DEPARTEMENT TOEGEPASTE
ECONOMISCHE WETENSCHAPPEN

RESEARCH REPORT 9938

A HYBRID GENETIC ALGORITHM FOR SOLVING A LAYOUT PROBLEM IN THE FASHION INDUSTRY

by
J. MARTENS
F. PUT
S. VIAENE
J. VAN BRECHT

D/1999/2376/38

A Hybrid Genetic Algorithm for Solving a Layout Problem in the Fashion Industry

Martens J.[*],

Put F., Viaene S. & Van Brecht J.

Catholic University of Leuven

Faculty of Economics and Applied Economics

Information Systems Group

[*] Corresponding author, Jurgen.Martens@econ.kuleuven.ac.be

A Hybrid Genetic Algorithm for Solving a Layout Problem in the Fashion Industry

Abstract: *As of this writing, many success stories exist yet of powerful genetic algorithms (GAs) in the field of constraint optimisation. In this paper, a hybrid, intelligent genetic algorithm will be developed for solving a cutting layout problem in the Belgian fashion industry. In an initial section, an existing LP formulation of the cutting problem is briefly summarised and is used in further paragraphs as the core design of our GA. Through an initial attempt of rendering the algorithm as universal as possible, it was conceived a threefold genetic enhancement had to be carried out that reduces the size of the active solution space. The GA is therefore rebuilt using intelligent genetic operators, carrying out a local optimisation and applying a heuristic feasibility operator. Powerful computational results are achieved for a variety of problem cases that outperform any existing LP model yet developed.*

Keywords: *Genetic Algorithms, Layout Problem, Constraint Handling*

1. Introduction

As of this writing, genetic algorithms (GAs) have demonstrated their potential for a large set of hard, intractable optimisation problems. Recently, there have been successful research attempts to apply GAs for solving a wide number of well-known constraint satisfaction problems, eg. the job scheduling [4] and the transportation problem [9], the travelling salesman dilemma [22] and the knapsack [2,18] and the set covering problem [1,3].

In this paper, we propose a genetic algorithm to solve a cutting layout problem in the Belgian clothing industry. For several years now, exclusive clothing manufacturers are coping with the problem of working out an optimal cutting pattern for expensive articles where operating costs and production excess should be as low as possible. The computational complexity that is inherently associated with this problem has motivated yet many researchers to come up with better, more efficient linear programming formulations that produce both optimal and near optimal solutions in an acceptable boundary of time [6,7]. However, when problem dimensions increase, these techniques tend to slow down considerably and take a significant longer amount of time to produce acceptable cutting proposals. As will be demonstrated in the following paragraphs, a genetic algorithm approach yields solutions faster than any LP model yet developed and is able to render excellent proposals almost instantaneously.

The paper is organised as follows. In section 2, a more detailed description of the layout problem is given, together with a condensed outline of the LP model as it will be used in our GA. In section 3, an initial GA is developed, giving special attention to the encoding mechanism applied and the layout of the fitness function. The core features of our GA are primarily developed in light of universal genetic algorithm theory, as can be found in many general GA textbooks [12,5,8]. Due to the poor quality of preliminary results achieved, a threefold genetic enhancement is carried out in section 4 that effectively shrinks the active operating universe of the genetic algorithm. Our initial GA is improved by creating intelligent genetic operators, by executing a level of local optimisation and by implementing a heuristic demand feasibility operator. Section 5 contains then our major research results and compares the performance of the GA with other techniques. The paper is concluded by summarising the most important topics and by giving some ideas for future research.

2. The Layout Problem

2.1. Problem description

For a particular clothing article that is available in various sizes, the layout problem boils down to finding a collection of cutting table layouts that allow to satisfy demand with as little article excess as possible. In essence, setting up a cutting table corresponds to placing a number of stencils on the table where every stencil consists of several layers of fabric. A sequence of stencils is called a pattern and every stencil in a pattern is associated with a particular size. Cutting equipment specifications make the number of stencils per pattern limited and enforce all stencils within a specific pattern to contain an equal number of fabric layers.

The figure below pictures a possible pattern composition, embodying 3 stencils of 2 different sizes, where the number of layers equals 5.

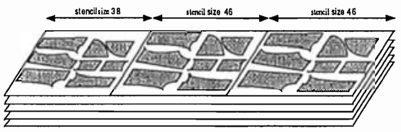

Figure 1: Possible layout of a pattern

Preparing a cutting table is a very time consuming and labour intensive process that makes the number of patterns should be kept at a minimum level. Since production costs are generally much more sensitive to adding additional patterns than to overproduction, the number of patterns is decided upon analytically by setting the number equal to the minimum number of patterns that are needed to fulfil demand.

Table 1 illustrates a typical cutting problem where demand is available for 5 different sizes ($|P|$), a maximum of 3 cutting patterns may be used ($|R|$) that can consist of a sequence of 4 stencils (b), no more than 35 layers of fabric high (H). The number of necessary patterns to produce sufficient articles of each size was set to its lower bound value of 3.

		Sizes				
	Height	38	40	42	44	46
Pattern 1	27	2	2	0	0	0
Pattern 2	31	0	1	2	1	0
Pattern 3	29	0	0	1	1	1
Production		54	85	91	60	29
Demand		54	84	91	60	29
Excess		0	1	0	0	0

Table 1: Cutting proposal, demands and excess production

2.2. LP formulation

As can be read in [7], a straightforward formulation of the layout problem leads to a general integer, *non-linear* programming problem. The authors attempt to render the model solvable as a general integer *linear* program through a set of discretizations and linearizations of the variables used. Also, a network-knapsack approach to the cutting problem is advocated where the number of stencils in a pattern is decided upon by going through the various sizes sequentially. After an extensive derivation, the authors have come up with the

LP formulation below, where the variables x_{ijrs} and z_j constitute the core variables of the problem at hand. In the model, the variable x_{ijrs} will equal 1 if one puts s stencils of size i in pattern j when r stencil places are still free. The variable z_j on the other hand represents the number of clothing layers in pattern j. The authors formulate the objective function of the cutting stock problem such that the total production of articles is minimised while satisfying a collection of demand, network and height constraints. The figure below depicts the final LP model (the variable $v_{ijk} = z_j$ when k stencils of size i are put on the table in pattern j, 0 otherwise):

$$\text{Min} \sum_{i \in P} \sum_{j \in R} \sum_{k \in B} k v_{ijk}$$

subject to:

$$\sum_{j \in R} \sum_{k \in B} k v_{ijk} \geq d_i \quad , \forall i \in P$$

{demand constraints}

$$y_j = \sum_{s \in B} x_{1jbs} \quad , \forall j \in R$$

$$x_{1jb(b-r)} = \sum_{s=0}^{r} x_{2jrs} \quad , \forall j \in R, \forall r \in B$$

$$\sum_{s=0}^{b-r} x_{ij(r+s)s} = \sum_{s=0}^{r} x_{(i+1)jrs} \quad , \forall j \in R, \forall r \in B, \forall i \in P \setminus \{1,|P|\}$$

