A Problem-Specific and Effective Metaheuristic For Flexibility Design

Michael Schneider, Jörn Grahl, David Francas

Working Paper 01/2009

June 2009

Working Papers in Information Systems and Business Administration

Johannes Gutenberg-University Mainz

Department of Information Systems and Business Administration

D-55128 Mainz/Germany

Phone +49 6131 39 22734, Fax +49 6131 39 22185

E-Mail: sekretariat[at]wi.bwl.uni-mainz.de

Internet: http://wi.bwl.uni-mainz.de

A Problem-Specific and Effective Metaheuristic for Flexibility Design

Abstract

Matching uncertain demand with capacities is notoriously hard. Operations managers can use mix-flexible resources to shift excess demands to unused capacities. To find the optimal configuration of a mix-flexible production network, a flexibility design problem (FDP) is solved. Existing literature on FDPs provides qualitative structural insights, but work on solution methods is rare. We contribute the first metaheuristic which integrates these structural insights and is specifically tailored to solve FDPs. Our genetic algorithm is compared to commercial solvers on instances of up to 15 demand types, resources, and 500 demand scenarios. Experimental evidence shows that in the realistic case of flexible optimal configurations, it dominates the comparison methods regarding runtime and solution quality.

1 Introduction

Demand uncertainty is commonplace in present markets for goods and services. The most important drivers for demand uncertainty are shortening product and service life cycles, high competition on saturated markets and increasing product and service variety. Unfortunately, operations managers have to commit to long-term resource capacity levels before demand realizes, for example when setting manufacturing plant or workforce capacities. These levels have to be matched with demand as closely as possible in order to avoid lost sales (when demand exceeds capacity) and unused capacity (when capacity exceeds demand).

Flexible resources are one means to encounter this problem. We refer the reader to Bertrand (2003) for a discussion of several flexibility types, their characteristics and benefits. The flexibility type relevant to this work is mix flexibility, i.e. the ability of a resource to process a variety of demand types without incurring high transition penalties (Koste and Malhorta 1999).

The following example illustrates how adding mix-flexibility to an inflexible resource can help to resolve demand-capacity mismatches. Consider two

resources 1,2 and two types of demand d_1, d_2. Both resources have a capacity of 100 units. Imagine a situation where demand is $d_1 = 150$ and $d_2 = 50$. If both resources are inflexible, resource i can process d_i only. This situation yields lost sales of 50 units at resource 1 and 50 units of unused capacity at resource 2. Now, if resource 2 is mix-flexible and can process both demand types, the 50 units of excess demand at resource 1 can be shifted to resource 2. We end up with no lost sales and two completely utilized resources.

A major application area of mix-flexibility lies in the automotive industry. The resource capacities that have to be set include production line and overall plant capacities. End-customer demand occurs for car models and is uncertain. Now, mix flexibility can either be used on production line level or plant level. Chrysler Group produces two entirely different products on the same production line (DaimlerChrysler News 2002) and Ford is able to manufacture nine different models in its Rouge plant (Mackintosh 2003). In the following, we adopt the terminology of the automotive industry and refer to demand types as product demands and to resources as plants. This shall increase the readability of the paper and does not restrict the validity of our findings to the automotive industry.

Flexibility decisions can be formalized by means of bipartite graphs. A bipartite graph consists of two disjoint sets of nodes. Arcs may only link two nodes from different sets. To model flexibility decisions, one set of nodes represents the products, the other set the plants. An arc between a product and a plant is called a link, and means that the plant is able to manufacture the product. If a plant is only connected to a single product, it is inflexible. For more than one linked product, it is mix-flexible. Figure 1 shows an example flexibility configuration represented as a bipartite graph.

What is the most profitable flexibility configuration a company can choose? To answer this question, the company has to solve a flexibility design problem (FDP). It consists of determining for each plant which products it shall be able to produce, i.e., which links in the bipartite graph must be set, such that total expected profits are maximized.

The next section reviews relevant literature and motivates our work. Section 3 gives a formal description of the FDP. Section 4 shows how to develop a customized GA (CGA) for solving this problem. Experimental results are presented in Section 5. The experiments evaluate the run-time and solution quality of CGA and several comparison methods. Section 6 concludes the work and gives a brief outlook on future research.

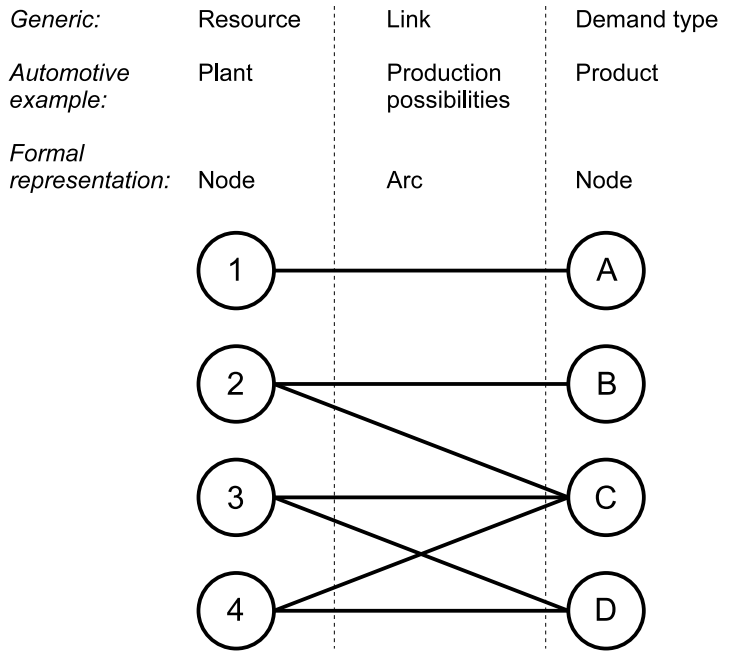

Figure 1: Example flexibility configuration with four products and four plants. Plant 1 is inflexible and dedicated to producing product A only. Plant 2 is able to produce products B and C, plant 3 is able to produce product C and D, plant 4 is able to produce products C and D. Plants 2,3 and 4 make use of mix-flexibility.

2 Motivation and literature

We motivate our work on the FDP via missing links between two strands of literature. The first strand centers around the structure of high-quality solutions of the FDP and was pioneered by Jordan and Graves (1995). The second strand of research aims at the development of practical planning models for strategic network design where flexibility design is accompanied by other strategic and tactical decisions, like capacity choice or work shift models.

