are required to construct a valid model. Furthermore, model validity must be assessed and maintained in order to experience the full benefits of optimal decision implementation. Complex simulation models will need to be deployed to fulfill this end. The descriptive model for determining the distance thresholds based on general percentages of optimal distance should continue to be refined. Moreover, it is likely the model could work backwards from customer-provided specifications to determine the maximum limit to which the model can be bounded while still meeting the minimum requirements. Cost control using the set of constraints will likely be exercised through the direct control over one parameter of choice. As model complexity grows and the parameters are made more explicit, descriptive models will become vital to the fine tuning and optimum performance of each optimization model.

Deployment

The optimization models were programmed in the Xpress interactive visual development environment (IVE). Although Mosel can be implemented as a standalone application, the modeling environment can also be extended into C, C++, Fortran, Java,

VB6 and .NET programming interfaces for practical application of the solver engines. No loss of functionality occurs when utilizing the modeling environment through callable libraries in other programming interfaces. This allows the models to be refined or reformulated without using Mosel's native GUI. Future optimization tools will need to be directly integrated into the architecture of the DMM for maximum effectiveness.

Database

The preliminary programming has been implemented to use Microsoft Excel as a database interface. While it is effective in the short term, it will ultimately need to be unified with the programming interface that will be chosen to run the entire MSDL/web-based platform. The established methodology and code for allowing the models to interact with Excel easily converts to any of the available choices for database systems.

Reformulation of the Column Generation Model

The Lagrangian duals from the LP relaxation are used to weight the decision variable coefficients of the knapsack sub-problems. The assignment and convexity constraints are dualized because they have a direct effect on the objective function. The row generation model is limited in its ability to constrain for capacity and distance because those constraints influence the objective through indirect control over the similarity scores associated with the assignments. The model could be reformulated to improve its capabilities in this regard. For example, capacity and distance could be penalized in the objective function to produce new dual variables. Additionally, if columns/rows represented entire work orders, distance could either be constrained within the knapsack problem or by performing column cuts in the master problem. Performing cuts to work order knapsacks resulted in an increase in the scalability over the traditional

formulation with vendor cuts. However, the disaggregated formulation has reduced control over effective capacity usage, when the capacity constraint is declared in the master problem or sub-problem. Although column generation extends the utility of the linear program to larger sizes, the penalties associated with disaggregation are significant. Continued pursuit of solutions to the problem through ILPs may require the model to forfeit robustness. Rather, future work may find nonlinear models better suited for the multi-criteria nature of the agile supply chain configuration problem.

A most promising approach involves column generation as traditionally practiced and initially presented in this paper. The massive numbers of columns that would need to be generated for an agile supply chain configuration problem prevents this approach from performing well, as shown in the results. However, the decomposition scheme can be modified such that individual columns include multiple suppliers. It is unclear how this methodology compares to the untailored formulation, but reducing the number of columns is likely to improve the performance in some cases where the column-to-row ratio is at least 1:10 and the ratio of columns to suppliers-per-columncolumn is ~ 1:1.

SLP Solver for Nonlinear Programs

The principals of a nonlinear implementation of the global distance constraint have proven to be sound. The constraint needs to be scaled up and incorporated as a column generation global distance constraint. Further study of Successive Linear Programming will likely be fruitful, as FICO has emphasized and improved the utility of its SLP solver in recent years. Nonlinear programming may even yield better results than the established method of cuts, as the small sized problems demonstrated.

APPENDIX A

XpressMP Input/Output for Traditional Formulation

Toy Problem Data Input File

```
score_matrix:[(1 1 1) 0.64 (1 1 2) 0.32 (1 1 3) 0.50 (1 1 4) 0.43 (1 1 5) 0.71
              (1 2 1) 0.23 (1 2 2) 0.63 (1 2 3) 0.95 (1 2 4) 0.58 (1 2 5) 0.01
              (1 3 1) 0.15 (1 3 2) 0.56 (1 3 3) 0.40 (1 3 4) 0.42 (1 3 5) 0.54
              (2 1 1) 0.11 (2 1 2) 0.32 (2 1 3) 0.38 (2 1 4) 0.67 (2 1 5) 0.89
              (2 2 1) 0.55 (2 2 2) 0.67 (2 2 3) 0.34 (2 2 4) 0.78 (2 2 5) 0.29
              (2 3 1) 0.01 (2 3 2) 0.02 (2 3 3) 0.03 (2 3 4) 0.04 (2 3 5) 0.05
              (3 1 1) 0.39 (3 1 2) 0.58 (3 1 3) 0.22 (3 1 4) 0.33 (3 1 5) 0.24
              (3 2 1) 0.41 (3 2 2) 0.08 (3 2 3) 0.92 (3 2 4) 0.62 (3 2 5) 0.78
              (3 3 1) 0.29 (3 3 2) 0.45 (3 3 3) 0.34 (3 3 4) 0.07 (3 3 5) 0.11]

capacity: [ (1) 17 (2) 13 (3) 12 (4) 19 (5) 17 ]

duration: [ (1 1) 3   (1 2) 7   (1 3) 9
            (2 1) 11  (2 2) 10  (3 3) 0
            (3 1) 5   (3 2) 8   (3 3) 3 ]

distance_matrix:  [ (1 1) 0  (1 2) 34 (1 3) 45 (1 4) 87 (1 5) 22
                    (2 1) 34 (2 2) 0  (2 3) 89 (2 4) 69 (2 5) 23
                    (3 1) 45 (3 2) 89 (3 3) 0  (3 4) 13 (3 5) 35
                    (4 1) 87 (4 2) 69 (4 3) 13 (4 4) 0  (4 5) 18
                    (5 1) 22 (5 2) 23 (5 3) 35 (5 4) 18 (5 5) 0  ]

valid_serv:  [ (1) 3   (2) 2   (3) 3   ]

D_max:       [ (1) 100 (2) 101 (3) 102 ]
```

Toy Problem Results Output File

```
Begin running model

 The total score (profit) is z = $5.35

 Following is the assignment of SUPPLIERS to services in workorders:

                  SUPPLIER

          s1    s2    s3    s4    s5

WO1 sv1   0     0     0     0     1
WO1 sv2   0     0     1     0     0
WO1 sv3   0     1     0     0     0
WO2 sv1   0     0     0     0     1
WO2 sv2   0     0     0     1     0
WO2 sv3   0     0     0     0     0
WO3 sv1   1     0     0     0     0
WO3 sv2   0     0     0     1     0
WO3 sv3   0     1     0     0     0

 End running model

 ==========================================

 Work Order WO1

 First supplier is s5

 The next supplier is s3

 The distance between Supplier s5 and Supplier s3 is 35

    s5 <--------- 35 ---------> s3

 The next supplier is s2

 The distance between Supplier s3 and Supplier s2 is 89

    s3 <--------- 89 ---------> s2

   The total distance traveled in workorder WO1 is 124

 The maximum distance allowed for workorder WO1 is 100

 Eliminate supplier and rerun problem

 ==========================================

 Work Order WO2

 First supplier is s5
```

```
The next supplier is s4
The distance between Supplier s5 and Supplier s4 is 18
   s5 <--------- 18 ---------> s4
  The total distance traveled in workorder WO2 is 18
The maximum distance allowed for workorder WO2 is 101
==========================================
Work Order WO3
First supplier is s1
The next supplier is s4
The distance between Supplier s1 and Supplier s4 is 87
   s1 <--------- 87 ---------> s4
The next supplier is s2
The distance between Supplier s4 and Supplier s2 is 69
   s4 <--------- 69 ---------> s2
  The total distance traveled in workorder WO3 is 156
The maximum distance allowed for workorder WO3 is 102
Eliminate supplier and rerun problem
```

APPENDIX B

Traditional Formulation Model Code in Mosel

```
model "Supplychain_tradformulation"
 options noimplicit
 uses "mmxprs", "mmsystem"
 uses "mmodbc"  !this is to gain access to the Xpress-Optimizer solver


 parameters
        WO      = 3
        SERV    = 3
        SUP     = 5
        INST    = 1
        DATAFILE=
string("GAP_"+WO+"_workorders_"+SERV+"_services_"+SUP+"_suppliers_"+INST+".dat")
        !OUTFILE= string("result_"+DATAFILE)
 end-parameters

 !declarations section
declarations
        need_to_solve: boolean
        supplis: integer ! number of different suppliers in a work order
        passes: integer !number of times the optimization problem needs to be solved
before finding
        !a solution that satisfies the distance constraint

      ! There are 3 sets: suppliers, services, and workorders
    NSUP: integer     ! Number of SUPPLI
    SERVMAX: integer    ! Number of SERVICES
    NWO: integer

 end-declarations

 initializations from DATAFILE
        NWO NSUP as "SUP" SERVMAX as "SERV"
 end-initializations

 declarations
   WORKORDERS = 1..NWO
   valid_serv: array(WORKORDERS) of integer
 end-declarations

 initializations from DATAFILE
        valid_serv
 end-initializations

declarations
        SUPPLI = 1..NSUP
        SERVICES = 1..SERVMAX

        cn: array (WORKORDERS,SERVICES) of integer

        total_distance: array(WORKORDERS) of real
        D_max: array(WORKORDERS) of real

        !scores is a parameter provided by the user in the datafile;
        !indicates the scores for the suppliers providing a particular
        !service in a given work order

        score_matrix,SCORE: array(WORKORDERS, SERVICES, SUPPLI) of real
```

```
        capacity: array(SUPPLI) of integer

        duration: array(WORKORDERS,SERVICES,SUPPLI) of integer

        !distance matrix is a parameter and it is a symetric matrix with distances between
pairs of suppliers
        distance_matrix: array(SUPPLI, SUPPLI) of integer
        formulation: integer
        !binary decision variable that are 1 if supplier k is assigned to service j in
workorder a
        supplier_to_service: array (WORKORDERS, SERVICES, SUPPLI) of mpvar
        avg_score,opt_time,objective: real
        no_vars: integer
        starttime: real
        Total_profit: linctr
        con_w: dynamic array(WORKORDERS,SERVICES) of linctr
        con_suppli: dynamic array(SUPPLI) of linctr
 end-declarations
setparam("XPRS_CPUTIME",1)

 initializations from DATAFILE
    score_matrix duration capacity distance_matrix D_max
 end-initializations

 !objective function
 Total_profit:= sum(a in WORKORDERS,j in SERVICES,k in SUPPLI)
score_matrix(a,j,k)*supplier_to_service(a,j,k)

 !constraints

 ! each valid service in a work order must be assigned to a single supplier
forall (a in WORKORDERS, j in 1..valid_serv(a)) con_w(a,j):= (sum(k in SUPPLI)
supplier_to_service(a,j,k) = 1)

!inactive services will not be considered
 forall (a in WORKORDERS,j in valid_serv(a)+1..SERVMAX) con_w(a,j):= sum(k in SUPPLI)
supplier_to_service(a,j,k) = 0

 ! for each supplier the services assigned must not exceed the available capacity
 forall (k in SUPPLI) con_suppli(k):= (sum (a in WORKORDERS) sum (j in SERVICES)
duration(a,j,k)* supplier_to_service(a,j,k)) <= capacity(k)

 ! Following are sign constraints

 forall (a in WORKORDERS, j in SERVICES, k in SUPPLI)supplier_to_service(a,j,k) is_binary

setparam("XPRS_HEURSTRATEGY",-1)
 starttime:= gettime
 !solve the problem
 maximize(Total_profit)
 opt_time:= gettime-starttime

 passes:= 0
 passes += 1

 !Printing objective value to screen
 writeln("\n Begin running model for the first pass")
 if(getprobstat<>XPRS_OPT) then
        writeln("\n Problem is infeasible")
        else
        writeln("\n The total score (profit) is z = $", getobjval)
 end-if

 !Output solution to screen as 3D matrix
 writeln("\n Following is the assignment of SUPPLIERS to services in workorders: \n")
 write("\n                   SUPPLIER \n\n      ")
 forall (k in SUPPLI) write("  ", k, "   ")
        writeln(" \n-----------------------------------------")
 forall(a in WORKORDERS, j in SERVICES) do
              write(strfmt(a,3,0)," sv",strfmt(j,2,0))
```

```
                        forall(k in SUPPLI)do
                                if(j > valid_serv(a)) then write("   -  ")
                                else
                                        write("   ", getsol(supplier_to_service(a,j,k))," ")
                end-if
                        end-do
                                writeln(" ")
 end-do

 writeln("\n End running model first pass")

 ! solution printing to output file
 fopen(OUTFILE, F_APPEND)
  writeln("\n Begin running model for the first pass")
  if(getprobstat<>XPRS_OPT) then
        writeln("\n Problem is infeasible")
  else
        writeln("\n Total profit: $", strfmt(getobjval,5,2))
  end-if
  writeln("\n Following the assignment of suppliers to services in workorders: \n")
  write("\n                  SUPPLIER \n\n       ")
  forall (k in SUPPLI) write("  ", k, "  ")
        writeln(" \n")
  forall (a in WORKORDERS, j in SERVICES) do write(a," sv", j)
        forall(k in SUPPLI) write("   ", getsol(supplier_to_service(a, j, k))","  ")
        writeln(" ")
  end-do

  writeln("\n End running model first pass")
  fclose(F_APPEND)
 graph:=IVEaddplot("Score",IVE_BLUE)
 forall(a in WORKORDERS)do
        WOSC:= a
        IVEdrawpoint(graph,gettime, getobjval)
 end-do

 (! Computing total distance for all workorders
  Look for chosen suppliers for work order a (these suppliers should be stored in arrays
because later we will use these supplier numbers to prohibit that all of them be in a
solution for a particular work order !)

  forall (a in WORKORDERS) do
  !writeln("\n ================== ", passes, " ================== ", passes," =======" )
  no_times := 0
  supplis:= 0

  total_distance(a):= 0
        forall (j in 1..valid_serv(a)) do
                forall(k in SUPPLI) do
                        if getsol(supplier_to_service(a,j,k)) = 1 then
                                if j = 1 then

                                        cn(a,j):=k
                                        supplis +=1
                                        writeln("\n Work Order ",A," \n\n First supplier is
",k)
                                else
                                                                                cn(a,j):= k
                                        if (cn(a,j) <> cn(a,j-1)) then
                                                supplis +=1
                                        end-if

                                        writeln("\n The next supplier is ", k)
                                        total_distance(a) += distance_matrix(cn(a,j-1), k)
                                        writeln("\n The distance between Supplier ",cn(j-
1)," and Supplier ",k," is ", distance_matrix(cn(j-1),k))
                                        !writeln("\n    ", cn(j-1)," <--------- ",
distance_matrix(cn(j-1),k)," ---------> ",k)
                                end-if
                        end-if
                end-do !SUPPLI
```

```
        end-do !SERVICES

    writeln ("\n The total distance traveled in workorder ", a," is ", total_distance(a))
    writeln ("\n The maximum distance allowed for workorder ", a," is ", D_max(a))
end-do !) !work order report
  ! average score
    avg_score:= getobjval/(sum(a in WORKORDERS)valid_serv(a))
    writeln("\naverage score: ",avg_score,"\n")

   writeln("(", opt_time, " sec) Optimal solution: ", getobjval,"\n*  *  *  *  *  *\n")
  objective:= getobjval
  time_sec:= gettime-starttime
        no_vars:= WO*SERV*SUP
        formulation:= 1
initializations to "mmodbc.excel:results_OUTPUT.xlsm"
        opt_time as 'sec' !'time_sec'
        avg_score as 'avg' !'avg_score'
        objective as 'obj' !'objective'
        no_vars as 'vars' !'no_vars'
        INST as 'instance'
        formulation as 'formulation'
end-initializations

end-model
```