{network constraints}

$$\sum_{k \in B} v_{ijk} = z_j \quad , \forall i \in P, \forall j \in R, \forall k \in B$$

$$v_{ijk} \leq H \sum_{r=k}^{b} x_{ijrk} \quad , \forall i \in P, \forall j \in R, \forall k \in B$$

$$z_j \leq H y_j \quad , \forall j \in R$$

{height constraints}

$$v_{ijk} \geq 0,\ x_{ijrs} \in \{0,1\},\ y_j \in \{0,1\},\ z_j \in \{0,1,2,3,...\}, \quad \forall j \in R, \forall i \in P, \forall k,r,s \in B, s \leq r$$

Figure 2: Objective function and demand, network and height constraints for the LP model

The demand constraints in the LP model assure that the final cutting proposal will meet the available demand data of the various sizes. The network constraints on the other hand force us to allocate sizes to a particular pattern in a sequential way, taking into account the amount of free space left. The network is processed starting with size 1 and ending with size $|P|$. Finally, height constraints take care of the fact that the number of clothing layers in a pattern is not exceeding its upper limit H.

3. An Initial Genetic Algorithm Approach

3.1. Genetic algorithms

Genetic algorithms were invented to mimic some of the processes observed in natural evolution when solving difficult problems in a wide spectrum of scientific domains. The basic features of this evolutionary process made Holland [12] in the early 70's believe that, when appropriately incorporated in a computer algorithm, they might yield a technique for solving problems in a way that nature has done in the past.

The process of evolution operates on so-called chromosomes or genetic strings in a population. These chromosomes represent solution candidates for a specific problem at hand while their fitness indicates the valuation of their associated solution. During the execution time of a genetic algorithm, a primitive population of chromosomes will evolve by generating offspring and applying simple genetic operators such as crossover and mutation. The basic idea of the algorithm boils down to Darwin's concept of survival of the fittest: elementary DNA structures will start to dominate the population and will in the long run constitute indispensable genetic schemes to survive. Chromosomes lacking these fundamental DNA patterns have a low chance of survival and will eventually be overruled by better fitting competitors. It is exactly the disclosure of those vital DNA densities (called schemata) at which a genetic algorithm is aimed. The more these schemata are spread throughout the population, the more the algorithm will have converged to a set of high quality solutions. It is common to retain the best fitting chromosome at that point in time as a resolution to the problem at hand.

3.2. Encoding of cutting proposals

Careful examination of the LP formulation of the cutting problem reveals that the variables x_{ijrs} and z_j suffice to constitute feasible cutting proposals and to cover the entire solution space. Moreover, the 0/1 nature of the x_{ijrs} variables makes them very suitable to be encoded in a genetic string without further transformation. In order to get a full binary representation of an LP solution, the number of fabric layers for a particular pattern was translated into a binary notation.

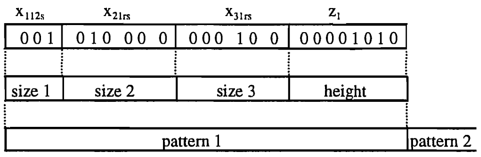

Figure 3: Layout of a genetic string when |P|=3, |R|=2 and b=2

Figure 3 visualises the above encoding strategy for a cutting problem with two patterns, three sizes and a maximum number of stencils on the cutting table of two. Reading the first pattern from the left to the right gives us the following values for the variables x_{ijrs} and z_j:

$x_{1122} = x_{1121} = 0$	$x_{3122} = x_{3121} = x_{3120} = 0$
$x_{1120} = 1$	$x_{3111} = 1$
$x_{2122} = 0$	$x_{3110} = x_{3100} = 0$
$x_{2121} = 1$	
$x_{2120} = 0$	
$x_{2111} = x_{2110} = x_{2100} = 0$	$z_1 = 10$

Table 2: Decoding of genetic material in Figure 3

A potential drawback of the above encoding method lies in the fact the application of genetic operators to a particular chromosome may easily introduce infeasibility or solutions that violate the knapsack constraints. Therefore, during early phases of research, an alternative genetic algorithm was developed that operates on the general *non-linear* integer problem formulation (full integer encoding) avoiding the introduction of a network-knapsack approach. Although this full integer representation severely reduced the magnitude and the complexity of the active genetic universe, feasibility could still not always be guaranteed after applying crossover and mutation operators. Moreover, the heuristic, "ad hoc" peculiarity of the algorithm made it extremely difficult to defend the resulting technique as a genuine, authentic application of genetic algorithm theory.

3.3. Algorithm design

3.3.1. Fitness function

Since a population of cutting proposals may consist of a mixture of feasible and infeasible solutions, careful design of the fitness function is imminent. In the literature, many approaches have yet been proposed to handle constraint breaching solutions in a penalty function [13,17,20,16]. It was believed for our GA to work properly, penalties for violating problem constraints should be assigned in a way that no feasible chromosome can ever be outperformed by an infeasible cutting proposal. In light of this rationale, the following fitness/penalty (Pf) function was developed:

$$Pf = f_n + f_u + f_h + f_o \qquad (1)$$

$$f_n = \sum_{j \in R} \left[\begin{array}{c} \left| \sum_{s \in B} x_{1jbs} - 1 \right| + \sum_{r \in B} \left| x_{1jb(b-r)} - \sum_{s=0}^{r} x_{2jrs} \right| + \\ \sum_{i \in P \setminus \{1,|P|\}} \sum_{r \in B} \left| \sum_{s=0}^{b-r} x_{ij(r+s)s} - \sum_{s=0}^{r} x_{(i+1)jrs} \right| \end{array} \right] * p_n \qquad (2)$$

$$f_u = \sum_{i \in P} \mathrm{Max} \left\{ d_i - \sum_{j \in R} \left(\sum_{r \in B(i)} \sum_{s=0}^{r} s * x_{ijrs} * z_j \right), 0 \right\} * p_u \qquad (3)$$

$$f_h = \sum_{j \in R} \mathrm{Max} \{ z_j - H, 0 \} * p_h \qquad (4)$$

$$f_o = \sum_{i \in P} \mathrm{Max} \left\{ \sum_{j \in R} \left(\sum_{r \in B(i)} \sum_{s=0}^{r} s * x_{ijrs} * z_j \right) - d_i, 0 \right\} * p_o \qquad (5)$$

The fitness function in (1) is composed of four different sub-functions: a penalty function for violating network constraints (f_n), demand constraints (f_u) and height boundaries (f_h) and a penalty function for every article produced in excess of demand (f_o). Feasible solutions will have a single penalty term (f_o), while infeasible solutions will be penalised both for constraint violation and overproduction. In order to get a clear boundary between feasible and infeasible solutions, the penalty factors p_n, p_u and p_h have to be greater than the total maximum demand overshoot penalty (f_o) a feasible solution can ever incur, which is given by $\left(b\,H\,|R| - \sum_i d_i \right)$. Hence, following the so-called minimum-penalty rule [14], penalty factors are established causing minimal gap between feasible and infeasible solutions:

$$p_u, p_h, p_n = \left(b\ H\ |R| - \sum_i d_i \right) p_o \qquad (6)$$

3.3.2. Crossover & mutation operators

In light of classic genetic algorithm theory, we constructed both standard one- and two-point crossover functions as well as a mutation operator. Parents can be selected in our algorithm on a pure random basis or using a roulette wheel strategy (RWS) that takes into account the fitness values of the chromosomes. Given a parental pair of chromosomes, the genetic operators below are executed with a certain probability. By varying this probability level for crossover and mutation separately, the genetic conduct of the algorithm can be customised during execution time.