Work on solution structure Jordan and Graves study the optimal assignment of products to plants in a stylized multi-product and multi-plant manufacturing network. They compare the flexibility benefits of different configurations in simulation studies, and use expected sales and expected capacity utilization as performance measures.

Central to their work is the concept of a *chain*. A chain is defined as a group of products and plants which are all connected, directly or indirectly,

by product assignment decisions. This means that the corresponding graph consists of exactly one component, i.e. the graph is connected.

They propose the following *flexibility principles*, which have heavily influenced academic research and industry practice: 1) Chains which encompass as many plants and products as possible are superior to configurations that contain several chains. 2) A complete chain which has all node of the graph in a cycle has virtually the same benefits as a totally flexible network. From a flexibility viewpoint, adding further links beyond a complete chain configuration is almost useless. 3) In general, there are several possibilities to configure a production network in a way that optimizes flexibility benefits. 4) When adding flexibility to an inflexible network, the demand for each product should be balanced with the capacity of the plants the product is directly connected to. Analogously, the capacity of each plant should be balanced with the demand of the products the plant is directly connected to.

Studies in other application areas have shown that the applicability of these design principles is not limited to strategic network planning but also holds for cross-training workers at production lines and call centers (see e.g. Brusco and Johns 1998, Hopp et al. 2004, Hopp and van Oyen 2004, Chevalier et al. 2004, Wallace and Whitt 2005) and designing queueing systems (see e.g. Gurumurthi and Benjaafar 2004, Andradóttir et al. 2007).

Iravani et al. (2005) develop domain-independent flexibility measures based on the number of demand shifting possibilities in a network. Their results underline the power of chain-like flexibility configurations. Analytical justification of chaining principles is given in Chou et al. (2009) and Aksin and Karaesmen (2007).

Work on practical planning approaches The FDP is a cornerstone of the strategic network planning of a company. However, apart from flexibility decisions alone, practical planning approaches typically consider additional issues. To name just a few examples, detailed operational work shift models (Bihlmaier et al. 2009), raw material supplier selection (Alonso-Ayuso et al. 2003), net present value based optimization (Fleischmann et al. 2006) and uncertain demand, costs and supply (Santoso et al. 2005) are included. Kauder and Meyr (2009) consider an extension of Fleischmann's deterministic planning model but restrict its solution space to chains. Flexibility is forced on solutions for its own sake, and the runtime increases when the solution space is restricted in this way. Generally, all resulting models are highly complex and require sophisticated solution methods. Typically, accelerated mixed-integer programming techniques like Benders Decomposition or Branch and Fix are used.

Let us now discuss shortcomings of the first strand. Chaining appears to be a reasonable and robust strategy for designing flexible networks. However, optimal solutions are only expected to be similar to chains, but not identical to chains. Knowing that flexible configurations are similar to chains does not suffice to solve the FDP to optimality. It is NP-hard even if product demands are certain (Garey and Johnson 1979). Despite its practical relevance and its challenging nature, no solution methods for this problem are given in these works.

Regarding the second strand of research, the planning methods described above require enormous computational effort. This limits the number of network designs practitioners can evaluate. Further, it puts a natural bound on the size of problems that can be solved in reasonable time. Efforts to lift this bound focus on accelerating the above mentioned mathematical programming methods. We are not aware of work that exploits the potential of using qualitative knowledge on partial aspects of the problems. In our case, the flexibility principles provide qualitative knowledge on how to solve the FDP. Yet, an integration of this knowledge into solution methods is still missing.

This paper makes the following contributions.

1. We bridge the gap between the described strands by integrating the flexibility principles and simple statistics, e.g., link ratios, into a metaheuristic solution approach for the FDP. The proposed genetic algorithm (GA) is, to the best of our knowledge, the first metaheuristic for this problem.
2. In designing the GA, we demonstrate how well accepted qualitative principles can be incorporated into a quantitative solution approach to increase its performance.
3. Our approach can handle problem sizes up to 15 products and plants and 500 scenarios which can no longer be solved by commercial optimization software in reasonable time. We use the solver Xpress-SP, which is specialized in solving stochastic programs. Our approach demonstrates superior performance and consistently finds flexibility configurations with higher expected profits in a fraction of the time required by Xpress-SP.
4. The approach is a starting point for intelligently solving more complex practical planning problems which include additional issues beyond flexibility decision.

3 Model description

We consider the FDP in a production network consisting of a set $I = \{1, .., n\}$ of products and a set $J = \{1, \ldots, m\}$ of plants. This problem can be described as two-stage mixed-integer linear stochastic program as follows. The first stage decides the flexibility design. A flexibility design is represented by binary variables $y_{ij}, i \in I, j \in J$. They are 1 if product i is assigned to plant j and 0 otherwise. Assigning product i to plant j raises investment costs c_{ij}. We assume that product demands are uncertain at the time decisions y_{ij} are made. This uncertainty is captured in a set of scenarios $S = \{1, \ldots, k\}$. A single scenario $s \in S$ has probability p^s and associated demand d_i^s for product i.

The second stage is a simple linear production planning problem. Its available production capabilities are determined by the first-stage decisions y_{ij}. The quantity of product i that is produced in plant j in scenario s is x_{ij}^s. Total production in plant j is limited by its capacity K_j. The π_{ij} are the associated profit margins. The second-stage recourse function is the expected profit $Q(y_{ij}, d_i^s)$ and depends on the first-stage decisions and the stochastic demands. The expected value is computed over all scenarios $s \in S$.

The overall objective is to determine a flexibility design y_{ij} that maximizes expected second-stage profits minus first-stage link investment costs. The objective function of the model is:

$$\max_{y_{ij}, i \in I, j \in J} \left\{ -\sum_{i \in I} \sum_{j \in J} c_{ij} \cdot y_{ij} + \sum_{s \in S} p^s \cdot Q(y_{ij}, d_i^s) \right\}, \tag{1}$$

with second-stage expected profit:

$$Q(y_{ij}, d_i^s) = \max_{x_{ij}^s} \sum_{i \in I} \sum_{j \in J} \pi_{ij} \cdot x_{ij}^s \tag{2}$$

$$\text{s.t.} \quad \sum_{i \in I} x_{ij}^s \leq K_j \qquad \forall j \in J, s \in S \tag{3}$$

$$\sum_{j \in J} x_{ij}^s \leq d_i^s \qquad \forall i \in I, j \in J, s \in S \tag{4}$$

$$x_{ij}^s \leq y_{ij} \cdot K_j \qquad \forall i \in I, j \in J, s \in S \tag{5}$$

$$x_{ij}^s \in \mathbb{R} \qquad \forall i \in I, j \in J, s \in S \tag{6}$$

Constraints (3) ensure that plant capacities are respected for all resources and scenarios. Constraints (4) state that no more units of a product can be sold than there is demand for the product. Demand may not be satisfied and unmet demand is lost. Constraints (5) ensure that each plant adheres to the production possibilities set on the first stage.