APPENDIX C

Traditional Formulation with Cuts

```
model "Supplychain_tradform with cuts"
 uses "mmxprs", "mmsystem"
 uses "mmodbc", "mmive" !this is to gain access to the Xpress-Optimizer solver

(!declarations
       WO,SERV,SUP: integer
end-declarations
       instance =
 end-parameters !)
 parameters
       NWO = 3
       SERV= 3
       SUP    = 5
       INST= 1
       dist_red_goal= 0.50
       known_threshold= true
       DATAFILE=
"column_generation_data.dat"!string("GAP_"+WO+"_workorders_"+SERV+"_services_"+SUP+"_supp
liers_"+INST+".dat")
       OUTFILE= string("result_"+DATAFILE)
       formulation=1
 end-parameters
       !Modify Optimizer control parameter MAXNODE
!Modify Optimizer control parameter LPITERLIMIT
setparam("XPRS_LPITERLIMIT",100000)
 declarations

       need_to_solve: boolean
       passes: integer !number of times the optimization problem needs to be solved
       ! before finding a solution that satisfies the distance constraint
       ! There are 3 sets: suppliers, services, and workorders
       ! Without lost of generality we are assuming equal number of services in each
workorder
    NSUP: integer       ! Number of SUPPLI
    SERVMAX: integer    ! Number of SERVICES
   ! NWO: integer

 end-declarations

 !initializations from DATAFILE
 !     NWO NSUP SERVMAX
 !end-initializations

declarations
   WORKORDERS = 1..NWO
   valid_serv: array(WORKORDERS) of integer
end-declarations

initializations from DATAFILE
   valid_serv
end-initializations

declarations
       SUPPLI = 1..SUP
       SERVICES = 1..SERV

       cn: array (WORKORDERS,SERVICES) of integer
```

```
       !valid_serv: array(WORKORDERS) of integer !this array will let us produce
       !some constraints just for valid services in each workorder

       total_distance: array(WORKORDERS) of integer
       D_max: array(WORKORDERS) of real

       score_matrix,SCORE: array(WORKORDERS, SERVICES, SUPPLI) of real

       !capacities of each supplier is also a parameter provided by the user
       capacity: array(SUPPLI) of integer

       !capacities required by services in different work orders
       ! we are assuming all suppliers will take the same hours in completing
       !a given service in a given work order therefore supplier index is not there
       duration: array(WORKORDERS,SERVICES,SUPPLI) of integer

       !distance matrix is a parameter and it is a symetric matrix with distances between
       pairs of suppliers
       distance_matrix: array(SUPPLI, SUPPLI) of integer

       !binary decision variable that are 1 if supplier k is assigned to service j in
       workorder a
       supplier_to_service: array (WORKORDERS, SERVICES, SUPPLI) of mpvar
       iteration: integer
 end-declarations


 !Parameters read from a datafile
 initializations from DATAFILE
    score_matrix duration capacity distance_matrix valid_serv!D_max as "D_max"
 end-initializations

!Modify Optimizer control parameter LPITERLIMIT
!setparam("XPRS_LPITERLIMIT",1000)
!Modify Optimizer control parameter CPUTIME
!setparam("XPRS_CPUTIME",0)

 !objective function
       Total_profit:= sum(a in WORKORDERS,j in SERVICES,k in SUPPLI)
                           score_matrix(a,j,k)*supplier_to_service(a,j,k)

 !constraints

 forall(a in WORKORDERS, j in 1..valid_serv(a)) con_w(a,j):= (sum(k in SUPPLI)
supplier_to_service(a,j,k) = 1)
  !inactive services will not be considered
 forall(a in WORKORDERS,j in valid_serv(a)+1..SERV) con_w(a,j):= sum(k in SUPPLI)
              supplier_to_service(a,j,k) = 0

 ! each supplier the services assigned must not exceed the supplier capacity
 forall(k in SUPPLI) con_suppli(k):= (sum (a in WORKORDERS) sum (j in SERVICES)
        duration(a,j,k)* supplier_to_service(a,j,k)) <= capacity(k)


 forall(a in WORKORDERS,j in SERVICES,k in SUPPLI) supplier_to_service(a,j, k) is_binary


 starttime:= gettime
 !solve the problem
 maximize(Total_profit)
 opt_time:= gettime-starttime

! writeln("(", gettime-starttime, " sec) Optimal solution: ", getobjval)
 passes:= 1
 iteration:= 1
 ! Printing objective value to screen
 writeln("\n Begin running model for the first pass")
 if(getprobstat<>XPRS_OPT) then
       writeln("\n Problem is infeasible")
 else
       writeln("\n The total score (profit) is z = $", getobjval)
```

```
 end-if

 !Output solution to screen as 3D matrix
 writeln("\n Following is the assignment of SUPPLIERS to services in workorders: \n")
 write("\n                  SUPPLIER \n\n      ")
 forall (k in SUPPLI) write("  ", k, "   ")
       writeln(" \n-----------------------------------------")
 forall(a in WORKORDERS, j in SERVICES) do
              write(strfmt(a,3,0)," sv",strfmt(j,2,0))
                     forall(k in SUPPLI)do
                            if(j > valid_serv(a)) then write("   -  ")
                            else
                                   write("   ",getsol(supplier_to_service(a,j,k))," ")
              end-if
                     end-do
                            writeln(" ")
 end-do

 writeln("\n End running model first pass")

 ! solution printing to output file too
 fopen(OUTFILE, F_APPEND)
  writeln("\n Begin running model for the first pass")
  if(getprobstat<>XPRS_OPT) then
       writeln("\n Problem is infeasible")
  else
       writeln("\n Total profit: $", strfmt(getobjval,5,2))
  end-if
  writeln("\n Following the assignment of suppliers to services in workorders: \n")
  writeln("\n                   SUPPLIER \n\n       ")
  forall (k in SUPPLI) write("  ", k, "  ")
       writeln(" \n")
  forall (a in WORKORDERS, j in SERVICES) do write(a," sv", j)
       forall(k in SUPPLI) write("   ", getsol(supplier_to_service(a, j, k))," ")
       writeln(" ")
  end-do
 !)
  writeln("\n End running model first pass")
  fclose(F_APPEND)
 !graph:=IVEaddplot("Score",IVE_BLUE)
 !forall(a in WORKORDERS)do    ! code to plot iteration progress
!       WOSC:= a
!       IVEdrawpoint(graph,gettime, getobjval)
! end-do

  (! Computing total distance for all workorders
   Look for chosen suppliers for work order a (these suppliers should be stored in arrays
such that they can be used to prohibit recurrence of the same combination of suppliers
for a particular work order !)

if known_threshold = false then
         initializations from DATAFILE
                D_max
         end-initializations
         else
         initializations from DATAFILE
                D_max as "D_max50"
         end-initializations
end-if


repeat
        need_to_solve:= false
        overall_distance:= 0
        IVEerase                !First erase the canvas
  forall (a in WORKORDERS) do
  writeln("\n ================== ", passes, " ==================== ", passes," =======" )
  no_times := 0
  supplis:= 0

  total_distance(a) := 0
```

```
        forall (j in 1..valid_serv(a)) do
                forall(k in SUPPLI) do
                        !this only happens once
                        if getsol(supplier_to_service(a,j,k)) = 1 and no_times = 0 then
                                column_number_1:=k
                                cn(a,j):=k  !cn0:=k
                                supplis +=1
                                !writeln("\n Work Order ", a," \n\n First supplier is ", k)
                                no_times += 1
                                wo += 1
                        !this happens twice in a problem where the work order with the
                        largest number of valid services is 3
                        elif getsol(supplier_to_service(a,j,k)) = 1 and no_times >= 1 then
                                        column_number_2:=k
                                        if (column_number_1 <> column_number_2) then
                                                supplis +=1
                                        end-if
                                        if (no_times = 1) then cn1:=k
                                        elif(no_times = 2)then cn2:=k
                                        end-if
                                        cn(a,j):= k
                                        writeln("\n The next supplier is ", k)
                                        no_times += 1
                                        total_distance(a) +=
                                                        distance_matrix(column_number_1, k)
                                        writeln("\n The distance between Supplier ",
column_number_1, " and Supplier ", k, " is ", distance_matrix(column_number_1, k))
                                        writeln("\n    ", column_number_1," <--------- ",
distance_matrix(column_number_1, k)," ---------> ", k)
                                        column_number_1 := k
                        end-if
                end-do !SUPPLI
        end-do !SERVICES
        !work order report
    writeln ("\n The total distance traveled in workorder ", a," is ", total_distance(a))
    writeln ("\n The maximum distance allowed for workorder ", a," is ", D_max(a))


 !distance threashold test
        if (total_distance(a)>D_max(a)) then
                writeln("\n|**|$$|** ELIMINATE SUCH COMBINATION OF SUPPLIERS & RERUN
                PROBLEM")

                if known_threshold then
                        need_to_solve:=true
                end-if

                !cut formulation

                sum(j in 1..valid_serv(a)) supplier_to_service(a,j,cn(a,j))<= no_times-1 !

   !    else need_to_solve:=false


        overall_distance+= total_distance(a)

        if passes = 1 then
                opt_dist+= total_distance(a)
                first_bound(a):= total_distance(a)
        end-if


                !elif total_distance(a)< best_bound(a) then
                !       best_bound(a):= total_distance(a)

   if not known_threshold then
                if overall_distance > ((1-dist_red_goal)*opt_dist) then !
                        need_to_solve:= true
                        !
                        !end-if
                !if(total_distance(a)>D_max(a)) then
```

```
                !       move_bound:= false

          !     end-if

                elif overall_distance < ((1-dist_red_goal)*opt_dist) then
                        writeln("if test")
                        need_to_solve:= false
           !    end-if

                if (total_distance(a)<=D_max(a)) and total_distance(a) >=
(0.5*first_bound(a)) then
                        D_max(a)-= 1
                end-if
   end-if
   end-if

                !if overall_distance <

        !if total_distance(a) > 0.25*opt_dist(a) then ! and total_distance(a) <
best_bound(a) then

        !end-if

  end-do        !end work order report
    !average score

    !avg_score:= getobjval/(sum(a in WORKORDERS)NSERV(a))
    !writeln("\naverage score: ",avg_score,"\n")
  !rerun original problem w/cuts if is needed for at least one work order after new
       constraints are added
  if (need_to_solve=true) then
        !solve the problem

        !minimize(XPRS_LIN,Total_profit)
         maximize(Total_profit)
                passes+= 1
                !IVEdrawpoint(graph,gettime, getobjval)
                writeln("\n ////////////////////////\n\n Begin running model with cuts")

                        if(getprobstat<>XPRS_OPT) then
                                writeln("\n Problem is infeasible")
                        else
                                writeln("\n The total score (profit) is z = $", getobjval)
                        end-if

                writeln("\n Following is the assignment of SUPPLIERS to services in
                        workorders: \n")
                write("\n                   SUPPLIER \n\n       ")
                forall (k in SUPPLI) write("  ", k, "  ")
                writeln(" \n")

                forall (a in WORKORDERS, j in SERVICES) do
                        write(a," sv", j)

                        forall(k in SUPPLI) write("   ", getsol(supplier_to_service(a, j,
                                                k)),"  ")
                        writeln(" ")
                end-do

                writeln("\n End running model")
                fopen(OUTFILE, F_APPEND)
                        if(getprobstat<>XPRS_OPT) then
                                writeln("\n Problem is infeasible")
                        else
                                writeln("\n Total profit: $", strfmt(getobjval,5,2))
                        end-if

                        writeln("\n Following the assignment of suppliers to services in
                                workorders: \n")
                        writeln("\n                   SUPPLIER \n\n       ")
                        forall (k in SUPPLI) write("  ", k, "  ")
```

```
                        writeln(" \n")
                        forall(a in WORKORDERS, j in SERVICES) do write(a," sv", j)

                        forall(k in SUPPLI)
                               write("   ", getsol(supplier_to_service(a, j, k)),"  ")
                               writeln(" ")
                        end-do

                         writeln("\n End running model")
                fclose(F_APPEND)
                !)
   dist_reduction:= (1-(overall_distance/opt_dist))*100
                writeln("distance reduction: ",dist_reduction,"%")
                end-if
   writeln("(", gettime-starttime, " sec) Optimal solution: ", getobjval)
   !writeln(" iteration: ",iteration)

                !graph:=IVEaddplot("agv_score(dist_reduction)",IVE_RED)  !Create a graph
 until (need_to_solve=false)
  ! average score
    avg_score:= getobjval/(sum(a in WORKORDERS)valid_serv(a))
    !!writeln("\naverage score: ",avg_score,"\n")
writeln("beginning distance: ",opt_dist)
writeln("ending distnance:   ",overall_distance)
dist_reduction:= (1-(overall_distance/opt_dist))*100
writeln("distance reduction: ",dist_reduction,"%")
  objective:= getobjval
  time_sec:= gettime-starttime
       no_vars:= NWO*SERV*SUP
initializations to "mmodbc.excel:results_OUTPUT.xlsm"
       time_sec as 'time_s' !'time_sec'
       avg_score as 'av' !'avg_score'
       objective as 'objval' !'objective'
       no_vars as 'num_vars' !'no_vars'
       INST as 'inst'
       formulation as 'fml'
       dist_reduction as "dist_reduction"
end-initializations

if not known_threshold then
       initializations to DATAFILE
              D_max as "D_max10"
       end-initializations
end-if

end-model
```