- *one-point crossover:* new chromosomes in the population are formed by exchanging segments of the parents. After determining a cutting point in a genetic string, segments beyond this point are swapped between parents to generate offspring.
- *two-point crossover*: this type of crossover generates new chromosomes by interchanging pieces of parental chromosomes between two randomly chosen cut-off points.
- *mutation*: the mutation operator scans every bit of a chromosome and inverts it with a certain probability.

Notice that the above crossover and mutation operators were actually carried out in a double stage procedure. As a matter of fact, chromosomes were split up into a part containing only network related variables and another part containing binary encoded height variables. Genetic operators were then carried out separately on both parts using individual crossover and mutation rates. Although this approach is essentially non-universal in nature, it was necessary to ensure the possibility of swapping cross pattern x_{ijrs}-variables without exchanging the accompanying z_j's. Direct application of classic genetic operators on the initial encoding sequence in Figure 3 might in that way have caused some feasible solutions to be highly unreachable.

3.3.3. Algorithm backbone

Given the above design of the fitness function and the working method of genetic operators, the major backbone of our algorithm can be summarised as follows (depending on the offspring strategy applied):

```
set up an initial population of feasible chromosomes
evaluate the population of chromosomes
repeat
        select two chromosomes which function as parents
        apply genetic operators (crossover & mutation) to
               generate offspring
        replace the worst chromosomes in the population by
               the new children
        evaluate the population of chromosomes
until a stopping condition is met
```

```
set up an initial population of feasible chromosomes
evaluate the population of chromosomes
repeat
        for all the chromosome pairs in the population do
        begin
               select two chromosomes which function as
                      parents
               apply genetic operators (crossover & mutation)
                      to generate offspring
               add the offspring to a new population
        end
        replace the old population by the offspring
        evaluate the new population of chromosomes
until a stopping condition is met
```

Table 3: Genetic algorithm layout for steady state (top) and generation strategy (bottom)

Adopting a steady genetic evolution strategy (top) means going through a number of iterations during which pairs of generated children replace the worst solutions in the population. Following a generation strategy (bottom) on the other hand implies creating an entire new population of solutions after which the original population is completely replaced by generated offspring. In order to maintain the best solutions in every population, an elitist strategy is applied in case of the generation approach.

3.4. Preliminary results

We applied the above genetic algorithm to an extensive number of problem cases, including the example in Table 1. The major parameters we examined in our algorithm were the penalty factors in the fitness function, the population size, the generation strategy, the parent selection method, the type of crossover applied and the crossover & mutation rates.

Although it was mentioned yet feasibility of chromosomes after executing genetic operators is not guaranteed, it was hoped that an accurate arrangement of penalty factors, together with a well-thought design of other parameters would somehow suppress constraint-breaching solutions and enforce feasibility in the long run. However, after a profound period of intensive testing, it was concluded that no suitable parameter set up existed to make the algorithm work. We varied also a few other minor ad hoc parameters without any significant success. In most cases, intermediate populations consisted largely of adamantine infeasible solutions and only a few valid proposals far from the optimal cutting pattern.

In light of these preliminary results, it was believed that letting the algorithm drive itself through the space of feasible/infeasible solutions soon renders it completely adrift and meant putting too much of a burden on the penalty factor/parameter set up. In the next paragraph, the algorithm is therefore submit to a profound enhancement phase to improve the algorithm's performance and to reduce the probability of generating infeasible solutions.

4. Genetic Enhancement

4.1. Enforcing feasibility continuance in the network constraints

4.1.1. Crossover design

In the appendix, we derive probability expressions for maintaining a feasible network-knapsack flow after applying either one- or two-point crossover. The analysis is based on a non-converged, random population of feasible knapsack solutions (genetic heterogeneity) that are submitted to either a one- or a two-point crossover operator with equal probability. For a particular problem dimension ($|R|$, $|P|$ and b), it can be proven the likelihood of preserving feasibility in the network constraints can be written as in the equation below (the reader is referred to the appendix for an explanation of the variables used).

$$\ell_{b,|R|,|P|} = \frac{1}{2}\left[\frac{|R|}{\lambda}+\frac{(|P|-1)|R|}{\lambda}\rho+\sum_{i=1}^{|P|}\alpha_i\left(\pi_i\delta_i+\tilde{\pi}_i\tilde{\delta}_i\right)\right]+ \frac{1}{2}\left[\frac{\binom{|R|}{2}\left(\frac{\lambda}{|R|}\right)^2}{\binom{\lambda}{2}}\left[\frac{|R|}{\lambda}+\frac{(|P|-1)|R|}{\lambda}\rho+\sum_{i=1}^{|P|}\alpha_i\left(\pi_i\delta_i+\tilde{\pi}_i\tilde{\delta}_i\right)\right]^2+\frac{|R|\binom{\lambda/|R|}{2}}{\binom{\lambda}{2}}\frac{\sum_{\psi=1}^{\lambda/|R|-1}\sum_{\beta(\psi)}\left(\Omega_1'\Phi_1+\Omega_2'\Phi_2\right)}{\sum_{\psi}|\beta(\psi)|}\right] \tag{7}$$

For the problem dimensions in Table 1 ($|R|=3$, $|P|=5$ and b=4,), direct evaluation of (7) reveals the likelihood $\ell_{b,|R|,|P|}$ of preserving feasibility in the network constraints turns out to be no more than 20%. Figure 4 illustrates this feasibility maintenance rate for a set of varying problem dimensions, fixing the number of patterns $|R|$ to a value of 3 and letting b and $|P|$ vary on the interval [1,10].

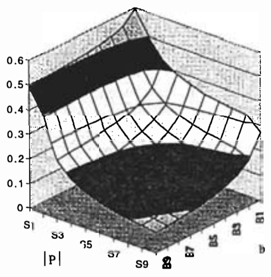

Figure 4: Genetic feasibility continuance plane after applying crossover for varying problem dimensions

Although it can be argued that infeasible solutions may return to the domain of feasible proposals after applying crossover or mutation, empirical results show this event to be extremely unlikely. An initial population of feasible solutions will hence be shrunk by an approximate factor of $\left(\ell_{b,|R|,|P|}\right)^{\iota}$ after going through ι crossover iterations.