4 Genetic algorithm for the FDP

Genetic algorithms (Holland 1975) are population-based metaheuristics for solving combinatorial optimization problems. We develop a customized GA for the FDP which is based on Goldberg (1989)'s simple GA (SGA). The CGA design shall concentrate the search around highly profitable flexibility configurations.

Section 4.1 explains how solutions are represented and how their fitness is evaluated. The creation of the initial population is described in Section 4.2. It determines the expected structure of good solutions and seeds such solutions systematically. The crossover operator is outlined in Section 4.3. The operator combines good parts of flexibility configurations and avoids their disruption by restricting the crossover points. Section 4.4 explains the mutation operator. It randomly opens or closes links, but only if promising.

4.1 Solution representation and fitness evaluation

CGA makes the first-stage decisions of the FDP. Thus, chromosomes are binary strings of fixed length $l = m \cdot n$. The bit at position u is binary decision y_{ij} with $i = \lceil \frac{u}{m} \rceil$ and $j = u - m \cdot (i-1)$. For each chromosome k demand scenarios are generated, and the linear production programs on the second stage are solved to optimality using Xpress-Optimizer (Dash Optimization 2009b). The value of the recourse function is the average of the optimal profits. Subtracting first-stage link investment costs yields expected profit (the fitness) of a chromosome.

4.2 Initial population

CGA starts from an initial population seeded with good flexibility configurations. This population is put together from three preliminary populations $P_{1\ldots3}$, see Figure 2. The goal of the first population is to get a rough picture of promising structures, the second aims at replicating such structures, and the third adds diversity. We now describe these steps in detail.

P_1: Supply We create P_1 to get a rough picture of promising flexibility structures. It is filled by randomly rounding the optimal solution $\bar{y}_{i,j}, i \in I, j \in J$ of the linear relaxation of the FDP (see e.g. Srinivasan 1999). For each $\bar{y}_{i,j}$, we draw a random number r in $[\,0,1\,[$. If $r < \bar{y}_{i,j}$, we round $\bar{y}_{i,j}$ to 1, and to 0 otherwise. The solution is then mapped onto a binary string as described in Section 4.1.

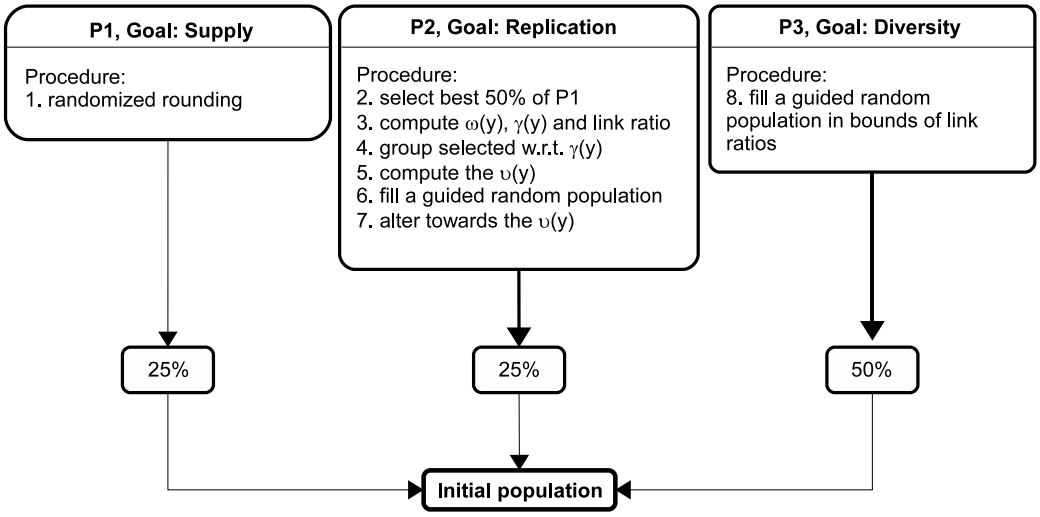

Figure 2: Flow-chart for creating the initial population of the CGA.

P_2: Replication P_2 shall be filled with solutions similar to the best of P_1. To this end, we select the better half of P_1 into a population $\bar{P}_1$ and compute the following indicators for all $y \in \bar{P}_1$: 1) A weight $\omega(y)$ which depends on the expected profit of y, 2) the number of components $\gamma(y)$ of the bipartite graph of y, and 3) a link ratio.

$\omega(y)$ relates the expected profit $F(y)$ to the expected profit of other solutions. The higher $\omega(y)$ the more outstanding its expected profit. Let F_{worst} denote the smallest expected profit in $\bar{P}_1$. Then, $\omega(y)$ is defined as

$$\omega(y) = \frac{F(y) - F_{worst}}{\sum_{k \in \bar{P}_1} F(k) - F_{worst}}. \tag{7}$$

The number of network components $\gamma(y)$ is obtained from y's bipartite graph. It roughly indicates to which degree flexibility and chaining structures are present in the network. $\gamma(y)$ can be calculated with depth-first search (see e.g. Cormen et al. 2001) or using our algorithm given in Appendix 6. It works directly on a chromosome and uses bit-wise operations only which allows for a very efficient implementation.

The link ratio is the number of links that are set in a solution divided by the number of possible links $n \cdot m$ and is a rough indicator for the degree of flexibility present in a solution. We now describe how $\omega(y)$, $\gamma(y)$ and the link ratio are used to construct P_2.

Grouping First, we build groups of individuals according to their number of components e and sum up the weights $\omega(y)$ of all solutions inside each

group to $\vartheta(e)$:

$$\vartheta(e) = \sum_{y \in \bar{P}_1, \gamma(y)=e} \omega(y). \tag{8}$$

The $\vartheta(e)$ are aggregate measures for the impact of flexibility on expected profit. If e.g. $\vartheta(1) \approx 1$, then graphs with only one component (1-chains) perform very well in $\bar{P}_1$. It is then likely that other highly-profitable solutions are also 1-chains.

Sampling We mirror the structures of solutions in $\bar{P}_1$ by filling P_2 so that it contains approximately $\vartheta(e) \cdot |P_2|$ solutions for each component number e. We achieve this by first filling P_2 with random solutions that are structurally similar to those in $\bar{P}_1$.