APPENDIX D

Quadradically Constrained Nonlinear Optimization Model

```
model "Supplychain_tradform NLP"
 uses "mmxprs", "mmsystem"
 uses "mmodbc", "mmive","mmnl" !this is to gain access to the Xpress-Optimizer solver

parameters
        NWO     = 3
        SERV= 3
        SUP     = 5
        INST= 1
        DATAFILE="column_generation_data.dat"!
string("GAP_"+WO+"_workorders_"+SERV+"_services_"+SUP+"_suppliers_"+INST+".dat")
        OUTFILE= string("result_"+DATAFILE)
        formulation=1
 end-parameters

!       XSLPcommand(A: string)
!declarations section
declarations

        !next 3 variables are useful for computation of distance in a workorder
        no_times: integer ! similar to max # of services in work order for small example
        ! with up to 3 services in a work order the max value that no_times is 3
        column_number1 : integer
        column_number2 : integer

    ! variables to record the id's of suppliers involved in a given work order
        cn_0 : integer
        cn_1 : integer
        cn_2 : integer

need_to_solve: boolean

        supplis: integer ! number of different suppliers in a work order
        passes: integer !number of times the optimization problem needs to be solved
        ! before finding a solution that satisfies the distance constraint

        ! There are 3 sets: suppliers, services, and workorders

        NSUP: integer     ! Number of SUPPLI
        SERVMAX: integer    ! Number of SERVICES
        !NWO: integer

 end-declarations

 !initializations from DATAFILE
 !      NWO SUP SERV
 !end-initializations

 declarations
   WORKORDERS = 1..NWO
   valid_serv: array(WORKORDERS) of integer
 end-declarations

 initializations from DATAFILE
        valid_serv
 end-initializations
```

```
declarations
        SUPPLI = 1..SUP
        SERVICES = 1..SERV

        cn: array (WORKORDERS,SERVICES) of integer
        !valid_serv : array(WORKORDERS) of integer !this array will let us produce
        !some constraints just for valid services in each workorder

        !invalid_serve: array(WORKORDERS) of integer ! for a particular service
        ! assignments should have an equal to 0 RHS for these invalid services

        total_distance: array(WORKORDERS) of integer
        D_max: array(WORKORDERS) of integer
        wo_distance: array(WORKORDERS) of integer

        score_matrix,SCORE: array(WORKORDERS, SERVICES, SUPPLI) of real

        !capacities of each supplier is also a parameter provided by the user
        capacity: array(SUPPLI) of integer

        !capacities required by services in different work orders
        ! we are assuming all suppliers will take the same hours in completing
        !a given service in a given work order therefore supplier index is not there
        duration: array(WORKORDERS,SERVICES) of integer

        !distance matrix is a parameter and it is a symetric matrix with distances between
pairs of suppliers
        distance_matrix: array(SUPPLI, SUPPLI) of integer

        !binary decision variable that are 1 if supplier k is assigned to service j in
workorder a
        supplier_to_service: array (WORKORDERS, SERVICES, SUPPLI) of mpvar
        iteration: integer
        !con_m:
        con_suppli: array (SUPPLI) of linctr
        con_w: array(WORKORDERS,SERVICES) of linctr!genctr
        con_m: array(WORKORDERS) of nlctr
        profit: mpvar
        Total_profit: linctr! gexp
 end-declarations

 !Things that we will read from a datafile
 initializations from DATAFILE
    score_matrix duration as "capacities" capacity as "cap_sup" distance_matrix
 end-initializations
 !Modify Optimizer control parameter LPITERLIMIT
!setparam("XPRS_LPITERLIMIT",100)

! forall(a in WORKORDERS) SLPDATA("IV",wo_distance(a),300)

 !objective function
 Total_profit:= sum(a in WORKORDERS,j in SERVICES,k in SUPPLI)
               score_matrix(a,j,k)*supplier_to_service(a,j,k)

profit = Total_profit
profit is_free

!constraints

forall (a in WORKORDERS, j in 1..valid_serv(a)) con_w(a,j):=
                              (sum(k in SUPPLI) supplier_to_service(a,j,k) = 1)
 !inactive services will not be considered
 forall (a in WORKORDERS,j in valid_serv(a)+1..SERVMAX) con_w(a,j):=
                               sum(k in SUPPLI) supplier_to_service(a,j,k) = 0
(!      if invalid_serve(a) <> 0 then
               forall (j in valid_serv(a)+1..invalid_serve(a)) con_w(a,j):=
                              (sum(k in SUPPLI) supplier_to_service(a,j, k) = 0)
        end-if
 end-do
 !)
```

```
 ! forall (a in WORKORDERS, j in SUPPLI) con_w(a,j):= (sum(k in SUPPLI)
supplier_to_service(a,j,k) = 1)

 ! Second constraint is for each supplier the services selected (or assigned) must not
       exceed the supplier capacity
 forall (k in SUPPLI) con_suppli(k):= (sum (a in WORKORDERS) sum (j in SERVICES)
                              duration(a,j)* supplier_to_service(a,j,k)) <= capacity(k)

 forall (a in WORKORDERS, j in SERVICES, k in SUPPLI) supplier_to_service(a,j,k)
is_binary

!quadratic distance constraint
 forall (a in WORKORDERS) con_m(a):=
       (sum(k,l in SUPPLI)distance_matrix(k,l)*sum(j in 1..valid_serv(a)-1)
              supplier_to_service(a,j,k)*supplier_to_service(a,j+1,l)) <= D_max(a)

!setparam ("xslp_verbose", true)
!setparam ("xslp_log", 0)

 starttime:= gettime
 !solve the problem
 !SLPloadprob(profit)
 !SLPmaximize(profit)

 maximize(Total_profit)

opt_time:= gettime-starttime

! writeln("(", gettime-starttime, " sec) Optimal solution: ", getobjval)

passes:= 1
iteration:= 1

! Printing objective value to screen
  writeln("\n Begin running model for the first pass")
! if(getprobstat<>XPRS_OPT) then
!       writeln("\n Problem is infeasible")
! else
       writeln("\n The total score (profit) is z = $", getobjval)
 !end-if

 !Output solution to screen as 3D matrix
 writeln("\n Following is the assignment of SUPPLIERS to services in workorders: \n")
 write("\n                  SUPPLIER \n\n      ")
 forall (k in SUPPLI) write("  ", k, "   ")
       writeln(" \n----------------------------------------")
 forall(a in WORKORDERS, j in SERVICES) do
              write(strfmt(a,3,0)," sv",strfmt(j,2,0))
                     forall(k in SUPPLI)do
                            if(j > valid_serv(a)) then write("   -  ")
                            else
                                   write("   ",getsol(supplier_to_service(a,j,k)),"  ")
              end-if
                     end-do
                            writeln(" ")
 end-do

 writeln("\n End running model first pass")
(!
 ! solution printing to output file too
 fopen(OUTFILE, F_APPEND)
         writeln("\n Begin running model for the first pass")
        !if(getprobstat<>XPRS_OPT) then
        !      writeln("\n Problem is infeasible")
        ! else
              writeln("\n Total profit: $", strfmt(getobjval,5,2))
        ! end-if
         writeln("\n Following the assignment of suppliers to services in workorders:
                \n")
         writeln("\n                  SUPPLIER \n\n       ")
         forall (k in SUPPLI) write("  ", k, "  ")
```

```
            writeln(" \n")

          forall (a in WORKORDERS, j in SERVICES) do write(a," sv", j)
               forall(k in SUPPLI) write("   ",getsol(supplier_to_service(a, j, k)),"  ")
               writeln(" ")
          end-do

          writeln("\n End running model first pass")
  fclose(F_APPEND)
 graph:=IVEaddplot("Score",IVE_BLUE)
 forall(a in WORKORDERS)do
        WOSC:= a
        IVEdrawpoint(graph,gettime, getobjval)
 end-do
 !)
 (! Computing total distance for all workorders
Look for chosen suppliers for work order a (these suppliers should be stored in arrays
such that they can be used to prohibit recurrence of specific combination for a
particular work order !)


 !initializations from "raw:shmem"
 !      D_max
 !end-initializations

repeat
        !SLPglobal
          writeln("\n ============== ", passes, " ==================== ", passes,"
===========" )
        need_to_solve:= false
  forall (a in WORKORDERS) do
  writeln("\n********************* Work Order",a," ******************\n")
  no_times := 0
  supplis:= 0
 maximize(Total_profit)
  total_distance(a) := 0
!DumpStack
        !total_distance(a):=
                (sum(k,l in SUPPLI)distance_matrix(k,l)*sum(j in 1..valid_serv(a)-1)
                        supplier_to_service(a,j,k)*supplier_to_service(a,j+1,l))

        forall(j in 1..valid_serv(a)) do
                forall(k in SUPPLI) do
                        !this only happens once
                        if getsol(supplier_to_service(a,j,k)) = 1 and j <= 1 then
                                column_number_1:=k
                                cn(a,j):=k  !cn0:=k
                                supplis +=1
                                writeln("      service ",j,": supplier ", k)
                                no_times += 1
                                wo += 1
                        !this happens twice in a problem where the work order with the
                         largest number of valid services is 3
                        elif getsol(supplier_to_service(a,j,k)) = 1 and j >= 1 then
                                        column_number_2:=k
                                        if (column_number_1 <> column_number_2) then
                                                supplis +=1
                                        end-if

                                        if (no_times = 1) then cn1:=k
                                        elif(no_times = 2)then cn2:=k
                                        end-if

                                        cn(a,j):= k
                                        writeln("\n      service ",j,": supplier ", k)
                                        no_times += 1

                                total_distance(a)+=distance_matrix(column_number_1, k)

                                        writeln("\n The distance between Supplier ",
column_number_1, " and Supplier ", k, " is ", distance_matrix(column_number_1, k))
```

```
                                    writeln("\n    ", column_number_1," <--------- ",
distance_matrix(column_number_1, k)," ---------> ", k)
                                    column_number_1 := k
                        end-if
                end-do !SUPPLI
        end-do !SERVICES
    !work order report
    writeln ("\n The total distance traveled in workorder ", a," is ", total_distance(a))
    writeln ("\n The maximum distance allowed for workorder ", a," is ", D_max(a))


 !distance threashold test
        if total_distance(a).sol>= 100 then
                writeln("\n|**|$$|** ELIMINATE SUCH COMBINATION OF SUPPLIERS & RERUN
                        PROBLEM")
                need_to_solve:=true
        end-if
 !end-do
                !cut formulation

                sum(j in 1..valid_serv(a)) supplier_to_service(a,j,cn(a,j))<= no_times-1

   !    else need_to_solve:=false
   !    end-if              ! this is for larger problems
   !    D_max(a):= total_distance(a) - 1
  end-do        !end work order report

    !average score
    avg_score:= getobjval/(sum(a in WORKORDERS)valid_serv(a))

  writeln("\naverage score: ",avg_score,"\n")

    !rerun original problem w/cuts if is needed for at least one work order after new
       constraints are added

    if (need_to_solve=true) then
         !solve the problem
         !forall (a in WORKORDERS) con_m(a):=
                (sum(k,l in SUPPLI)distance_matrix(k,l)*sum(j in 1..NSERV(a)-1)
                   supplier_to_service(a,j,k)*supplier_to_service(a,j+1,l)) <= D_max(a)

         maximize(Total_profit)
                passes+= 1
                !IVEdrawpoint(graph,gettime, getobjval)
                writeln("\n ////////////////////////\n\n Begin running model with cuts")

                        if(getprobstat<>XPRS_OPT) then
                                writeln("\n Problem is infeasible")
                        else
                                writeln("\n The total score (profit) is z = $", getobjval)
                        end-if

                 writeln("\n Following is the assignment of SUPPLIERS to services in
                          workorders: \n")
                 write("\n                  SUPPLIER \n\n       ")

                forall (k in SUPPLI) write("  ", k, "  ")
                writeln(" \n")

                forall (a in WORKORDERS, j in SERVICES) do
                        write(a," sv", j)

                        forall(k in SUPPLI)write(" ",getsol(supplier_to_service(a,j,k)),
                                                 " ")

                        writeln(" ")
                end-do

                writeln("\n End running model")
```

```
                fopen(OUTFILE, F_APPEND)
                        if(getprobstat<>XPRS_OPT) then
                                writeln("\n Problem is infeasible")
                        else
                                writeln("\n Total profit: $", strfmt(getobjval,5,2))
                        end-if

                        writeln("\n Following the assignment of suppliers to services in
                                workorders: \n")
                        writeln("\n                  SUPPLIER \n\n       ")

                        forall (k in SUPPLI) write("  ", k, "  ")

                        writeln(" \n")

                        forall (a in WORKORDERS, j in SERVICES) do
                                write(a," sv", j)

                                forall(k in SUPPLI)

                                write("   ", getsol(supplier_to_service(a, j, k)),"  ")
                                writeln(" ")
                        end-do

                        writeln("\n End running model")
                fclose(F_APPEND)

 end-if
   writeln("(", gettime-starttime, " sec) Optimal solution: ", getobjval)
   writeln(" iteration: ",iteration)

 until need_to_solve=false

    ! average score
    avg_score:= getobjval/(sum(a in WORKORDERS)valid_serv(a))
    !!writeln("\naverage score: ",avg_score,"\n")

  objective:= getobjval
  time_sec:= gettime-starttime
  no_vars:= NWO*SERV*SUP

initializations to "mmodbc.excel:results_OUTPUT.xlsm"
        time_sec as 'sec' !'time_sec'
        avg_score as 'avg' !'avg_score'
        objective as 'obj' !'objective'
        no_vars as 'vars' !'no_vars'
        INST as 'instance'
        formulation as 'formulation'
end-initializations
!)
!)
end-model
```