In order to circumvent this infeasibility tendency, many authors in the literature apply some sort of repair strategy that fixes errors in a chromosome [1,9,2,18]. Although preliminary experiments with this repair strategy showed promising results, we felt that little authentic genetic behaviour was left. Indeed, for the problem dimensions in Table 1, a repair was needed in 80% of the cases and was basically carried out by scanning a chromosome for errors and fixing either the preceding or the following part in a complete random fashion. We therefore designed an intelligent one- and two-point crossover operator (χ_{n_1} & χ_{n_2}) that does guarantee feasible offspring and makes any repair action superfluous. Careful analysis of the genetic material reveals that the class of feasible knapsack solutions is completely covered when crossover points are positioned only after a collection of x_{ijrs}-variables for a particular size. Moreover, when a crossover point (cp) is chosen among a set of points at which the network state (ie. the total flow in cp) is identical across both parents, an intelligent crossover operator comes out that takes into account the idiosyncrasy of the problem at issue. The figure below illustrates the working method of this crossover for the two-point crossover scenario, taking into account that the network flow at both crossover points must be equal across both parents.

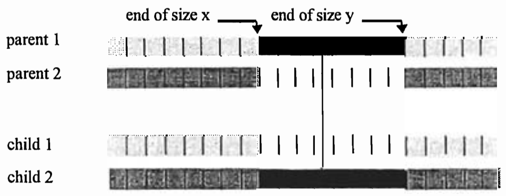

Figure 5: Illustration of the intelligent two-point crossover operator

Although the above crossover operator is basically non-universal in nature, we feel that it is much more suitable from a schemata point of view and that it is capable of exchanging fundamental genetic structures without loosing essential genetic granularity power. As a matter of fact, elementary genetic schemata for the problem at hand are strings of size related bits that represent a number of stencils laid for a particular amount of free space left on the cutting table. Genetic evolution comes in this case then down to the swapping of size related genetic material across chromosomes instead of incoherent individual bits.

4.1.2. Mutation design

While the above crossover design warrants feasibility maintenance, the application of mutation may still rupture the network flow. In the appendix, the class of mutation scenarios that preserve feasibility with respect to the network constraints is defined. The analysis comes down to the disclosure of the fact that mutations on feasible chromosomes have to take place in pairs within a particular size. Moreover, since mutating bits in a particular size boils down to altering the number of stencils, mutations should always proceed in strings (**σ**) of length ι ($\iota >= 2$) of consecutive sizes in order to preserve the total flow through **σ**. Taking into account feasible mutation threads can be formed by combining several **σ**'s, the total number of allowable mutation scenarios can be written as in the equation below (the reader is again referred to the appendix for an explanation of the variables).

$$|M| = \left[\sum_{i=2,\text{even}}^{2|P|} \sum_{j=1}^{\lceil i/4 \rceil} \sum_{\sum_{k=1}^{j} \iota_k} \left| \bigtimes_{k=1}^{j} \Sigma_{\iota_k} \right| \prod_{k=1}^{j} \left(\theta_{\iota_k}(\gamma_k) \right) \right]^{|R|} \quad \textbf{(8)}$$

Consequently, we designed an intelligent mutation operator (μ_n) that randomly carries out a mutation scenario of **M**. The figure below illustrates the application of this intelligent mutation operator when mutations take place in strings σ_1 and σ_2 within pattern j.

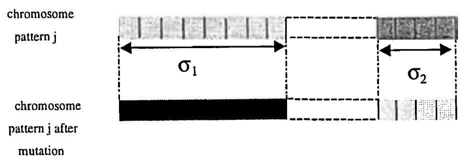

Figure 6: Illustration of the intelligent mutation operator

4.2. Local stack altitude optimisation

Although a similar strategy as above could now be advocated for managing feasibility in the height constraints, we felt that the translation of integer height variables into a binary representation, followed by the application of a parallel genetic algorithm with intelligent crossover or mutation operators would be an inefficient way of determining height variables for a particular chromosome. Moreover, careful examination of the problem at hand reveals that the algorithm's search space is primarily set up by the network related x_{ijrs}-variables. As a matter of fact, it is straightforward to associate optimal or near optimal numbers of stack layers (z_j-variables) with a particular network flow by setting all z_j's to their maximum value and decrementing them step by step until one or more demand constraints become violated. Hence, instead of developing a parallel genetic algorithm to work on the height variables, a local height optimisation strategy (η) was chosen that assigns stack variables z_j to every chromosome in the population.

4.3. Heuristic demand feasibility operator

The genetic algorithm so far developed operates within the domain of feasible solutions with respect to both network and height constraints. Although intelligent crossover and mutation operators could have been designed coercing the algorithm to satisfy demand constraints as well, it was believed this would severely have lessened the number of feasible genetic progress pathways and have endangered the algorithm to strand early in local minima.

Extensive initial experiments revealed the genetic algorithm gives favourable results, although it was felt both the execution time and a regular reach of the optimal solution still needed improvement. The table below depicts typical results that were obtained by applying our GA to the problem case in Table 1. In some runs, it appears that after many iterations, the algorithm was still navigating through solutions that violated the demand constraints. Also, only once the optimal cutting proposal to the problem at issue was found.

Best Solution	5	1	3	3	3	2	3
Avg. Penalty	35.1	2568	26.3	89	33.2	32.1	2660

Table 4: Best solution and average penalty for seven runs on Table 1

To enforce the algorithm to produce only acceptable cutting stock proposals during the entire genetic evolution path, a local heuristic network optimisation (ν) was carried out that rearranges the flow through the network until a solution is achieved that satisfies the demand constraints as well. Chromosomes are altered through a cascade of flow redirections using the following heuristic procedural network modification:

```
try to satisfy demand by increasing the number of layers
while demand is not satisfied do
begin
        either try to satisfy demand by filling empty places on the
                cutting table
        either switch stencils across sizes within a pattern
        as follows
                select randomly a demand breaching size s
                select randomly a demand overshooting size s'
                decrement the number of stencils for s' in a random
                        pattern p
                increment the number of stencils for s in p
        end 'as follows'
        adjust the sets of demand breaching and demand
                overshooting sizes
end
delete any superfluous stencil within a particular pattern
```

Table 5: Heuristic network flow redirection

Every chromosome that goes through the above heuristic feasibility operator is guaranteed to have a network flow for which an accompanying set of feasible fabric layers exists (z_j-variables) that renders the entire chromosome feasible with respect to all the problem constraints.

5. Results

5.1. Parameter setup

5.1.1. Population size, replacement rate & number of generations

A well-thought population size (κ) is imminent since a population too small may leave vast areas of the solution space uncovered, while a population too large soon demands a tremendous amount of processor time. Also, the algorithm should run for an adequate number of generations (ι) to allow for a satisfactory level of convergence. Concerning the percentage of chromosomes to be generated every iteration (r), a combination of a steady state generation approach and a delete all strategy with elitism is advocated. In fact, initial experiments revealed it is desirable to replace only a few chromosomes by offspring to allow smooth and regular convergence while more replacements should take place to increase the algorithm's performance. Anyway, bad specification of κ, ι and r may jeopardise both the coverage of the solution space and the algorithm's overall speed of convergence.