To create the random solutions, we do not use the standard approach which is to distribute 0s and 1s uniformly over binary chromosomes. This would lead to an expected number of $0.5 \cdot n \cdot m$ links. Remember, that adding links beyond a complete chain has only very limited benefits. A complete chain has $2 \cdot n = 2 \cdot m$ links if $n = m$, i.e., substantially fewer than $0.5 \cdot n \cdot m$ links are set when $n, m \geq 5$. Thus, uniformly distributing 0s and 1s introduces too many links. The magnitude of this effect aggravates for growing n and/or m.

To control the number of links, we change the probability of setting a link from 0.5 to a parameter g, which leads to an expected number of $g \cdot n \cdot m$ links. This means, that g is the expected link ratio of a created solution. Setting g biases the search towards a certain degree of flexibility. For example, $g = 1/m$ on average supplies the number of links to form a completely inflexible configuration, whereas $g = 2/m$ on average supplies the number of links required to form a complete chain.

To mirror the diversity of flexibility structures in $\bar{P}_1$, we set a lower bound $\underline{g} = \min\{\mathit{linkRatio}(y) | y \in \bar{P}_1\}$ on g. The respective upper bound is $\overline{g} = \max\{\mathit{linkRatio}(y) | y \in \bar{P}_1\}$. The value of g is then uniformly sampled from $[\underline{g}; \bar{g}]$. Note that random sampling cannot guarantee a certain network structure. Setting $g = \underline{g}$ may not result in an inflexible configuration, setting $g = \bar{g}$ may not form a 1-chain. It only regulates link supply, while the formation of desired structures is achieved by crossover and mutation.

After the solutions have been sampled using g, we modify them such that the proportions of each number of components e are approximately $\vartheta(e) \cdot |P_2|$. We transform solutions by adding links and thus reducing the number of components. Links are added in a way that balances product demands and plant capacities (Jordan and Graves 1995), i.e. we successively add links

from the component with the highest excess demand to the component with the highest excess capacity.

P_3: Adding diversity The randomized rounding used to create P_1 potentially produces many similar or even identical solutions. In P_2 the flexibility characteristics of the best solutions in P_1 are mirrored. To diversify, we generate P_3 by the sampling technique used to generate P_2. However, $\underline{g}$ ($\bar{g}$) is set to the minimal (maximal) link ratio of the best $\frac{1}{2}$ of individuals in P_2.

Finally, we build a single population P by merging the best $\frac{1}{4}$ of P_1 and P_2 and the best $\frac{1}{2}$ of P_3. The fraction of P_3 is set this high in order to increase diversity.

4.3 Customized crossover

We present a crossover operator more adequate for the FDP than the one-point or uniform crossover used in standard GAs. The latter operators break good solutions at arbitrary positions. Consider the problem instance in Figure 3 with three products and plants. The chromosomes are divided into three groups, so that each group contains the plant assignment decisions of a product. Crossover points are vertical lines (|). Assume that the plant assignment for the second product (bits four to six) of Parent 1 is optimal. After applying one-point crossover, it is not present in any offspring. To avoid this problem, we use a one-point crossover that cannot cut the plant assignment of any product but only at product boundaries, i.e. only crossover points $r \in \{t \cdot m \mid t = 1, \ldots, n-1\}$ are possible. In the following, this customized one-point crossover is referred to as one-point crossover, the one-point crossover used in SGA is referred to as standard one-point crossover.

Note that the one-point crossover is not able to introduce new plant assignments into the population. Sufficient supply of such assignments must be assured by an initial population of adequate size and the mutation operator. The crossover operator recombines these assignments to evolve better solutions.

4.4 Customized mutation

We customize bit-flip mutation to prevent CGA from searching in low-quality areas of the search space. To this end, the minimal and maximal link ratio for the best 30% of the population is determined at the beginning of each generation, denoted as $linkRatio^{min}$ and $linkRatio^{max}$. If an individual has a link ratio greater than $linkRatio^{max}$, mutation does not open another link.

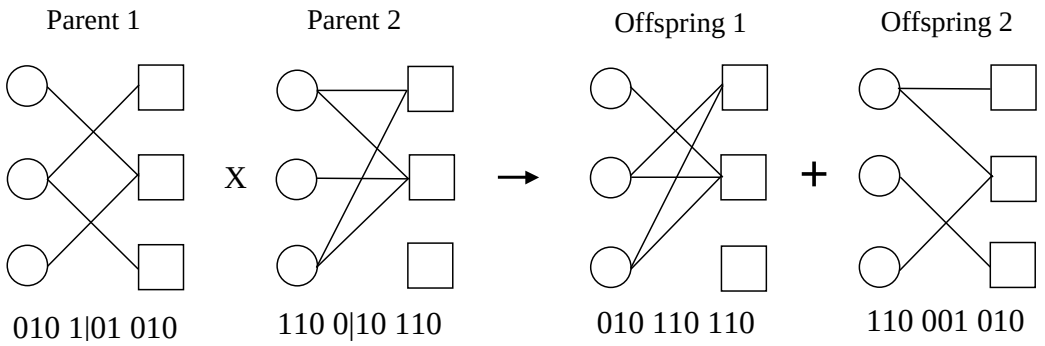

Figure 3: Application of one-point crossover at arbitrary positions can destroy high-quality partial solutions.

Analogously, mutation does not close a link in an individual that already has a lower link ratio than $linkRatio^{min}$.

Mutation probability is $1/2l$, but to prevent premature convergence we adaptively enlarge it after the best found solution has not improved over a certain number of generations. The number of generations without improvement that causes the GA to terminate is denoted by ϱ. After $0.5 \cdot \varrho, 0.8 \cdot \varrho$, and $0.9 \cdot \varrho$, the mutation probability is scaled by a factor α.

5 Numerical Studies

The numerical studies are subdivided into preliminary tests and the main study. The former investigate the impact of flexibility present in optimal solutions on the performance of the following comparison methods: 1) CGA as presented in Section 4, 2) SGA, 3) Xpress-Optimizer, a solver that integrates exact solution routines for mixed-integer programs (X-O), 4) Xpress-SP (Dash Optimization 2009c), a solver integrating exact methods for optimization problems involving uncertainty (X-SP).

We found that problem instances with inflexible optimal solutions are relatively easy to solve for the exact methods (X-O, X-SP). As soon as the solutions exhibit some flexibility, the performance of solution methods depends mainly on the size of a problem. Therefore, the main study examines the effect of an increasing number of products, plants and/or scenarios on the performance of all comparison methods.