APPENDIX E

Column Generation Master Model

```
model "column_generation_formulation"
uses "mmxprs", "mmsystem","mmjobs", "mmive"; !gain access to the Xpress-Optimizer solver
 !uses  "mmnl";
!optional parameters section
parameters
        NWO = 10
        SERV= 10
        SUP    = 50
        INST= 1
        DATAFILE=
string("GAP_"+NWO+"_workorders_"+SERV+"_services_"+SUP+"_suppliers.dat")!_"+INST+".dat")

end-parameters
declarations
        WORKORDERS = 1..NWO
        SERVICES= 1..SERV
        SUPPLI = 1..SUP
        NCOLS: array(SUPPLI) of integer
        price: real
end-declarations

forward procedure column_gen
!forward function column_knapsack(capacities:array(WORKORDERS,SERVICES) of integer,
!                                                  cap_sup: array(SUPPLI) of integer,
!
        initialsolution:array(WORKORDERS,SERVICES,SUPPLI) of integer):integer
forward function supplier_knapsack(SCORE_DUAL:array(WORKORDERS,SERVICES)of real,
                                                  sup_scores:
array(WORKORDERS,SERVICES) of real,

        capacities:array(WORKORDERS,SERVICES,SUPPLI) of integer,
                                                  capctr: integer,
                                                  xbest:array(WORKORDERS,SERVICES) of
integer,
                                                  !K: integer,
                                                  SUP_DUAL: real,
                                                  valid_serv: array(WORKORDERS) of
integer):real
forward procedure optimization_report
forward procedure optimization
forward procedure generate_random_cols
!Modify Optimizer control parameter CPUTIME
setparam("XPRS_CPUTIME",1)
declarations
  !next 3 variables are useful for distance computation
        !no_times : integer
        !column_number1 :integer
        !column_number2 : integer
        !cn_0,cn_1,cn_2: integer
        !need_to_solve: boolean
        valid_serv: array(WORKORDERS) of integer
        capacities: array(WORKORDERS,SERVICES,SUPPLI) of integer
        score_matrix: array(WORKORDERS,SERVICES,SUPPLI) of real
        cap_sup: array(SUPPLI) of integer         !      Max capacity per supplier
        !dual_record: array(way,WORKORDERS,SERVICES)of real
        xbest: array(WORKORDERS,SERVICES) of integer
        SCORE_DUAL: array(WORKORDERS,SERVICES)of real
```

```
        !initialsolution: array(WORKORDERS,SERVICES,SUPPLI) of integer
        !sup_scores: array(WORKORDERS,range) of real
        pass: integer
        way_selector: array(SUPPLI,range) of mpvar
        way_matrix: array(SUPPLI,range,WORKORDERS,range)of integer
        xway: integer
        way_score: array(SUPPLI,range) of real
        RHS: array(WORKORDERS,SERVICES) of real
        EPS = 1e-6                          ! Zero tolerance
        !total_distance:  real
        !D_max: array(WORKORDERS)of real

        score: real
        distance_matrix: array(SUPPLI,SUPPLI) of integer
        Grand_Total: linctr
        way_ctr: array(SUPPLI)of linctr
        assignment: array(WORKORDERS,range) of linctr
        supplier: array(SUPPLI)of real
        starttime: real
        max_supplier: integer
        K: integer
        Column_knapsack: Model
        Supplier_knapsack: Model !mpproblem
        SUP_DUAL: real
        PRICE_DUAL,pricing_prob: array(SUPPLI) of real
        new_column: basis
        iter: integer
        col_gen_break: array(SUPPLI) of integer
        scores: array(WORKORDERS,range,SUPPLI) of real
        durations: array(WORKORDERS,range,SUPPLI) of integer

end-declarations

starttime:=gettime

initializations from DATAFILE
    valid_serv as "NSERV"  ! distance_matrix D_max!
    score_matrix as "SCORE"
    cap_sup as "CAP" capacities as "duration"
end-initializations

forall(k in SUPPLI) NCOLS(k):=0

forall(a in WORKORDERS,j in 1..valid_serv(a),k in SUPPLI)do
        duration(a,j,k):= capacities(a,j,k)
end-do

forall(a in WORKORDERS,j in 1..valid_serv(a))do
        forall(k in SUPPLI) do
                scores(a,j,k):= score_matrix(a,j,k)
                !write(scores(a,j,k)," ")
        end-do
        !writeln
end-do

generate_random_cols
generate_random_cols

setparam("XPRS_verbose",true)

res:= compile (" ","Supplier_knapsack.mos","shmem:bim")
load (Supplier_knapsack, "shmem:bim")

forall(k in SUPPLI) way_ctr(k):= sum(b in 1..NCOLS(k))way_selector(k,b) = 1
!asssignment constraints
forall(a in WORKORDERS, j in 1..valid_serv(a)) assignment(a,j):=
        (sum(k in SUPPLI)sum(b in 1..NCOLS(k))way_matrix(k,b,a,j)*way_selector(k,b)) = 1

forall(a in WORKORDERS, j in valid_serv(a)+1..SERV) potato(a,j):=
        (sum(k in SUPPLI)sum(b in 1..NCOLS(k))way_matrix(k,b,a,j)*way_selector(k,b)) = 0
```

```
! way score calculation
forall(k in SUPPLI,b in 1..NCOLS(k))way_score(k,b):=
        sum(a in WORKORDERS,j in 1..valid_serv(a))way_matrix(k,b,a,j)*scores(a,j,k)

! Objective: maximize total value
Grand_Total:= sum(k in SUPPLI,b in 1..NCOLS(k)) way_selector(k,b)*way_score(k,b)

        column_gen

maximize(XPRS_LIN,Grand_Total)

write("\n\n      ")
forall(k in SUPPLI,b in 1..NCOLS(k))do
        if(way_selector(k,b).sol = 1 and way_score(k,b) > 0)then
                write("(",k,") ")
        end-if
end-do

write("\n----------------------------\n")
forall(a in WORKORDERS,j in SERVICES)do
        write("sv ",j)
        forall(k in SUPPLI)do
                write(" | ")
                forall(b in 1..NCOLS(k))do
                        if(way_selector(k,b).sol = 1 and way_score(k,b) > 0)then
                                write(way_matrix(k,b,a,j))
                        end-if
                end-do
        end-do

        write(" = ",RHS(a,j))
        write("\n")
end-do

forall(k in SUPPLI,b in 1..NCOLS(k)) do
        if way_selector(k,b).sol = 1 then
                write(k,"(",b,") ")
        end-if
end-do

writeln("\nobjective value: ",getobjval)
write("\nComputation time: ", gettime-starttime," sec")

!*********************************************************************
!  Column generation loop ~ MASTER PROBLEM
!*********************************************************************
procedure column_gen
  defcut:=getparam("XPRS_CUTSTRATEGY") ! Save setting of `CUTSTRATEGY'
  setparam("XPRS_CUTSTRATEGY", 0)      ! Disable automatic cuts: MIP
  setparam("XPRS_PRESOLVE", 0)         ! Switch presolve off: disable
  setparam("zerotol", EPS)             ! Set comparison tolerance of Mosel

iter:=0
repeat
        iter+=1
        if iter > 1 then
        maximize(XPRS_LIN,Grand_Total)
        savebasis(new_column)

        optimization

        !optimization_report
        end-if
        writeln("\n\n           >  >  ~  column generation loop pass ",iter,"  ~  <  <\n")

        ! supplier column generation loop
        forall(k in SUPPLI)do

                if pricing_prob(k) = 0 and iter > 2 then
                        !writeln(pricing_prob(k))
                        col_gen_break(k):= 1
```

```
                else
                        NCOLS(k)+=1
                        K:= k
                        capctr:= cap_sup(k)
                        forall(a in WORKORDERS,j in SERVICES)do
                                sup_scores(a,j):= score_matrix(a,j,k)
                        end-do
                        !writeln("SUP_SCORES",sup_scores)
                        SUP_DUAL:= PRICE_DUAL(k)
                        price:=
supplier_knapsack(SCORE_DUAL,sup_scores,capacities,capctr,xbest,SUP_DUAL,valid_serv)

                        writeln("price: ",price)
                        pricing_prob(k):= price
                        create(way_selector(k,NCOLS(k)))
                        forall(a in WORKORDERS,j in 1..valid_serv(a)) do
                                way_matrix(k,NCOLS(k),a,j):= xbest(a,j)
                                assignment(a,j)+=
                                        way_matrix(k,NCOLS(k),a,j)*way_selector(k,NCOLS(k))
                        end-do
                        way_selector(k,NCOLS(k)) is_binary
                        way_ctr(k)+= way_selector(k,NCOLS(k))
                        way_score(k,NCOLS(k)):= sum(a in WORKORDERS,j in 1..valid_serv(a))
                                                way_matrix(k,NCOLS(k),a,j)*scores(a,j,k)

                        Grand_Total+= way_selector(k,NCOLS(k))*way_score(k,NCOLS(k))

                end-if

        end-do   ! SUPPLIER column gen loop

        ! print new columns
!write("\n\n         KNAPSACK COLUMNS \n        ")
!forall (k in SUPPLI) write(k, "   ")
!forall(a in WORKORDERS,j in 1..valid_serv(a)) do
!       write("\nsv ",j,"  ")
!       forall(k in SUPPLI)do
!               write("| ")
!               write(way_matrix(k,NCOLS(k),a,j)," ")
!       end-do
!end-do
!writeln

!       if iter >= 100 then
!               break
!       end-if

!generate_random_cols
if iter > 1 then
savebasis(new_column)
        loadprob(Grand_Total)
        if sum(k in SUPPLI)col_gen_break(k) >= SUP then ! and sum(a in WORKORDERS,j in
1..valid_serv(a))RHS(a,j) >= 8 then
                writeln("no profitable column found.\n")
                break
        end-if
end-if

until(false)

  setparam("XPRS_CUTSTRATEGY", defcut) ! Enable automatic cuts
  setparam("XPRS_PRESOLVE", 1)         ! Switch presolve on

 end-procedure
!*************************************************************************************
****
! knapsack problem
!*************************************************************************************