It is therefore believed that a useful lower bound relationship between κ, ι and r can be defined by ensuring that the number of feasible network flows for a particular pattern is covered by generated offspring. Straightforward calculation shows the total number of feasible network flows to be:

$$n_{fs} = \binom{b+|P|}{b} \qquad \textbf{(9)}$$

Hence, following the above rationale, any parameter setup of κ, ι and r should satisfy:

$$n_{fs} \leq \iota\, r\, \kappa \qquad \textbf{(10)}$$

5.2. Crossover and mutation strategy

Extensive preliminary testing indicated crossover is mainly responsible for overall population convergence during early phases of evolution. It is therefore preferable to hold mutation as low as possible not to disturb this crossover driven genetic population improvement. However, after some point in time, crossover seems to have shaped fundamental schemata that appear in many solutions, as indicated by the figure below:

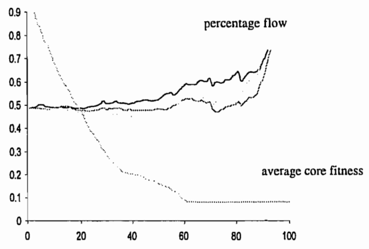

Figure 7: Crossover driven schemata density evolution and average core fitness

Figure 7 depicts a typical percentage flow of chromosome pairs in the steady core that have an equal number of free stencil positions at particular stages in the knapsack network over several generations. The figure is based on the problem dimensions of Table 1; values are shown for all patterns and are averaged across sizes within a pattern.. The average core fitness values were calculated using the following scaling function (κ represents the core size): $\frac{1}{\kappa}\sum_{\bar{\kappa}} Pf_t(\bar{\kappa}) \Big/ \frac{1}{\kappa}\sum_{\bar{\kappa}} Pf_1(\bar{\kappa})$.

Although at some points in time a minor level of diversity is reappearing, it is clear that fundamental network states (schemata) are distributed across the entire population and that crossover in the long run will merely swap these schemata across parents without little additional convergence effect. At that moment, it is highly preferable to reintroduce variety in the entire population by mutation. Hence, in order for mutation and crossover to be effective, it was concluded that:

- the crossover rate should be as high as possible when no major convergence has occurred yet
- the mutation rate should depend on a measure of convergence
- mutation should proceed through the entire population
- mutations should take place within those sizes that contribute most to the penalty function

Let now $s_d^j(t)$ stand for the schemata density curve as pictured in Figure 7 for pattern j at generation t, then a measure of population convergence within pattern j is given by estimating $\left|{\partial s_d^j(t)}/{\partial t}\right|$. Modelling the fact mutation should be more active in penalty contributing sizes can be done by defining a mutation rate per size and letting the rate vary with the size's overall core total overproduction share, which is given by:

$$a = \left. \sum_{\tilde{\kappa}=1}^{\kappa}\left(\left(\sum_{\bar{j}=1}^{|R|}\sum_{k=1}^{b} kv_{i\bar{j}k}(\tilde{\kappa})\right)-d_i\right) \middle/ \sum_{\bar{i}=1}^{|P|}\sum_{\tilde{\kappa}=1}^{\kappa}\left(\left(\sum_{\bar{j}=1}^{|R|}\sum_{k=1}^{b} kv_{\bar{i}\bar{j}k}(\tilde{\kappa})\right)-d_{\bar{i}}\right) \right. \qquad (11)$$

Extensive preliminary experiments revealed a scaling factor was needed in combination with the above expression to define an efficient mutation and crossover rate as follows:

$$m(i,j) = \left(e^{-1.10^4 \frac{\left|{\partial s_d^j(t)}/{\partial t}\right|}{a}} \right) \qquad (12)$$

$$c_r = 1 - \frac{1}{|P||R|}\sum_{i=1}^{|P|}\sum_{j=1}^{|R|} m(i,j) \qquad (13)$$

Chromosomes in the entire population will thus be submitted to a mutation process that scans the genetic material across sizes (i) and patterns (j) and initiates a particular mutation scenario involving size i within pattern j with a probability equal to m(i,j). Also, offspring is generated by applying a crossover rate c_r that is inversely proportional to the average m(i,j).

5.3. Major results and comparison to other techniques

We compared the performance of our GA to both the LP formulation of the problem as discussed in paragraph 2.2 (LP1) and two alternative formulations that were recently proposed by [6] (LP2 & LP3). The table below gives an overview of all the test cases used with their optimal solution (minimum excess).

Case Nr.	Parameters b	$\lvert R \rvert$	$\lvert P \rvert$	H	Available Demand Data	Min. Excess
1	4	3	5	35	54 84 91 60 29	1
2	4	3	5	35	25 70 63 54 39	1
3	4	3	5	35	33 82 77 62 34	1
4	4	3	5	35	74 64 28 34 59	1
5	4	3	5	35	21 54 61 5 41	1
6	4	3	5	35	21 54 61 15 41	2
7	5	3	6	10	8 12 14 23 16 6	0
8	5	3	4	50	78 133 176 96	0
9	5	3	6	50	44 58 68 119 78 34	1
10	3	4	6	50	37 48 58 63 44 29	0
11	5	4	6	10	19 25 40 43 31 17	5
12	5	4	6	50	98 145 180 207 167 83	0
13	6	5	6	50	115 152 44 284 196 135	0
14	6	5	7	50	120 70 130 170 208 50 100	0
15	6	6	7	35	59 100 103 73 121 52 35	0
16	6	6	8	35	112 142 127 72 71 56 102 51	0
17	8	6	8	35	58 71 106 311 208 101 161 70	0

Table 6: Characteristics of test cases used

The genetic algorithm applied on a population of chromosomes (c) and a steady core (κ) at time t with preservence of the best solution (c_t^b) can now be jotted down as:

$$\begin{gathered}
\forall c \notin \kappa : c_{t+1} = \eta \circ \nu \circ \left\{ \sum_{i,j} m(i,j)\, \mu_n \circ c_r \left\{ \chi_{n_1} \wedge \chi_{n_2} \right\} \left(\tilde{c}_{t,} \tilde{\tilde{c}}_t \right) \right\} \\
\forall c \in \kappa \setminus \left\{ c_t^b \right\} : c_{t+1} = \eta \circ \nu \circ \left\{ \sum_{i,j} m(i,j)\, \mu_n \left(c_t \right) \right\} \qquad \textbf{(14)} \\
c_{t+1}^b = c_t^b
\end{gathered}$$

We defined the performance of the above GA by executing 10 independent runs for each case and by calculating the average time the algorithm took to reach the optimal solution. If the optimal cutting layout was not found, the time it took the algorithm to reach the best cutting pattern in the population was taken into consideration. The table below gives an overview of the initial problem cases tested together with their optimal solution (minimum excess) and the execution times for LP1-3 and the genetic algorithm. Also, the table depicts the best solution the GA found in 10 runs and an accuracy level as the percentage of runs that reached the optimal cutting pattern. Values between parentheses indicate best solutions if the optimal solution wasn't found, eg. in case 4, 2 runs produced a sub-optimal solution with an excess of 2 articles.