Implementation details and method parameters are explained in Section 5.1. The preliminary tests and the main studies are in Sections 5.2 and 5.3. The results of the main study are presented and discussed in Section 5.4.

5.1 Implementation details and method parameters

We first describe implementation details and method parameters identical in CGA and SGA. Both algorithms are implemented from scratch in Java. The models for evaluating the expected profit of an individual and for computing the relaxed solution (required for creating the initial CGA solutions, see Section 4.2) are written in Xpress Mosel and are integrated into the Java code using the Mosel Java Library (Dash Optimization 2009a).

SGA and CGA use the representation and evaluation scheme described in Section 4.1. Their population size is $5 \cdot n \cdot m$. Both select the best $\lfloor 0.5 \cdot popSize \rfloor_{\mathsf{mod2}}$ individuals at the beginning of a generation into the mating pool, and maintain the best $\lfloor 0.1 \cdot popSize \rfloor$ individuals for the next generation.

Two random parents from the mating pool are crossed with a probability of 0.8 and put directly into the offspring pool otherwise. Mutation is applied on individuals of the offspring pool. The probability for a bit in a chromosome to be mutated is $\frac{1}{2l}$. Both GAs terminate after at most 200 generations or if the best solution found has not improved for more than $\varrho = 50$ generations. Due to computational limitations we restrict the number of CGA and SGA runs per instance to five.

Details specific to CGA or SGA follow. The initial population of SGA is uniformly sampled and standard one-point crossover and bit-flip mutation are applied. CGA uses crossover and mutation as outlined in Section 4. The adaptation factor for adaptive mutation in CGA is $\alpha = 2$.

X-O and X-SP have a maximum runtime that is the sum of all five CGA run-times. Finally, all (pseudo) random number generators are initialized with fixed random seeds to make the results reproducible.

5.2 Preliminary tests

The preliminary tests examine the impact of the degree of flexibility in optimal solutions on the performance of the solution methods. It is well-known, that low demand variability and high plant capacities (Jordan and Graves 1995) as well as positively correlated demand (Eppen 1979) and low profit margins with high link costs (Fine and Freund 1990, Van Mieghem 1998) decrease the value of flexibility.

We use these insights to generate a series of smaller instances with varying degrees of flexibility in the optimal solutions. A comparison of the performance of all solution methods revealed that X-O and X-SP can quickly find the optimal solutions if the optimum has little flexibility. The reason is that flexible solutions quickly become too expensive because of the cost of additional links. The solution space can be strongly reduced, and many branches

in the branch and bound tree can be cut. In this case, CGA also delivers excellent results in terms of optimality gaps, but its use is not necessary since the exact methods are able to deliver optimal solutions faster.

For instances where the optimal solution is more flexible, this clear-cut ranking of solution methods is not possible. In this case, problem size is the major discriminating factor for the comparison methods.

5.3 Main study

The main study investigates the impact of problem size on the performance of the solution methods. To this end, we generate problem instances with a small, medium and high $(m, n = 5; m, n = 10; m, n = 15)$ number of products m and plants n combined with a small, medium and high $(k = 50, 200, 500)$ number of scenarios. For each of the nine combinations, we create three instances, resulting in 27 instances total.

Problem parameters are chosen in a way that optimal solutions are likely to exhibit flexibility, i.e. they are not inflexible. Apart from reasons put forward in Section 5.2, this is desirable because one can expect practical systems to exhibit some degree of flexibility. Problem parameters are chosen as follows.

Link costs c_{ij} are uniformly drawn from $[150, 300]$.

Profit margins π_{ij} are uniformly drawn from $[20, 40]$.

Product demands are uncorrelated and normally distributed with expected value μ_i (uniformly distributed in $[60, 140]$) and standard deviation $\sigma_i = \mu_i * 0.3$. Differing amounts of total expected demand $\sum_{i=1}^{n} \mu_i$ in the instances causes undesirable side-effects. To avoid them, we let the expected demands sum up to the number of products multiplied by 100, i.e. $\sum_{i=1}^{n} \mu_i = n * 100$. This is achieved by the following procedure:

1. Start with an empty list L of expected demand values.
2. Draw an expected demand value μ from the interval $[60, 140]$ and add μ and $\hat{\mu} = 200 - \mu$ to L.
3. Repeat Step 2 $\lfloor \frac{n}{2} \rfloor$ times. If n is odd, add $\mu = 100$ to L.
4. Shuffle L and assign the i^{th} value in the list to μ_i.

The product demands d_{is} for the different scenarios $s \in S$ are then generated from μ_i and σ_i by means of descriptive sampling, see e.g. (Pidd 1984).

Plant capacities K_j, $j \in J$ shall be random but shall stay in the bound of the total expected product demand, i.e. $\sum_{j=1}^{m} K_j = n * 100$. This is accomplished as follows:

1. Start with an empty list C of plant capacities.
2. Determine the average plant capacity $\bar{c} = \frac{n \cdot 100}{m}$.
3. Draw a plant capacity value c from the interval $[0.7 \cdot \bar{c},\ 1.3 \cdot \bar{c}]$ and add c as well as $\hat{c} = 2 \cdot \bar{c} - c$ to C.
4. Repeat Step 2 $\lfloor \frac{m}{2} \rfloor$ times. If m is odd, add $c = \bar{c}$ to C.
5. Shuffle C and assign the j^{th} value in the list to K_j.

In this way, we generate 27 problem instances that are solved by CGA, SGA, X-SP and X-O.

5.4 Results

All experiments are conducted on a standard desktop PC with AMD Athlon 64 CPU at 2.41 GHz and 3.25 GB of RAM, running Windows XP Professional. A complete listing of results is given in Table 1. We show the best solution found by X-O and X-SP. If this solution is found by X-O, this is indicated by a diamond ($\diamond$), for X-SP by a dagger ($\dagger$). For X-O and X-SP, the tables also show the upper bound, the gap between the best solution found and the upper bound (denoted as optimality gap) and the run-time in minutes. For the GAs, the tables show the average, best and worst solution found as well as the average run-time in minutes. For all solution methods, we provide the gap to the best solution found by any of the methods for the respective instance. For the GAs, the average solution is used to determine this gap.

Comparison of CGA and SGA Regarding expected profit, the average solution found by CGA is better than that of SGA for all instances. The same is true even for the worst solutions found by the CGA for all but one problem instance.