function supplier_knapsack(SCORE_DUAL:array(WORKORDERS,SERVICES)of real,
                           sup_scores: array(WORKORDERS,SERVICES) of real,
```

```
                            capacities:array(WORKORDERS,SERVICES,SUPPLI) of integer,
                            capctr: integer,
                            xbest:array(WORKORDERS,SERVICES) of integer,
                            !K: integer,
                            SUP_DUAL: real,
                            valid_serv: array(WORKORDERS) of integer):real

!with Burglar do
initializations to "raw:noindex"
   SCORE_DUAL as "shmem:SCORE_DUAL"  sup_scores capacities as "shmem:capacities"
   capctr SUP_DUAL valid_serv
end-initializations

run (Supplier_knapsack, "k="+K+",SUPPLIERS="+SUP+",SERVMAX="+SERV+",NWO="+NWO)
                ! Start solving knapsack subproblem

wait                                ! Wait until subproblem finishes
dropnextevent                       ! Ignore termination message

initializations from "raw:"
   xbest as "shmem:xbest" returned as "shmem:score"
end-initializations

(!
 declarations
        con_suppli: linctr
 GenerateWay: linctr
 rkji: array(WORKORDERS,range) of mpvar           ! 1 if we take item i; 0 otherwise
 !xbest: array(WORKORDERS,range) of integer
 score: real
 !valid_serv: array(WORKORDERS)of integer
 !K: integer
 noassign: array(WORKORDERS,range) of linctr
 end-declarations

 !Objective: maximize total value
   GenerateWay:= sum(a in WORKORDERS,j in 1..valid_serv(a))
               ((scores(a,j,K)-SCORE_DUAL(a,j))*rkji(a,j))-SUP_DUAL

 !Capacity constraint
 con_suppli:= (sum(a in WORKORDERS,j in 1..valid_serv(a))duration(a,j)*rkji(a,j))<=capctr
 !BIP
 forall(a in WORKORDERS,j in 1..valid_serv(a)) rkji(a,j) is_binary  ! All x are 0/1
 forall(a in WORKORDERS,j in valid_serv(a)..SERVMAX) noassign(a,j):= rkji(a,j) = 0
 maximize(GenerateWay)                    ! Solve the MIP-problem

writeln("test supplier: ",K)
returned:=getobjval
writeln(getobjval)
forall(a in WORKORDERS,j in 1..valid_serv(a),k in SUPPLI)
xbest(a,j):=integer(rkji(a,j).sol)

 !)

 end-function

!*************************************************************************
! master problem optimization procedure
!
!
! *************************************************************************
 procedure optimization
        forall(a in WORKORDERS,j in SERVICES)do
                SCORE_DUAL(a,j):= getdual(assignment(a,j))
                !IVEdrawpoint(plot1,iter,SCORE_DUAL(a,j))
        end-do
        ! pricing dual values from convexity constraint
        forall(k in SUPPLI) do
                PRICE_DUAL(k):= getdual(way_ctr(k))
                !IVEdrawpoint(plot2,iter,PRICE_DUAL(k))
```

```
        end-do

        !writeln("assignment duals: ",SCORE_DUAL)
        !writeln("convexity duals: ",PRICE_DUAL)
 end-procedure

!*********************************************************************
! optimization report procedure
!
!
! *********************************************************************
 procedure optimization_report

 forall(a in WORKORDERS,j in SERVICES)RHS(a,j):=
        sum(k in SUPPLI)sum(b in 1..NCOLS(k))way_matrix(k,b,a,j)*(way_selector(k,b).sol)

!print way matrix
write("\n\n                    WAY MATRIX \n                 ")
forall (k in SUPPLI) write("supplier ",k,"                       ")
write("\n       ")
forall(k in SUPPLI) do
        forall(b in 1..NCOLS(k))write(strfmt(b,2))
        write("  ")
end-do

write("\n-------------------------------------------------------------------------------------
----------------------")
forall(a in WORKORDERS,j in SERVICES) do
        write("\nsv ",j,"  ")
        forall(k in SUPPLI)do
                write("| ")
                forall(b in 1..NCOLS(k))do
                        write(way_matrix(k,b,a,j)," ")
                end-do
        end-do
        write("= ",RHS(a,j))
end-do
write("\n-------------------------------------------------------------------------------------
----------------------\nXklk  ")
forall(k in SUPPLI) do
        write("| ")
        forall(b in 1..NCOLS(k))write(getsol(way_selector(k,b))," ")
end-do
        (!write("\n\nWay Score\n")
                forall(b in 1..NCOL(k))write(" | way ",strfmt(b,2))
                write("\n")
                forall(k in SUPPLI)do
                        write("\n")
                        forall(b in 1..NCOL(k))write(" |",strfmt(way_score(k,b),7))
                end-do
        write("\n-------------------------------------------------------------------------\n")

        !forall(b in way)write(" | ",strfmt(way_total(b),6))!)
        write("\n\n*-*-*-*-*-*-*-*-*-*-*-*\n|",strfmt("|",22),"\n|  Grand_score:
",strfmt(getobjval,-6),"|\n|",strfmt("|",22),"\n*-*-*-*-*-*-*-*-*-*-*-*")

 end-procedure


!*********************************************************************
! random column generation procedure
!
!
! *********************************************************************
procedure generate_random_cols
! generate initial columns
forall(k in SUPPLI)NCOLS(k)+=1



        !writeln("\nway ",NCOLS(k),"\n")
```

```
       forall(a in WORKORDERS,j in 1..valid_serv(a))do
                      !forall(k in SUPPLI)supplier(k):= scores(a,j,k) !random  ! assign
random supplier
                      !min_supplier:=
minlist(supplier(1),supplier(2),supplier(3),supplier(4),supplier(5))
                       max_supplier:= 1
                       forall(k in SUPPLI)do
                                     if score_matrix(a,j,k) >
score_matrix(a,j,max_supplier) then
                                            max_supplier:= k
                                     end-if
                                     end-do
                      !forall(k in SUPPLI)do
                      !      if supplier(k)= min_supplier  then
                      !             supplier(k):= 1
                      !             else supplier(k):= 0
                      !      end-if
                      !end-do
                      forall(k in SUPPLI)do
                             way_matrix(k,NCOLS(k),a,j):= 0 !integer(supplier(k)) !
feasible but not constrained
                      end-do
                      way_matrix(max_supplier,NCOLS(1),a,j):= 1
              end-do
(!     write("\n\n         Random WAY \n        ") ! write way
forall (k in SUPPLI) write(k, "   ")
forall(a in WORKORDERS,j in SERVICES) do
       write("\nsv ",j,"  ")
       forall(k in SUPPLI)do
              write("| ")
              write(way_matrix(k,xway,a,j)," ")
       end-do
end-do !)
!writeln

repeat waycheck:= false
  repeat invalid_demand:=false
       forall(k in SUPPLI) do      ! verify capacity ctr
              if sum(a in WORKORDERS,j in 1..valid_serv(a))
                      way_matrix(k,NCOLS(k),a,j)*capacities(a,j,k)>cap_sup(k) then
                             invalid_demand:= true
                             waycheck:= true
                             !writeln("XXX supplier ",k," demand ",
              sum(a in WORKORDERS,j in SERVICES)way_matrix(k,xway,a,j)*capacities(a,j),
                             " X< ",cap_sup(k))
              forall(a in WORKORDERS,j in 1..valid_serv(a)) do !reassign ½ of services
                      if way_matrix(k,NCOLS(k),a,j) = 1 then
                             if round(random) = 1 then! remove half of assignments
                               ! assign random supplier
                               forall(l in 1..k-1)supplier(l):= score_matrix(a,j,k)
                               supplier(k):= 0
                               forall(l in k+1..SUP) supplier(l):= score_matrix(a,j,k)
                               max_supplier:=1

                               forall(l in SUPPLI)do
                                     if supplier(l) > score_matrix(a,j,max_supplier) then
                                            max_supplier:= l
                                     end-if
                               end-do
                                     forall(l in SUPPLI)do
                                            if supplier(l)= min_supplier  then
                                                   supplier(l):= 1
                                            else supplier(l):= 0
                                            end-if
                                     end-do

                                     forall(l in SUPPLI)do
                                            way_matrix(l,NCOLS(l),a,j):= 0
                                     end-do
                                     way_matrix(max_supplier,NCOLS(1),a,j):= 1
                             end-if
```

```
                    end-if
                end-do
                !writeln("new demand ",sum(a in WORKORDERS,j in SERVICES)
                                way_matrix(k,xway,a,j)*capacities(a,j)," < ",cap_sup(k))
        end-if  ! invalid capacity loop
        end-do  ! supplier loo
        !forall(k in SUPPLI)do
        !writeln("supplier ",k," demand ",sum(a in WORKORDERS,j in SERVICES)
                                way_matrix(k,xway,a,j)*capacities(a,j)," < ",cap_sup(k))
  !end-do
  until invalid_demand= false

until waycheck = false

forall(k in SUPPLI) do
        create(way_selector(k,NCOLS(k)))
        way_selector(k,NCOLS(k)) is_binary
end-do

 forall(k in SUPPLI)do
        writeln("supplier ",k," demand ",sum(a in WORKORDERS,j in
1..valid_serv(a))way_matrix(k,NCOLS(k),a,j)*capacities(a,j,k)," <= ",cap_sup(k))
 end-do
end-procedure
end-model
```