Case	Min. Exc.	Execution Times				GA best	Accuracy
		LP1	LP2	LP3	GA		
1	1	18:32	2:19	0:56	< 1s	1	1
2	1	38:27	6:16	3:51	0:05	1	1
3	1	20:04	4:00	2:01	0:05	1	1
4	1	35:29	4:39	4:04	0:09	1	0.8 (2x2)
5	1	41:12	5:56	2:08	0:08	1	1
6	2	7:41	4:35	5:33	0:03	2	1
7	0	> 10h	31:53	> 1h	0:03	0	1
8	0	3:40	0:35	5:04	0:08	0	1
9	1	> 40h	≅ 2h	> 15h	0:11	1	0.8 (2x2)
10	0	56:41	13:44	5:31	0:13	0	0.4 (6x1)
11	5	> 40h	26:00	> $1^{1/2}$h	0:00	5	1
12	0	> 40h	29:12	36:43	0:16	0	0.7 (3x5)

Table 7: GA versus LP1-3 performance (minutes:seconds)

The table above was constructed by generating a maximum of 5000 new chromosomes, using the parameters r=25%, κ=100, ι=200, well satisfying the lower bound relationship (10). For the LP models, a Pentium II 233Mhz was available, while the genetic algorithm ran on a Pentium II 400Mhz station. For comparative reasons, one can multiply the execution times of the GA by a factor of approximately 3/2 to obtain an estimate of results that would have been achieved on a 233Mhz station.

The conclusions that can be drawn from Table 7 are twofold. First, the performance of the GA is severely better than any of the LP formulations presented. The algorithm finds optimal or near optimal solutions in a time span far narrower than any LP model. It should be mentioned however that the execution times for LP formulations involve a large amount of time to prove the superiority of a solution that was found much earlier. On the other hand, direct comparison of execution times between an LP model and the GA can be done for cases that have an optimal solution with no overproduction (zero excess). Hence, for cases 7,8,10 & 12, it still took the LP models a significant larger amount of time to actually reach the optimal solution of zero excess. In order to circumvent the possible blurring effect of comparing results for cases with a non-zero overproduction an additional set of test cases was designed. The table below contains the execution times of 5 new test cases for the second LP model (LP2 performs best for complex cases) and the genetic algorithm. The results of Table 8 were achieved using customised values of κ, ι and r (all execution times are 400Mhz results). Combining the results of Table 7 and Table 8 for cases with zero excess, it is apparent the GA needs a significant lower amount of time to reach the optimal cutting pattern compared to any of the LP models. Also, for the most complex cases (16 and 17), it was found it took LP2 more than a full day of processing to come up with a feasible solution.

Case	Min. Exc.	Execution Times			GA best	Accuracy
		κ ι r	LP2	GA		
13	0	100 200 25%	≅ 2h	0:50	0	0.6 (4x1)
14	0	100 200 25%	> 24h*	0:32	0	1
15	0	100 200 25%	> 24h*	0:50	0	1
16	0	200 200 25%	> 24h*	2:48	0	0.6 (4x1)
17	0	200 400 25%	> 24h*	7:16	0	1

Table 8: Results on additional zero-excess cases (minutes:seconds, * = execution was halted after indicated time)

A second remark that has to be made about the results in Table 7 concerns the fact for some cases the optimal cutting pattern is not always reached by the GA. Indeed, for cases 4, 9, 10 and 12, the accuracy rating drops to a somewhat mediocre level of about 60%. It is believed however the accuracy is heavily dependent on the parameter setup of κ, ι and r. Therefore, we submitted the former cases again to our GA removing the upper limit on ι, setting r to 25% and κ to 100. The results in the table below indicate the average execution times (10 runs per case) are still lower compared to any of the LP models. In most runs, the optimal solution was found in less than 2 minutes. However, for case 10, it took the GA in a particular run more than 7 minutes to find the optimal solution. Leaving the removal of the upper limit, we therefore also increased both the population size and the replacement rate. As can be seen in Table 9, GA average execution times are improved for all cases while maximum times are severely reduced. Based on the results in Table 9 it is conceived careful arrangement of κ, ι and r is imminent since these parameters play an important role in the overall performance of the genetic algorithm.

Case	Rep. Rate (r)	Population Size (κ)	Average Time	Max Time
4	25%	100	0:22	1:07
9	25%	100	0:36	1:42
10	25%	100	2:08	7:06
12	25%	100	1:21	3:10
4	50%	200	0:08	0:13
9	50%	200	0:17	0:41
10	50%	300	1:07	5:50
12	50%	300	0:25	1:15

Table 9: Execution times removing upper limit on ι for varying κ and r

6. Conclusions and Future Research

In this paper, we constructed a powerful hybrid genetic algorithm for solving a layout problem in the Belgian fashion industry. Through a set of initial experiments, it was found a universal GA approach yields less than satisfactory results. Further analysis indicated an effective constraint handling technique was indispensable to improve the algorithm's performance. In light of these findings, intelligent schemata based genetic operators were constructed and implemented together with a local optimisation strategy and a heuristic feasibility operator. These enhancements severely reduced the active genetic universe and were able to confine the algorithm to the space of feasible solutions. Moreover, an intelligent variable mutation and crossover rate scheme was used to allow for rapid convergence during early phases of evolution without loss of essential genetic diversity. Finally, computational results indicated the GA was considerably faster than any LP model yet developed.

As a result of this study, many future research activities can be carried out on the problem at hand. It might be interesting to investigate how the GA can effectively be used as an initialisation procedure for any of the LP models. On the other hand, research can be carried out on how the algorithm can decide itself about good (feasible) crossover points by inductive learning [21] as an alternative to the peculiar design strategy applied in this article. Also, in light of recent activities [11], the fruitfulness of applying co-evolutionary genetic algorithm techniques for handling problem constraints can be analysed.

Acknowledgement

We would like to thank Prof. Dr. F. Put for making this research possible. Also we would like to acknowledge Prof. Dr. Ir. Z. Degraeve for providing us with initial research material and R. Jans for additional comments on the LP model results. Finally, a word of gratitude is owed to P. Claes for research assistance.

7. References

[1] Beasley J.E. & Chu P.C., *A Genetic Algorithm for the Set Covering Problem,* European Journal of Operations Research, 94, 1996, pp. 392-404.

[2] Beasley J.E. & Chu P.C., *A Genetic Algorithm for the Multidimensional Knapsack Problem,* Journal of Heuristics, 4, 1998, pp. 63-96.

[3] Beasley J.E. & Chu P.C., *Constraint Handling in Genetic Algorithms, The Set Partitioning Problem,* Journal of Heuristics, 11, 1998, pp. 323-357.

[4] Chen C.L., Vempati V.S. & Aljaber N., *An Application of Genetic Algorithms for Flow Shop Problems,* European Journal of Operations Research, 80, 1995, pp. 389-396.