Regarding runtime, SGA is only faster than CGA for the instances with 5 products, 5 plants and 50/200 scenarios. The CGA runs significantly faster than SGA for all other instances: for the medium-sized and large problems CGA is approximately twice as fast as SGA. This can be explained by SGA breaking chromosomes at arbitrary positions and having a very poor initial population for problems with a high number of products and plants (see Section 4.2). A blind crossover operator and an unfocused initial population slow down convergence and increase the number of evaluations required.

To sum up, CGA shows clearly better performance than SGA, especially in those studies that are important in practice: instances that have a high complexity due to the number of products, plants and/or a high number of scenarios.

		$k = 50$			$k = 200$			$k = 500$		
$m = n = 5$		Inst. 1	Inst. 2	Inst. 3	Inst. 4	Inst. 5	Inst. 6	Inst. 7	Inst. 8	Inst. 9
X-SP (†)	Best Sol.	13282†	13204†	12448$^{\dagger\diamond}$	13237†	13093†	12352$^{\diamond}$	13175†	13049 †	12332$^{\diamond}$
X-O ($\diamond$)	Opt. Gap (%)	-	-	-	-	-	-	1.37	1.39	1.07
	UB	-	-	-	-	-	-	13355	13231	12466
	Gap (%)	0	0	0	0	0	0	0	0	0
	Time (min)	0.6	0.3	0.5	7.2	5.2	5.4	11.5	9.5	11.5
CGA	Avg. Sol.	13247	13141	12440	13237	13031	12339	13166	13026	12331
	Best Sol.	13282	13144	12448	13237	13034	12352	13189	13026	12332
	Worst Sol.	13222	13138	12427	13237	13022	12331	13143	13026	12328
	Gap (%)	0.26	0.48	0.07	0	0.47	0.10	0.06	0.17	0.01
	Time (min)	0.6	0.5	0.6	1.8	1.8	1.9	2.3	1.9	2.3
SGA	Avg. Sol.	13212	13098	12402	13216	13016	12324	13151	13001	12306
	Best Sol.	13222	13138	12448	13237	13048	12352	13174	13049	12332
	Worst Sol.	13181	13046	12346	13184	12988	12261	13133	12983	12274
	Gap (%)	0.53	0.80	0.37	0.16	0.59	0.23	0.17	0.37	0.21
	Time (min)	0.4	0.3	0.4	1.2	1.3	1.1	3.8	3.4	3.0
$m = n = 10$		Inst. 10	Inst. 11	Inst. 12	Inst. 13	Inst. 14	Inst. 15	Inst. 16	Inst. 17	Inst. 18
X-SP (†)	Best Sol.	26722$^{\diamond}$	27122†	26592†	26708$^{\diamond}$	27086$^{\diamond}$	26421$^{\diamond}$	26620†	26979†	26251†
X-O ($\diamond$)	Opt. Gap (%)	2.41	2.35	2.37	2.79	2.50	2.81	3.09	2.67	3.20
	UB	27381	27775	27238	27474	27780	27186	27469	27718	27120
	Gap (%)	0.29	0.51	0.38	0.51	0.70	0.38	0.57	0.67	1.03
	Time (min)	29.0	27.5	22.5	130.0	116.5	132.5	360.0	337.5	304.0
CGA	Avg. Sol.	26798	27260	26693	26844	27276	26521	26772	27159	26521
	Best Sol.	26863	27334	26723	26893	27308	26584	26817	27180	26556
	Worst Sol.	26763	27204	26648	26800	27211	26427	26736	27121	26512
	Gap (%)	0	0	0	0	0	0	0	0	0
	Time (min)	5.8	5.4	4.5	26	23.3	26.5	72	67.5	60.8
SGA	Avg. Sol.	26577	26879	26283	26614	26971	26214	26469	26803	26147
	Best Sol.	26659	27002	26329	26775	27131	26318	26574	27010	26229
	Worst Sol.	26450	26734	26241	26358	26740	26102	26345	26658	26016
	Gap (%)	0.83	1.40	1.54	0.86	1.12	1.16	1.13	1.31	1.41
	Time (min)	9.4	9.4	9.5	47.4	42.5	44.3	136.7	130	130
$m = n = 15$		Inst. 19	Inst. 20	Inst. 21	Inst. 22	Inst. 23	Inst. 24	Inst. 25	Inst. 26	Inst. 27
X-SP (†)	Best Sol.	40780$^{\diamond}$	40332†	34189†	40614$^{\diamond}$	40194†	34049$^{\diamond}$	40464†	40018†	33897$^{\diamond}$
X-O ($\diamond$)	Opt. Gap (%)	2.71	3.09	3.57	3.00	3.44	3.85	3.25	3.67	4.26
	UB	41918	41618	35454	41871	41628	35413	41824	41542	35405
	Gap (%)	0.37	0.48	0.27	0.52	0.76	0.45	0.75	0.93	0.86
	Time (min)	165	190	172	805	835	809	2560	2800	2662
CGA	Avg. Sol.	40931	40525	34280	40825	40500	34204	40770	40395	34192
	Best Sol.	40976	40593	34353	40867	40551	34267	40815	40467	34240
	Worst Sol.	40888	40472	34221	40765	40460	34165	40711	40328	34162
	Gap (%)	0	0	0	0	0	0	0	0	0
	Time (min)	33	38	34	161	167	162	512	560	473
SGA	Avg. Sol.	40314	39875	33680	40366	39769	33609	40072	39743	33675
	Best Sol.	40475	40079	33755	40599	39889	33704	40157	39858	33796
	Worst Sol.	39946	39731	33614	40258	39488	33540	40001	39620	33539
	Gap (%)	1.51	1.60	1.75	1.12	1.80	1.74	1.71	1.61	1.51
	Time (min)	77	79	70	396	403	371	1247	1278	1272

Table 1: Results of main study: solution values, runtime, and gaps for all methods and instances. Instances 1-9 have $m = n = 5$ products and plants, instances 10-18 have $m = n = 10$, instances 19-27 have $m = n = 15$. *Opt. Gap (%)* denotes the percentage deviation of the best solution to the upper bound. *Gap (%)* denotes the percentage deviation to the overall best solution of the instance.

Comparison of CGA and X-O/X-SP X-O/X-SP is not able to find the optimal solution in the given time limit in any but the 6 small instances with 5 products, 5 plants and 50/200 scenarios. The best found solution of CGA is optimal for four of these instances, the average solution is optimal for one instance.

For the small instances with 500 scenarios, the X-O/X-SP solution beats the average solution of CGA, however, the best solution found by CGA is better or equal than the X-O/X-SP solution for 2 of the 3 instances.