Supplier Knapsack Code

```
model "Supplier_knapsack"           ! Start a new model

uses "mmxprs"                       ! Load the optimizer library


declarations
SUPPLIERS = 50
SERVMAX = 10
NWO = 10
WORKORDERS = 1..NWO
SERVICES = 1..SERVMAX
SUPPLI = 1..SUPPLIERS

 SCORE_DUAL: array(WORKORDERS,SERVICES) of real        ! score of services
 capacities: array(WORKORDERS,SERVICES,SUPPLI) of integer       ! decision variables
 score_matrix: array(WORKORDERS,SERVICES,SUPPLI) of real
 scores: array(WORKORDERS,range,SUPPLI) of real
 duration: array(WORKORDERS,range,SUPPLI) of integer
 assign_dual: array(WORKORDERS,range) of real
 capctr: integer         !      Max capacity per supplier
 SUP_DUAL: real
 con_suppli: linctr
 GenerateWay: linctr
 rkji: array(WORKORDERS,SERVICES) of mpvar           ! 1 if we take item i; 0 otherwise
 xbest: array(WORKORDERS,SERVICES) of integer
 score: real
 valid_serv: array(WORKORDERS)of integer
 K: integer
 noassign: array(WORKORDERS,range) of linctr
end-declarations

!writeln("knapsack test")

 initializations from "raw:noindex"
        score_matrix  capctr K SUP_DUAL valid_serv      SCORE_DUAL as "shmem:SCORE_DUAL"
        capacities as "shmem:capacities"
 end-initializations
```

```
forall(a in WORKORDERS,j in 1..valid_serv(a),k in SUPPLI)do
       duration(a,j,k):= capacities(a,j,k)
       assign_dual(a,j):= SCORE_DUAL(a,j)
end-do

forall(a in WORKORDERS,j in 1..valid_serv(a))do
       forall(k in SUPPLI) do
              scores(a,j,k):= score_matrix(a,j,k)
              !write(scores(a,j,k)," ")
       end-do
       !writeln
end-do
       !forall(a in WORKORDERS,j in 1..valid_serv(a))do
              !write("wo ",a," service ",j," ")
       !      forall(k in SUPPLI) write(scores(a,j,k),' ')
              !writeln
       !end-do

! Objective: maximize total value
  GenerateWay:= sum(a in WORKORDERS,j in 1..valid_serv(a))
                            ((scores(a,j,K)-SCORE_DUAL(a,j))*rkji(a,j))-SUP_DUAL

! Capacity constraint
con_suppli:= (sum(a in WORKORDERS,j in 1..valid_serv(a),k in SUPPLI)
              duration(a,j,k)*rkji(a,j)) <= capctr
 !BIP

 forall(a in WORKORDERS,j in 1..valid_serv(a)) rkji(a,j) is_binary  ! All x are 0/1
!forall(a in WORKORDERS,j in valid_serv(a)..SERVMAX) noassign(a,j):= rkji(a,j) = 0

  maximize(GenerateWay)                  ! Solve the MIP-problem

score:=getobjval

forall(a in WORKORDERS,j in SERVICES) xbest(a,j):=integer(rkji(a,j).sol)

initializations to "raw:"
  xbest as "shmem:xbest" score as "shmem:score"
end-initializations
                              ! Print out the solution
!writeln("Solution:\n Objective: ", getobjval)
!forall(j in SERVICES, k in SUPPLI)  writeln(" r(", j,k, "): ", rkji(j,k).sol)

end-model
```