[5] Davis L., *Handbook of Genetic Algorithms,* Van Nostrand Reinhold, NY, 1991.

[6] Degraeve Z., Gochet W. & Jans R., *Alternative Formulations for a Layout Problem in the Fashion Industry,* Research Report, KUL, 1998.

[7] Degraeve Z. & Vandebroek M., *A Mixed Integer Programming Model for Solving a Layout Problem in the Fashion Industry,* Management Science, 44, 1998, pp. 301-310.

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

[9] Gottlieb J. & Paulmann L., *Genetic Algorithms for the Fixed Charge Transportation Problem,* Proceedings of the 1998 World Congress on Computational Intelligence, FUZZ-IEEE & ICEC, 1998, pp. 330-335.

[10] Grefenstette J.J., *Incorporating Problem Specific Knowledge into Genetic Algorithms,* in: Davis L., *Genetic Algorithms and Simulated Annealing,* Los Altos, CA, 1987.

[11] Handa H., Katai O., Baba N. & Sawaragi T., *Solving Constraint Satisfaction Problems by Using Coevolutionary Genetic Algorithms,* Proceedings of the 1998 World Congress on Computational Intelligence, FUZZ-IEEE & ICEC, 1998, pp. 21-26.

[12] Holland J.H., *Adaptation in Natural and Artificial Systems,* Ann Arbor, The University of Michigan Press, 1975.

[13] Homaifar A., Lai S.H.-Y & Qi. X., *Constraint Optimisation via Genetic Algorithms,* Simulation, 62, 1994, pp. 242-254.

[14] Le Riche R., Vayssade C. & Haftka R.T., *A Segragated Genetic Algorithm for Constrained Optimisation in Structural Mechanics,* Technical Report, Université de Technologie de Compiegne, France, 1995.

[15] Michalewicz Z. & Attia N., *Evolutionary Optimisation of Constrained Problems,* Proceedings of the 3rd Annual Conference on Evolutionary Programming, 1994, pp. 98-108.

[16] Michalewicz Z., *The Significance of the Evaluation Function in Evolutionary Algorithms,* Proceedings of the Workshop on Evolutionary Algorithms, University of Minnesota, 1996.

[17] Powell D. & Skolnick M.M., *Using Genetic Algorithms in Engineering Design Optimisation with Non-Linear Constraints,* Proceedings of the 5th International Conference on Genetic Algorithms, 1993, pp. 424-430.

[18] Raidl G.R., *An Improved Genetic Algorithm for the Multiconstrained 0-1 Knapsack Problem,* Proceedings of the 1998 World Congress on Computational Intelligence, FUZZ-IEEE & ICEC, 1998, pp. 207-211.

[19] Rojas R., *Neural Networks: A Systematic Introduction,* Springer-Verlag, Berlin, 1996.

[20] Schoenauer M. & Xanthakis S., *Constrained GA Optimisation,* Proceedings of the 5th International Conference on Genetic Algorithms, 1993, pp. 573-580.

[21] Sebag M. & Schoenauer M., *Controlling Crossover through Inductive Learning,* Proceedings of the 3rd Parallel Problem Solving from Nature Conference, Jerusalem, Israel, 1994.

[22] Whitley D., Starkweather T. & Shaner D., *The Travelling Salesman and Sequence Scheduling: Quality Solutions using Genetic Edge Recombination,* in: Davis L., *Handbook of Genetic Algorithms,* Van Nostrand Reinhold, New York, 1991.

8. Appendix

8.1. Crossover analysis

8.1.1. One-point crossover

Let E_1 and E_2 denote the events of maintaining solutions that satisfy the flow constraints after applying respectively one- and two-point crossover. Careful analysis of a genetic structure reveals that feasibility continuance depends heavily on the position of the crossover point(s) (cp(s)) within the network related variables x_{ijrs}. For one-point crossover, the following cases can be identified:

1. *cp after pattern j:* feasibility is always guaranteed when crossover takes place right after a pattern since network variables are independent across patterns.
2. *cp after size i within pattern j, $i \neq |P|$:* feasibility is only maintained if the number of free stencil positions at the crossover point is identical in both parental chromosomes. Assuming a heterogeneous, non-converged genetic population, this number will be uniform on [0,b] and hence will be equal across both parents with probability $\rho = \frac{1}{(b+1)}$.
3. *cp within size i within pattern j:* feasibility can only be preserved if the crossover point is positioned ahead of the first or behind the last non-zero x_{ijrs} variable within size i across both parents. Assuming genetic heterogeneity, the probability the crossover point falls ahead of the first non-zero x_{ijrs} within size i can be written as:

$$\frac{E\left[\#\text{ of cp's before the first non}-\text{zero } x_{ijrs} \text{ within size i}\right]}{\text{number of cp's within size i}}$$

After some calculation, the probability of performing a crossover preceding the first significant x_{ijrs} within size i equals:

$$\pi_i(\zeta_i) = \frac{\sum_{k=1}^{\zeta_i+1} (k-1)\left(2\left((\zeta_i+1)-k\right)+1\right)/(\zeta_i+1)^2}{\zeta_i}$$

with $\zeta_i = \left(\sum_{k=1+b\delta_{1i}}^{b+1} k\right) - 1,\ i \in \{1,\ldots,|P|\}.$

The probability then the crossover point falls after the last non-zero x_{ijrs} within size i reads after some calculation:

$$\tilde{\pi}_i(\zeta_i) = \frac{\sum_{k=1}^{\zeta_i+1} (\zeta_i+1-k)(2(k-1)+1)/(\zeta_i+1)^2}{\zeta_i}$$

In both foregoing cases, the application of the crossover operator will only hold feasibility if the number of free stencil positions is identical at the crossover point across both parents.

Due to the network-knapsack formulation of the problem, the probability of having an equal number of remaining stencil positions at the crossover point, is size dependent. Following this rationale, $P(E_1)$ can then be written as an aggregate of three terms corresponding to the above scenarios, where the following matrix notation was used (λ represents the total number of network related bits in a chromosome):

$$\underset{|P|\times 1}{\Delta} = \begin{bmatrix} \delta_1 \\ \vdots \\ \delta_{|P|} \end{bmatrix} = \begin{bmatrix} 1 \\ \rho \\ \vdots \end{bmatrix} \qquad \underset{|P|\times 1}{\tilde{\Delta}} = \begin{bmatrix} \tilde{\delta}_1 \\ \vdots \\ \tilde{\delta}_{|P|} \end{bmatrix} = \begin{bmatrix} \rho \\ \vdots \\ 1 \end{bmatrix}$$

$$\underset{|P|\times 1}{\Pi} = \begin{bmatrix} \pi_1 \\ \vdots \\ \pi_{|P|} \end{bmatrix} \qquad \underset{|P|\times 1}{\tilde{\Pi}} = \begin{bmatrix} \tilde{\pi}_1 \\ \vdots \\ \tilde{\pi}_{|P|} \end{bmatrix}$$

$$\underset{|P|\times 1}{A} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_{|P|} \end{bmatrix} = \begin{bmatrix} b\big/(\lambda/|R|) \\ \left(\sum_{k=1}^{b+1} k-1\right)\Big/(\lambda/|R|) \\ \vdots \\ \end{bmatrix}$$

$$P(E_1) = \frac{|R|}{\lambda} + \frac{(|P|-1)|R|}{\lambda}\rho + \sum_{i=1}^{|P|} \alpha_i \left(\pi_i \delta_i + \tilde{\pi}_i \tilde{\delta}_i\right) \qquad (15)$$