For the medium-sized and large problems, the average solution of CGA is better than the solution found by X-O/X-SP for all instances. More impressive, the worst solution of CGA is still better. The medium-sized and large problem instances are application areas of CGA. Exact solution methods are no longer able to find the optimal solution in adequate time for these instances. The dominance of CGA for this type of problems is illustrated in Figure 4. It pictures solution quality and the run-time of CGA and X-O/X-SP for a problem with 15 products, 15 plants and 50, 200 and 500 scenarios. This figure is typical for our results on medium and large instances. The average solution quality of CGA lies strictly above the best solution found by X-O/X-SP, while the average CGA run-time is five times below the time limit given X-O/X-SP.

Note that, contrary to CGA, SGA is not able to match the solution quality of X-O/X-SP for any of the 27 problem instances.

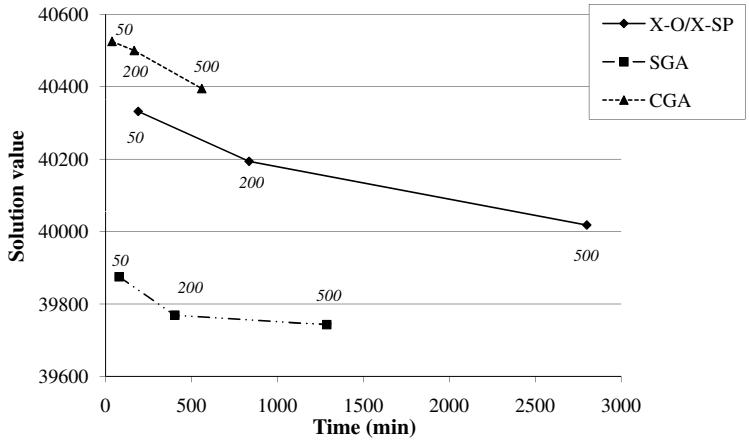

Figure 4: CGA vs. X-O/X-SP: 15 products, 15 plants with 50, 200, 500 scenarios

6 Conclusions

This paper presents the first metaheuristic solution approach to the FDP. CGA successfully integrates qualitative knowledge on the FDP to guide its search. It uses a customized technique to generate its initial population, which is based on the estimation and replication of good flexibility structures. The crossover operator avoids the disruption of good partial solutions, while mutation randomly opens or closes links when appropriate. We conducted experiments on instances up to 15 products and plants and 500 scenarios and

compared the solution quality and runtime of CGA and three comparison methods: SGA, X-O and X-SP.

The results show clearly that the customization of CGA is beneficial. A standard GA cannot compete with commercial solvers, while CGA is able to obtain better solutions in a fraction of the time required by commercial solvers under certain conditions. If the problem instance is small and/or its optimum is inflexible, X-O/X-SP provides the optimum quickly. However, medium-sized and large instances of the FDP whose optimal solution is flexible are of primary interest in practice. In these cases, X-O/X-SP are unable to find the optimal solution in reasonable time. CGA dominates them with respect to both runtime and solution quality.

Future research is to integrate CGA into larger planning models which include plant capacity choices. The crux is to gain qualitative insights into the interplay between the two types of decisions and to examine how this knowledge can be used in a solution approach. Furthermore, we noted in the experiments that CGA spends most of its time evaluating the second stage production program. It shall be investigated how its resolution can be speed up, e.g. by using iterated dual reoptimization.

Appendix: Determination of the number of components

The following algorithm determines the structure of the production network that corresponds to a first-stage solution y_{ij}, $i \in \{1,..,n\}$, $j \in \{1,..,m\}$. The algorithm is given y_{ij} in the form of n product strings that represent the plant assignment of the respective product:

$$productString[i] = y_{i1},..,y_{im}.$$

The algorithm returns a set that contains the components that constitute the network graph. A component consists of a list of the products it contains (*productList*) and a bitstring that represents the contained plants (*plantString*). Obviously, the number of components can be determined as the cardinality of the returned component set. The pseudocode of the algorithm is shown in Figure 5.

```
NETWORK-STRUCTURE(productStrings[1..n])
 1  Components = { }
 2  for j ← 1 to n
 3        do
 4           c ← new Component
 5           c.productList ← ⟨j⟩
 6           c.plantString ← productStrings[j]
 7           forall k ∈ Components
 8                 do
 9                    if c.plantString ∧ k.plantString ≠ 0
10                       then temp ← remove k from Components
11                            add products in c.productList to temp.productList
12                            temp.plantString ← temp.plantString ∨ c.plantString
13                            c ← temp
14           add c to Components
15  return Components
```

Figure 5: Procedure NETWORK-STRUCTURE

The following example with four products and four plants illustrates the

algorithm. We are given:

$$y_{ij} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The rows $i \in \{1, \ldots, n\}$ in the matrix represent the product strings. We start with an empty set of components $\mathit{Components} = \{\ \}$. We create a component with the first product string (Lines 4–6): $c_1 = (\langle 1 \rangle,\ \mathtt{1\,1\,0\,0})$ and add it to *Components* in Line 14, i.e. $\mathit{Components} = \{c_1\}$. Next, we create a component with the second product string $c_2 = (\langle 2 \rangle,\ \mathtt{0\,1\,1\,0})$. For each component in *Components*, we check whether a bitwise $\wedge$ between the plant string of the component from *Components* and the plant string of c_2 is different from zero (Lines 7–9):

$$c_1.\mathit{plantString} \wedge c_2.\mathit{plantString} = \mathtt{1\,1\,0\,0} \wedge\ \mathtt{0\,1\,1\,0} = \mathtt{0\,1\,0\,0} \neq \mathtt{0\,0\,0\,0}.$$

As the result is different from zero, we remove c_1 from *Components* and merge c_1 and c_2 (Lines 10–13) so that $c_2 = (\langle 1, 2 \rangle,\ \mathtt{1\,1\,1\,0})$. c_2 is added to *Components*, i.e. $\mathit{Components} = \{c_2\}$. Next, we create the component $c_3 = (\langle 3 \rangle,\ \mathtt{0\,1\,0\,0})$ and perform the bitwise $\wedge$ operation:

$$c_2.\mathit{plantString} \wedge c_3.\mathit{plantString} = \mathtt{1\,1\,1\,0} \wedge\ \mathtt{0\,1\,0\,0} = \mathtt{0\,1\,0\,0} \neq \mathtt{0\,0\,0\,0}.$$

We remove c_2 from *Components* and merge c_2 and c_3 to $c_3 = (\langle 1, 2, 3 \rangle,\ \mathtt{1\,1\,1\,0})$, which is added to *Components*. Next, we create the component $c_4 = (\langle 4 \rangle,\ \mathtt{0\,0\,0\,1})$. Since $c_3.\mathit{plantString} \wedge c_4.\mathit{plantString} = \mathtt{1\,1\,1\,0} \wedge\ \mathtt{0\,0\,0\,1} = \mathtt{0\,0\,0\,0}$, the component c_4 is added to *Components*, which yields the final result:

$$\mathit{Components} = \{(\langle 1, 2, 3 \rangle,\ \mathtt{1\,1\,1\,0}),\ (\langle 4 \rangle,\ \mathtt{0\,0\,0\,1})\}.$$

The production network for the given y_{ij} is depicted in Figure 6 and confirms our result.