APPENDIX F

Row Generation Formulation

Row Generation Master Model

```
model "column_generation_formulation"
uses "mmxprs", "mmsystem","mmjobs", "mmive"; !gain access to the Xpress-Optimizer solver
 !uses  "mmnl";
!optional parameters section
parameters
        NWO = 125
        SERV= 10
        SUP     = 12500
        INST= 1
        DATAFILE=
string("GAP_"+NWO+"_workorders_"+SERV+"_services_"+SUP+"_suppliers_"+INST+".dat")

end-parameters
declarations
!       WORKORDERS: set of integer
        WORKORDERS = 1..NWO
        SERVICES= 1..SERV
        SUPPLI = 1..SUP
        NCOLS: array(WORKORDERS,SERVICES) of integer
        price: real
end-declarations
!finalize(WORKORDERS,SERVICES,SUPPLI)
forward procedure column_gen
!forward function column_knapsack(capacities:array(WORKORDERS,SERVICES) of integer,
!                                                  cap_sup: array(SUPPLI) of integer,
!
        initialsolution:array(WORKORDERS,SERVICES,SUPPLI) of integer):integer
forward function service_knapsack(xbest:array(SUPPLI) of integer):real
forward procedure optimization_report
forward procedure optimization
forward procedure generate_random_cols
!Modify Optimizer control parameter CPUTIME
setparam("XPRS_CPUTIME",0)


declarations
        opt_time: real
        valid_serv: array(WORKORDERS) of integer
        capacities: array(WORKORDERS,range,SUPPLI) of integer       ! decision variables
        score_matrix: array(WORKORDERS,SERVICES,SUPPLI) of real
        cap_sup: array(SUPPLI) of integer         !      Max capacity per supplier
        !dual_record: array(way,WORKORDERS,SERVICES)of real
        xbest: array(SUPPLI) of integer
        PRICE_DUAL: array(WORKORDERS,SERVICES)of real
        !initialsolution: array(WORKORDERS,SERVICES,SUPPLI) of integer
        !sup_scores: array(WORKORDERS,range) of real
        pass: integer
        way_selector: array(WORKORDERS,SERVICES,range) of mpvar
        way_matrix: array(SUPPLI,WORKORDERS,range,range)of integer
        !xway: integer
        way_score: array(SUPPLI,range) of real
        RHS,pricing_prob: array(WORKORDERS,SERVICES) of real
        EPS = 1e-6                            ! Zero tolerance
        !total_distance:  real
```

```
        !D_max: array(WORKORDERS)of real
        !!capctr: array(SUPPLI) of integer,
        score: real
        distance_matrix: array(SUPPLI,SUPPLI) of integer
        Grand_Total: linctr
        way_ctr: array(WORKORDERS,range)of linctr
        capacity_ctr: array(SUPPLI) of linctr
        supplier: array(SUPPLI)of real
        starttime: real
        max_supplier: integer
        A,J: integer
        Service_knapsack: Model !mpproblem
        SCORE_DUAL: array(SUPPLI) of real
        new_column: basis
        iter: integer
        col_gen_break: array(WORKORDERS,range) of integer
        avg_score: real

end-declarations

starttime:=gettime

initializations from DATAFILE
    valid_serv  ! distance_matrix D_max!
    score_matrix
    cap_sup as "capacity"
    capacities as "duration"
end-initializations

forall(a in WORKORDERS, j in 1..valid_serv(a)) NCOLS(a,j):=0

!forall(a in WORKORDERS,j in 1..valid_serv(a),k in SUPPLI)do
!       duration(a,j,k):= capacities(a,j,k)
!end-do

!forall(a in WORKORDERS,j in 1..valid_serv(a))do
!       forall(k in SUPPLI) do
!              scores(a,j,k):= score_matrix(a,j,k)
!              !write(scores(a,j,k)," ")
!       end-do
        !writeln
!end-do

setparam("XPRS_verbose",true)

res:= compile (" ","transposed_knapsack.mos","shmem:bim")
load (Service_knapsack, "shmem:bim")


(!forall(l in SUPPLI) do
        capctr(l):= cap_sup(l)
                        forall(a in WORKORDERS,j in SERVICES,k in SUPPLI)do
                                !sup_scores(a,j):= score_matrix(a,j,k)
                                duration(a,j):= capacities(a,j,k)
                        end-do
!)

! load all fixed info into shared memory

initializations to "raw:noindex"
   valid_serv score_matrix capacities cap_sup
end-initializations

forall(a in WORKORDERS,j in 1..valid_serv(a)) way_ctr(a,j):=
                                        sum(b in 1..NCOLS(a,j))way_selector(a,j,b) = 1

forall(k in SUPPLI)capacity_ctr(k):=
        sum(a in WORKORDERS,j in 1..valid_serv(a),b in 1..NCOLS(a,j))
                capacities(a,j,k)*way_matrix(k,a,j,b)*way_selector(a,j,b) <= cap_sup(k)
```

```
! Objective: maximize total value
Grand_Total:= sum(k in SUPPLI,a in WORKORDERS,j in 1..valid_serv(a),b in 1..NCOLS(a,j))
                              way_selector(a,j,b)*way_matrix(k,a,j,b)*score_matrix(a,j,k)

optimization

        column_gen

maximize(XPRS_LIN,Grand_Total)

!optimization_report

objective:=getobjval
writeln("\nobjective value: ",getobjval)
opt_time:=gettime-starttime
write("\nComputation time: ",opt_time," sec")

avg_score:= getobjval/(sum(a in WORKORDERS)valid_serv(a))
writeln("\naverage score: ",avg_score,"\n")

no_vars:= NWO*SERV*SUP
formulation:=2

initializations to "mmodbc.excel:results_OUTPUT.xlsm"
        opt_time as 'sec' !'time_sec'
        avg_score as 'avg' !'avg_score'
        objective as 'obj' !'objective'
        no_vars as 'vars' !'no_vars'
        INST as 'instance'
        formulation as 'formulation'
end-initializations

!*************************************************************************
!  Column generation loop at the top node:     MASTER MODEL
!    solve the LP and save the basis
!    get the solution values
!    generate new column(s) (=cutting pattern)
!    load the modified problem and load the saved basis
!*************************************************************************
procedure column_gen
  defcut:=getparam("XPRS_CUTSTRATEGY") ! Save setting of `CUTSTRATEGY'
  setparam("XPRS_CUTSTRATEGY", 0)      ! Disable automatic cuts: MIP
  setparam("XPRS_PRESOLVE", 0)         ! Switch presolve off: disable
  setparam("zerotol", EPS)             ! Set comparison tolerance of Mosel

iter:=0

repeat
        iter+=1
        if iter > 1 then
        maximize(XPRS_LIN,Grand_Total)

        writeln
        integer_solution:= true
        repeat
                forall(a in WORKORDERS,j in 1..valid_serv(a),b in 1..NCOLS(a,j)) do
                        if way_selector(a,j,b).sol <>0 and way_selector(a,j,b).sol <>1 then
                                integer_solution:= false
                                break
                                write(getsol(way_selector(a,j,b))," ")
                        end-if
                end-do
                if integer_solution= true then
                        savebasis(new_column)
                        else loadbasis(new_column)
                                maximize(XPRS_LIN,Grand_Total)
                end-if
                until integer_solution = true

        optimization
```

```
        !optimization_report
        end-if
        writeln("\n\n           >  >  ~  column generation loop pass ",iter,"  ~  <  <\n")

        ! supplier column generation loop
        forall(a in WORKORDERS,j in 1..valid_serv(a))do

                if pricing_prob(a,j) <= 0 and iter > 1 then
                        !writeln(pricing_prob(k))
                        col_gen_break(a,j):= 1
                else
                        NCOLS(a,j)+=1
                        A:= a
                        J:= j
                        !writeln("SUP_SCORES",sup_scores)
                        !SUP_DUAL:= PRICE_DUAL(k)
                        price:= service_knapsack(xbest)

                        writeln("price: ",price)
                        pricing_prob(a,j):= price
                        create(way_selector(a,j,NCOLS(a,j)))
                        forall(k in SUPPLI) do
                                way_matrix(k,a,j,NCOLS(a,j)):= xbest(k)
                                capacity_ctr(k)+= way_matrix(k,a,j,NCOLS(a,j))
                                        *way_selector(a,j,NCOLS(a,j))*capacities(a,j,k)
                        end-do
                        way_selector(a,j,NCOLS(a,j)) is_binary
                        way_ctr(a,j)+= way_selector(a,j,NCOLS(a,j))
                        !way_score(k,NCOLS(a,j)):=
                sum(a in WORKORDERS,j in 1..valid_serv(a))way_matrix(k,NCOLS(k),a,j)
                                                                *score_matrix(a,j,k)

                        Grand_Total+= sum(k in SUPPLI)way_selector(a,j,NCOLS(a,j))
                                        *way_matrix(k,a,j,NCOLS(a,j))*score_matrix(a,j,k)

                end-if

        end-do   ! SUPPLIER column gen loop

        ! print new columns
!write("\n\n         KNAPSACK COLUMNS \n        ")
!forall (k in SUPPLI) write(k, "   ")
!forall(a in WORKORDERS,j in 1..valid_serv(a)) do
!       write("\nsv ",j,"  ")
!       forall(k in SUPPLI)do
!               write("| ")
!               write(way_matrix(k,NCOLS(k),a,j)," ")
!       end-do
!end-do
!writeln

!       if iter >= 100 then
!               break
!       end-if

!generate_random_cols
if iter > 0 then

        loadprob(Grand_Total)
        loadbasis(new_column)
        if sum(a in WORKORDERS,j in 1..valid_serv(a))col_gen_break(a,j) >= sum(a in
WORKORDERS)valid_serv(a) then ! and sum(a in WORKORDERS,j in 1..valid_serv(a))RHS(a,j) >=
8 then
                writeln("no profitable column found.\n")
                break
        end-if
 !      if iter >= 3 then
 !              break
 !              end-if
end-if
writeln
```

```
       !end-do
until(false)

  setparam("XPRS_CUTSTRATEGY", defcut) ! Enable automatic cuts
  setparam("XPRS_PRESOLVE", 1)         ! Switch presolve on

 end-procedure
!*******************************************************************************
! knapsack problem
!*******************************************************************************
function service_knapsack(xbest:array(SUPPLI) of integer):real

!with Burglar do
  !initializations to "raw:noindex"
  !     SCORE_DUAL as "shmem:SCORE_DUAL" SUP_DUAL !capacities as "shmem:capacities"
  !    valid_serv sup_scores    capctr
  !end-initializations

run (Service_knapsack,"A="+A+",J="+J+",SUPPLIERS="+SUP+",SERVMAX="+SERV+",NWO="+NWO)
                            ! Start solving knapsack subproblem
wait                                ! Wait until subproblem finishes
dropnextevent                       ! Ignore termination message

 initializations from "raw:"
       xbest as "shmem:xbest" returned as "shmem:score"
 end-initializations

 end-function

!*******************************************************************
! master problem optimization procedure
!
!
! *******************************************************************
 procedure optimization
       forall(k in SUPPLI)do
              SCORE_DUAL(k):= integer(getdual(capacity_ctr(k)))+1
              !IVEdrawpoint(plot1,iter,SCORE_DUAL(a,j))
              !if SCORE_DUAL(k)>0 then
              !      writeln(SCORE_DUAL)
              !end-if
       end-do
       ! pricing dual values from convexity constraint
       forall(a in WORKORDERS,j in 1..valid_serv(a)) do
              PRICE_DUAL(a,j):= getdual(way_ctr(a,j))
              !IVEdrawpoint(plot2,iter,PRICE_DUAL(k))
       end-do
       initializations to "raw:noindex"
              SCORE_DUAL PRICE_DUAL
       end-initializations

       !writeln("assignment duals: ",SCORE_DUAL)
       !writeln("convexity duals: ",PRICE_DUAL)
 end-procedure
!*******************************************************************
! optimization report procedure
!
!
! *******************************************************************
 procedure optimization_report

forall(a in WORKORDERS,j in 1..valid_serv(a))RHS(a,j):=
                            sum(k in SUPPLI,b in 1..NCOLS(a,j))way_matrix(k,a,j,b)
                                                    *(way_selector(a,j,b).sol)
!print way matrix
write("\n\n                   WAY MATRIX \n      ")
forall (k in SUPPLI) write("  ",strfmt(k,-2))
!writeln!("\n       ")
!forall(k in SUPPLI) do
!       !forall)write(strfmt(b,2))
```

```
!       write("  ")
!end-do

write("\n-------------------------------------------------------------------------------
-----------------------------")
forall(a in WORKORDERS,j in 1..valid_serv(a)) do
        write("\nsv ",j,"  ")
        forall(k in SUPPLI)do
                write("| ")
                !forall(b in 1..NCOLS(k))do
                        write(way_matrix(k,a,j,NCOLS(a,j)),"  ")
                end-do
        !end-do
        write("= ",RHS(a,j))
end-do
write("\n-------------------------------------------------------------------------------
-------------------------------\ncapcty")
forall(k in SUPPLI) do
        write("| ",strfmt(sum(a in WORKORDERS,j in 1..valid_serv(a))
                        way_matrix(k,a,j,NCOLS(a,j))*capacities(a,j,k),-2))

end-do
        write("\n\ncapsup")
        forall(k in SUPPLI) do
                write("| ",strfmt(cap_sup(k),-2))
        end-do

        (!write("\n\nWay Score\n")
        forall(b in 1..NCOL(k))write(" | way ",strfmt(b,2))
        write("\n")
        forall(k in SUPPLI)do
                write("\n")
                forall(b in 1..NCOL(k))write(" |",strfmt(way_score(k,b),7))
        end-do

        write("\n-------------------------------------------------------------------------\n")

        forall(b in way)write(" | ",strfmt(way_total(b),6))!)
        write("\n\n*-*-*-*-*-*-*-*-*-*-*-*-*\n|",strfmt("|",24),"\n|  Grand_score:
            ",strfmt(getobjval,-8),"|\n|",strfmt("|",24),"\n*-*-*-*-*-*-*-*-*-*-*-*-*")
        writeln
        writeln
        forall(a in WORKORDERS,j in 1..valid_serv(a),b in 1..NCOLS(a,j)) do
                if way_selector(a,j,b).sol >0 then
                        write(b," ")
                end-if
        end-do

 end-procedure


!*********************************************************************
! random column generation procedure
!
!
! *********************************************************************
procedure generate_random_cols
! generate initial columns
forall(a in WORKORDERS,j in 1..valid_serv(a))NCOLS(a,j)+=1
        !writeln("\nway ",NCOLS(k),"\n")
        forall(a in WORKORDERS,j in 1..valid_serv(a))do
                        !forall(k in SUPPLI)supplier(k):= scores(a,j,k) !random  ! assign
random supplier
                        max_supplier:= 1
                        forall(k in SUPPLI)do
                              if score_matrix(a,j,k) >= score_matrix(a,j,max_supplier) then
                                   max_supplier:= k
                              end-if
                        end-do
                        !if round(random)=1 then
                        !       writeln("2/3 test")
```

```
                        !       max_supplier:= integer(round((random*(SUP-1))+.5))
                        !end-if

                        !forall(k in SUPPLI)do
                        !       if supplier(k)= min_supplier  then
                        !               supplier(k):= 1
                        !               else supplier(k):= 0
                        !       end-if
                        !end-do
                        forall(k in SUPPLI)do   ! feasible but not constrained
                                way_matrix(k,a,j,NCOLS(a,j)):= 0 !integer(supplier(k))
                        end-do

                        way_matrix(max_supplier,a,j,NCOLS(a,j)):= 1
                end-do

(!      write("\n\n         Random WAY \n        ") ! write way
forall (k in SUPPLI) write(k, "   ")
forall(a in WORKORDERS,j in SERVICES) do
        write("\nsv ",j,"  ")
        forall(k in SUPPLI)do
                write("| ")
                write(way_matrix(k,xway,a,j)," ")
        end-do
end-do !)
!writeln

repeat waycheck:= false
  repeat invalid_demand:=false
        forall(k in SUPPLI) do
         if sum(a in WORKORDERS,j in 1..valid_serv(a))way_matrix(k,a,j,NCOLS(a,j))
                                *capacities(a,j,k)>cap_sup(k) then ! verify capacity ctr

                invalid_demand:= true
                waycheck:= true
                !writeln("XXX supplier ",k," demand ",sum(a in WORKORDERS,j in SERVICES)
                          way_matrix(k,a,j,NCOLS(a,j))*capacities(a,j,k)," X< ",cap_sup(k))

                forall(a in WORKORDERS,j in 1..valid_serv(a)) do !reassign 1/2 of services
                        if way_matrix(k,a,j,NCOLS(a,j)) = 1 then
                                if round(random*.75) = 1 then! remove half of assignments
                                        !assign random supplier
                                        forall(l in 1..k-1)supplier(l):= score_matrix(a,j,l)
                                  supplier(k):= 0
                                  forall(l in k+1..SUP) supplier(l):= score_matrix(a,j,l)
                                  max_supplier:=1

                                  forall(l in SUPPLI)do
                                        if supplier(l) >= score_matrix(a,j,max_supplier)
then
                                                max_supplier:= l
                                        end-if
                                        end-do

                                        forall(l in SUPPLI)do
                                                if supplier(l)= min_supplier  then
                                                        supplier(l):= 1
                                                else supplier(l):= 0
                                                end-if
                                        end-do
                                        !)
                                        forall(l in SUPPLI)do
                                                way_matrix(l,a,j,NCOLS(a,j)):= 0
                                                !feasible but not constrained
                                                !integer(supplier(l))
                                        end-do
                                        way_matrix(max_supplier,a,j,NCOLS(a,j)):= 1
                                end-if
                        end-if
                end-do
```

```
!writeln("new demand ",sum(a in WORKORDERS,j in 1..valid_serv(a))
                        way_matrix(k,a,j,NCOLS(a,j))*capacities(a,j,k)," < ",cap_sup(k))
       end-if  ! invalid capacity loop
end-do  ! supplier loop
       !forall(k in SUPPLI)do
       !writeln("supplier ",k," demand ",sum(a in WORKORDERS,j in SERVICES)
                              way_matrix(k,xway,a,j)*capacities(a,j)," < ",cap_sup(k))
  !end-do
  until invalid_demand= false

until waycheck = false

forall(a in WORKORDERS,j in 1..valid_serv(a)) do
       create(way_selector(a,j,NCOLS(a,j)))
       way_selector(a,j,NCOLS(a,j)) is_binary
end-do

 !forall(k in SUPPLI)do
       !writeln("supplier ",k," demand ",sum(a in WORKORDERS,j in 1..valid_serv(a))
                        way_matrix(k,a,j,NCOLS(a,j))*capacities(a,j,k)," <= ",cap_sup(k))
! end-do
 !way:= sum(a in WORKORDERS,j in 1..valid_serv(a),k in SUPPLI)
                              way_matrix(k,a,j,NCOLS(a,j))*score_matrix(a,j,k)
 !writeln("way score ",way)


end-procedure

end-model
```