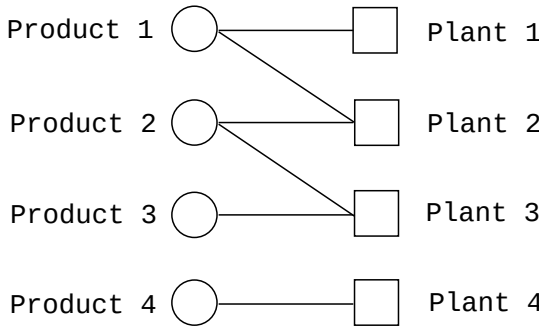

Figure 6: Production network for given y_{ij}

References

Aksin, O. Z. and Karaesmen, F.: 2007, Characterizing the performance of process flexibility structures, *Operations Research Letters* **35**(4), 477–84.

Alonso-Ayuso, A., Escudero, L. F., Garín, A., Ortuño, M. T. and Pérez, G.: 2003, An approach for strategic supply chain planning under uncertainty based on stochastic 0-1 programming, *Journal of Global Optimization* **26**(1), 97–124.

Andradóttir, S., Ayhan, H. and Down, D. G.: 2007, Compensating for failures with flexible servers, *Operations Research* **55**(4), 753–768.

Bertrand, J.: 2003, Supply Chain Design: Flexibility Considerations, *in* A. de Kok and S. Graves (eds), *Supply Chain Management*, Vol. 11 of *Handbook in Operations Research and Management Sciences*, Elsevier, chapter 4, pp. 133–198.

Bihlmaier, R., Koberstein, A. and Obst, R.: 2009, Modeling and optimizing of strategic and tactical production planning in the automotive industry under uncertainty, *OR Spectrum* **31**(2), 311–336.

Brusco, M. J. and Johns, T. R.: 1998, Staffing a multiskilled workforce with varying levels of productivity: An analysis of cross-training policies, *Decision Sciences* **29**(2), 499–515.

Chevalier, P., Shumsky, R. A. and Tabordon, N.: 2004, Routing and staffing in large call centers with specialized and fully flexible servers, *Technical report*, Simon Graduate School of Business, University of Rochester.

Chou, M. C., Chua, G. A., Teo, C.-P. and Zheng, H.: 2009, Design for process flexibility: Efficiency of the long chain and sparse structure. Operations Research, Articles in Advance.
URL: *http://or.journal.informs.org/cgi/content/abstract/opre.1080.0664v1*

DaimlerChrysler News: 2002, Chrysler Group's Windsor assembly plant launches next phase of flexible manufacturing.

Dash Optimization: 2009a, Xpress-Mosel.
URL: *http://www.dashoptimization.com/home/products/products_mosel.html*

Dash Optimization: 2009b, The Xpress Optimizer.
URL: *http://www.dashoptimization.com/home/products/products_optimizer.html*

Dash Optimization: 2009c, Xpress-SP.
URL: *http://www.dashoptimization.com/home/products/products_sp.html*

Eppen, G. D.: 1979, Effects of centralization on expected costs in a multi-location newsboy problem, *Management Science* **25**(5), 498–501.

Fine, C. and Freund, R.: 1990, Optimal investment in product flexible manufacturing capacity, *Management Science* **36**(4), 449–466.

Fleischmann, B., Ferber, S. and Henrich, P.: 2006, Strategic planning of BMW's global production network, *Interfaces* **36**(3), 194–208.

Garey, M. and Johnson, D.: 1979, *Computers and Intractability : A Guide to the Theory of NP-Completeness*, W.H. Freeman and Co.

Goldberg, D. E.: 1989, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley.

Gurumurthi, S. and Benjaafar, S.: 2004, Modeling and analysis of flexible queueing systems, *Naval Research Logistics* **51**, 755–782.

Holland, J. H.: 1975, *Adaptation in Natural and Artificial Systems*, University of Michigan Press.

Hopp, W. J., Tekin, E. and van Oyen, M. P.: 2004, Benefits of skill chaining in serial production lines with cross-trained workers, *Management Science* **50**(1), 83–98.

Hopp, W. J. and van Oyen, M. P.: 2004, Agile workforce evaluation: A framework for cross-training and coordination, *IIE Transactions* **36**, 919–940.

Iravani, S. M., van Oyen, M. P. and Sims, K. T.: 2005, Structural flexibility: A new perspective on the design of manufacturing and service operations, *Management Science* **51**(2), 151–166.

Jordan, W. C. and Graves, S. C.: 1995, Principles on the benefits of manufacturing process flexibility, *Management Science* **41**(4), 577–594.

Kauder, S. and Meyr, H.: 2009, Strategic network planning for an international automotive manufacturer, *OR Spectrum* **31**(3), 507–532.
URL: *http://dx.doi.org/10.1007/s00291-009-0171-x*

Koste, L. L. and Malhorta, M. K.: 1999, A theoretical framework for analyzing the dimensions of manufacturing flexibility, *Journal of Operations Management* **18**(1), 75–93.

Mackintosh, J.: 2003, Ford learns to bend with the wind, *Financial Times*. February, 14.

Pidd, M.: 1984, *Computer simulation in management science*, John Wiley & Sons, Inc.

Santoso, T., Ahmed, S., Goetschalckx, M. and Shapiro, A.: 2005, A stochastic programming approach for supply chain network design under uncertainty, *European Journal of Operational Research* **167**(1), 96–115.

Srinivasan, A.: 1999, Approximation algorithms via randomized rounding: A survey, *Series in Advanced Topics in Mathematics, Polish Scientific Publishers PWN*, pp. 9–71.

Van Mieghem, J. A.: 1998, Investment strategies for flexible resources, *Management Science* **44**(8), 1071–1078.

Wallace, R. B. and Whitt, W.: 2005, A staffing algorithm for call centers with skill-based routing, *Manufacturing & Service Operations Management* **7**(4), 276–294.