Row Generation Submodel

```
model "transposed_knapsack"                          ! Start a new model

uses "mmxprs","mmsystem"                           ! Load the optimizer library

parameters
       SUPPLIERS = 25
       SERVMAX = 10
       NWO = 5
       A=1
       J=1
end-parameters

declarations
       WORKORDERS = 1..NWO
       SERVICES = 1..SERVMAX
       SUPPLI = 1..SUPPLIERS
 PRICE_DUAL: array(WORKORDERS,SERVICES) of real
 capacities: array(WORKORDERS,SERVICES,SUPPLI) of integer
 score_matrix: array(WORKORDERS,SERVICES,SUPPLI) of real
 !scores: array(WORKORDERS,range,SUPPLI) of real
 !duration: array(WORKORDERS,SERVICES) of integer
 SCORE_DUAL: array(SUPPLI) of real
 GenerateWay: linctr
 rkji: array(SUPPLI) of mpvar           ! 1 if we take item i; 0 otherwise
 xbest: array(SUPPLI) of integer
 score: real
 valid_serv: array(WORKORDERS)of integer
 assignment: linctr
 start: real
end-declarations

!start:=gettime
!setparam("XPRS_CPUTIME",1)
```

```
 initializations from "raw:noindex"
       score_matrix capacities SCORE_DUAL PRICE_DUAL
 end-initializations

!forall(a in WORKORDERS,j in SERVICES)do
!       duration(a,j):= capacities(a,j,K)
!       assign_dual(a,j):= SCORE_DUAL(a,j)
!end-do
!forall(a in WORKORDERS,j in 1..valid_serv(a))do
!       forall(k in SUPPLI) do
!              scores(a,j,k):= score_matrix(a,j,k)
!              !write(scores(a,j,k)," ")
!       end-do
        !writeln
!end-do
        !forall(a in WORKORDERS,j in 1..valid_serv(a))do
                !write("wo ",a," service ",j," ")
        !       forall(k in SUPPLI) write(scores(a,j,k),' ')
                !writeln
        !end-do

! Objective: maximize total value
GenerateWay:= sum(k in SUPPLI)
                      ((score_matrix(A,J,k)-SCORE_DUAL(k))*rkji(k))-PRICE_DUAL(A,J)

!asssignment constraints
assignment:= (sum(k in SUPPLI) rkji(k)) = 1

 !BIP

 forall(k in SUPPLI) rkji(k) is_binary  ! All x are 0/1

 maximize(GenerateWay)                   ! Solve the MIP-problem

 !writeln("\nComputation time: ", gettime-start," sec")

score:=getobjval

forall(k in SUPPLI) xbest(k):=integer(rkji(k).sol)



initializations to "raw:"
  xbest as "shmem:xbest" score as "shmem:score"
end-initializations
                              ! Print out the solution
!writeln("Solution:\n Objective: ", getobjval)
!forall(j in SERVICES, k in SUPPLI)  writeln(" r(", j,k, "): ", rkji(j,k).sol)

end-model
```

APPENDIX G

Data Generation Code

```
  model GenData
  uses "mmsystem"

  parameters
        NI     = 1
        NWO    = 3
        SERV   = 3
        SUP    = 5
  end-parameters

  declarations
    WORKORDERS = 1..NWO
    NSERV: array(WORKORDERS) of integer
    DATAFILE: string
    SUPPLI = 1..SUP
    SERVICES = 1..SERV
    duration: array(WORKORDERS,SERVICES,SUPPLI) of integer
    SCORE: array(WORKORDERS,SERVICES,SUPPLI) of real
    capacity: array(SUPPLI) of integer
    D_max: array(WORKORDERS) of integer
    distance_matrix: array(SUPPLI,SUPPLI) of integer
    tot: integer
  end-declarations

forall(a in WORKORDERS)do
     valid_serv(a):= integer(round((random*.33+0.67)*SERV))
end-do

setrandseed(666)

! generate distance matrix
        forall(k in SUPPLI) do
                forall(l in SUPPLI) do
                        if l < k then distance_matrix(k,l):= distance_matrix(l,k)
                                elif l = k then distance_matrix(k,l):= 0
                                        else distance_matrix(k,l):= integer(random*100)
                        end-if
                end-do
        end-do

total:=0

! Generate data
            DATAFILE:=
string(text("GAP_")+text(NWO)+"_workorders_"+text(SERV)+"_services_"+text(SUP)+"_supplier
s_"+text(NI)+".dat")

        forall(a in WORKORDERS)do
                !forall(k in SUPPLI)score_root(k):=(random*.75)+0.125 !patterned scores

                forall(j in 1..valid_serv(a)) do
                ! Duration(a,j,k) = random integer in [1,13]
                dur_root:= integer(round((10*random)+0.5))! average dur of service (a,j)

                total+=dur_root
```

```
                forall(k in SUPPLI) do
                        ! random dur within range +-25% of dur_root(a,j)
                        duration(a,j,k):= integer(round(dur_root*((random*.5)+0.75)))

                        ! for patterned scores--> score_root(k)*((random*.25)+0.875)
                        score_matrix(a,j,k):=random
                end-do
            end-do
        end-do
        forall(k in SUPPLI) capacity(k):= integer(round(total/NWO)*(0.75+(.5*random)))

        forall(a in WORKORDERS)D_max(a):= integer(valid_serv(a)*30)

  ! Write data to file
    initializations to DATAFILE
        SUP valid_serv NWO SERV
        duration capacity score_matrix D_max distance_matrix
    end-initializations

end-model
```

REFERENCES

Ameri, F., & Dutta, D. (2006). An upper ontology for manufacturing service description, In Proceedings of ASME DETC 2006, Philadelphia, PA, United States.

Ameri, F., & McArthur, C. (2011). An Experimental Evaluation of a Rule-Based Approach to Manufacturing Supplier Discovery in Distributed Environments. Retrieved from http://www.kirkman-enterprises.com/sites/kirkman-enterprises.com/files/portfolio/publications/ASME-IDETC-2011-47768.pdf

Ameri, F., & Patil, L. (2012). Digital manufacturing market: a semantic web-based framework for agile supply chain deployment. Journal of Intelligent Manufacturing, 1–16.

Ameri, Farhad, & Dutta, D. (2008). A Matchmaking Methodology for Supply Chain Deployment in Distributed Manufacturing Environments. Journal of Computing and Information Science in Engineering, 8(1), 011002. doi:10.1115/1.2830849

Cayot, B. (2007). Strategic transportation sourcing: A comprehensive approach. PRTM, Retrieved http://www.prtm.com/uploadedFiles/Strategic_Viewpoint/Articles/Article_Content/PRTM_Strategic_Transportation_Sourcing.pdf

Çebi, F., & Bayraktar, D. (2003). An integrated approach for supplier selection. Logistics Information Management, 16(6), 395–400. doi:10.1108/09576050310503376

Chamodrakas, I., Batis, D., & Martakos, D. (2010). Supplier selection in electronic marketplaces using satisficing and fuzzy AHP. Expert Systems with Applications, 37(1), 490–498. doi:10.1016/j.eswa.2009.05.043

Desaulniers, G., Desrosiers, J., & Marius, M. S. (2005). Column generation. New York, NY: Springer Science Business Media Inc. Retrieved from http://books.google.com/books?id=ncD7qnkT-1EC&printsec=frontcover&source=gbs_atb

Dickson, G.W. (1966). An analysis of vendor selection systems and decisions, J. Purchasing, 2 1966(2), pp. 5–17

Ding, H., Benyoucef, L., & Xie, X. (2004). A multiobjective optimization method for strategic sourcing and inventory replenishment. In Proceedings of the IEEE international conference on robotics and automation (pp. 2711–2716).

Eaton, D. J., Daskin, M. S., Simmons, D., Bulloch, B., & Jansma, G. (1985). Determining emergency medical service vehicle deployment in Austin, Texas. Interfaces, 15(1), 96–108.

Fourer, R., Gay, D., & Kernighhan, B. (2002). AMPL A Modeling Language for Mathematical Programming (2nd Edition.). Duxbur Press.

Huo, H., & Wei, Z. (2008). Suppliers selection and order allocation in the environment of supply chain based on fuzzy multi-objective integer programming model (Vol. 2, pp. 2299–2304). Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4682919

Ko, C. S., Kim, T., & Hwang, H. (2001). External partner selection using tabu search heuristics in distributed manufacturing. International Journal of Production Research, 39(17), 3959–3974. doi:10.1080/00207540110072263

Konstantinos Kirytopoulos, Vrassidas Leopoulos, George Mavrotas, Dimitra Voulgaridou (2010). Multiple sourcing strategies and order allocation: an ANP-AUGMECON meta-model. Supply Chain Management: An International Journal, Vol. 15 Iss: 4, pp.263 – 276.

Messe Frankfurt Group. (2013). Apparel sourcing paris. Retrieved from http://apparelsourcing-fr.messefrankfurt.com/paris/en/exhibitors/welcome.html

MFG.com, Inc. (2013). CNC machining, injection molding and metal fabrication marketplace. Retrieved from http://www.mfg.com.

Min, H. (1994). International supplier selection: A multi-attribute utility approach. International Journal of Physical Distribution & Logistics Management, 24(5), 24–33.

Nydick, R. L., & Hill, R. P. (1992). Using the analytic hierarchy process to structure the supplier selection procedure. International Journal of Purchasing and Materials Management, 28(2), 31–36.

Ogtildeuz, O. (2002). Generalized column generation for linear programming. Management Science, 48(3), 444–452.

Rardin, R. (1998). Optimization in operations research. Upper Saddle River, New Jersey 07458: Prentice Hall, Inc.

Savelsbergh, M. (1997). A branch-and-price algorithm for the generalized assignment problem. Operations Research, 831–841.

Sanayei, A., Farid Mousavi, S., Abdi, M. R., & Mohaghar, A. (2008). An integrated group decision-making process for supplier selection and order allocation using multi-attribute utility theory and linear programming. Journal of the Franklin Institute, 345(7), 731–747. doi:10.1016/j.jfranklin.2008.03.005.

Shapiro, J. F. (2006). Modeling the Supply Chain (2nd ed.). Brooks Cole.

Roa, C.P. Kiser, G.E. (1980). Educational buyer's perception of vendor attributes, J. Purchasing Mater. Manage, 1980(16), pp. 25–30.

Taha, H. (2006). Operations research: An introduction (8th ed.). Upper Saddle River, NJ 07458: Pearson Prentice Hall.

Wu, J., Zhang, J., Zheng, C., & Xi, C. Y. (2110). The study on neural network-based supplier selection and evaluation (Vol. 1, pp. 31–34). Presented at the Third International Conference on Information and Computing. Retrieved from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5514244.

VITA

Hayden Beauchamp is graduate student in the Industrial Technology program at Texas State University-San Marcos. In 2013 he interned at T.B. Woods, a long-standing manufacturer of industrial power transmission couplings. He has previously been published in the field of human factors engineering and ergonomics. Following this, he spent two years conducting research in polymer nano-composites for the Institute of Industrial Science at Texas State University before starting graduate school.

Permanent Email Adress: hb1087@gmail.com

This thesis was typed by Hayden Beauchamp.