8.1.2. Two-point crossover

A two-point crossover operates by randomly selecting two crossover points (cp_1 & cp_2) and by swapping the interlaying parts of the parental chromosomes. Hence, a two-point crossover can be regarded as a "virtual sequence" of two one-point crossovers. Depending on cp_1 & cp_2, we have:

1. *cp_1 & cp_2 within pattern j & j', j≠j':* feasibility is maintained if both imaginary one-point crossovers preserve feasibility.
2. *cp_1 & cp_2 within pattern j:* a distinction must be made between a feasible and an infeasible result after applying the first imaginary one-point crossover. In case the intermediary result is feasible, feasibility will only be further maintained if the number of remaining stencil positions at cp_2 is equal, which occurs with probability $\tilde{\rho} = \frac{1}{(\beta+1)}$ where β stands for the number of free positions at cp_1. In case the intermediary result is infeasible however, "undoing" the action of the first crossover can only restore feasibility.

Let ψ stand for the relative position of cp_1 in pattern j and i for the size in which cp_1 is located (i is completely determined by ψ). Let ψ and β be independent for i > 1 (genetic heterogeneity) and let ω be the number of remaining bits within size i, $\omega = \left(\sum_{i=1}^{i} \sum_{k=1+b\delta_{\tilde{i}1}}^{b+1} k\right) - \psi - 1$. If one defines the set of allowable β's for a particular ψ as $\beta(\psi)$ then $P(E_2)$ can be written as in (16), using the matrix notation below:

$$\underset{4\times 1}{\Omega_1} = P(E_1)\begin{bmatrix} \dfrac{1}{\lambda/|R|-\psi} \\ \dfrac{|P|-i}{\lambda/|R|-\psi} \\ \dfrac{\omega}{\lambda/|R|-\psi} \\ \dfrac{(|P|-i)\left(\sum_{k=1}^{b+1} k-1\right)}{\lambda/|R|-\psi} \end{bmatrix}$$

$$\underset{4\times 1}{\Phi_1} = \begin{bmatrix} 1 \\ \tilde{\rho} \\ \dfrac{\pi_i(\zeta_i)}{\pi_i(\zeta_i)+\tilde{\pi}_i(\zeta_i)}\left(\pi_i(\omega)+\tilde{\pi}_i(\omega)\tilde{\varepsilon}_i\right)+\dfrac{\tilde{\pi}_i(\zeta_i)}{\pi_i(\zeta_i)+\tilde{\pi}_i(\zeta_i)} \\ \pi_{i+1}(\zeta_{i+1})\varepsilon_{i+1}+\tilde{\pi}_{i+1}(\zeta_{i+1})\tilde{\varepsilon}_{i+1}\big| i \neq |P| \end{bmatrix}$$

$$\underset{3\times 1}{\Omega_2} = \begin{bmatrix} \pi_i(\zeta_i)(1-\delta_i)\left(\dfrac{\omega}{\lambda/|R|-\psi}\right) \\ \left(1-\pi_i(\zeta_i)-\tilde{\pi}_i(\zeta_i)\right)\left(\dfrac{\omega}{\lambda/|R|-\psi}\right) \\ \tilde{\pi}_i(\zeta_i)\left(1-\tilde{\delta}_i\right) \end{bmatrix}$$

$$\underset{3\times 1}{\Phi_2} = \begin{bmatrix} \pi_i(\omega) \\ 1/2 \\ \dfrac{\omega}{\lambda/|R|-\psi}+\left[\left(\zeta_{i+1}/(\lambda/|R|-\psi)\right)\pi_{i+1}(\zeta_{i+1})\big| i \neq |P|\right] \end{bmatrix}$$

$$\underset{|P|\times 1}{E} = \begin{bmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_{|P|} \end{bmatrix} = \begin{bmatrix} 1 \\ \tilde{\rho} \\ \vdots \end{bmatrix} \qquad \underset{|P|\times 1}{E} = \begin{bmatrix} \tilde{\varepsilon}_1 \\ \vdots \\ \tilde{\varepsilon}_{|P|} \end{bmatrix} = \begin{bmatrix} \tilde{\rho} \\ \vdots \\ 1 \end{bmatrix}$$

$$P(E_2) = \frac{\binom{|R|}{2}\left(\frac{\lambda}{|R|}\right)^2}{\binom{\lambda}{2}} P(E_1)^2 + \frac{|R|\binom{\lambda/|R|}{2}}{\binom{\lambda}{2}} \frac{\sum\limits_{\psi=1}^{\lambda/|R|-1} \sum\limits_{\beta(\psi)} \left(\Omega_1'\Phi_1 + \Omega_2'\Phi_2\right)}{\sum\limits_{\psi} \left|\beta(\psi)\right|} \quad \textbf{(16)}$$

8.2. Mutation analysis

In order to maintain feasibility in a chromosome during mutation, the x_{ijrs}-variables have to be modified in pairs at the bit level within a specific size. Moreover, the network-knapsack formulation of the problem implies that mutations have to take place in strings of consecutive sizes $\boldsymbol{\sigma}$. For a particular $\boldsymbol{\sigma}$, a feasible mutation comes down to a rearrangement of the number of stencils for sizes within $\boldsymbol{\sigma}$. Let now Σ_{ι_k} stand for the set of ι_k ($\iota_k \geq 2$) consecutive sizes (the network formulation implies there is one exception for the ι_k -constraint: $\Sigma_{\iota_k} = \{|P|\}$ for ι_k=1) and let $\theta_{\iota_k}(\gamma_k)$ stand for the number of rearrangements when γ_k stencils are laid in a string of length ι_k. For an even number of mutations i, feasibility maintenance boils down to combining elements σ_{ι_k} of several sets Σ_{ι_k}, satisfying $\sum_k \iota_k = i/2$, and by rearranging the number of stencils in each σ_{ι_k}, taking into account the total flow in & out of σ_{ι_k}. Let now $\mathop{\times}\limits_{k=1}^{j} \Sigma_{\iota_k}$ stand for the Cartesian product of sets Σ_{ι_k}, enforcing $\mathop{\times}\limits_{k=1}^{j} \Sigma_{\iota_k} \cap \mathop{\times}\limits_{k=1}^{\bar{j}} \Sigma_{\iota_k} = \varnothing, j \neq \bar{j}$, then it follows that the number of feasible mutation scenarios |**M**| can be written as in (17).

$$|M| = \left[\sum_{i=2,\text{even}}^{2|P|} \sum_{j=1}^{\lceil i/4 \rceil} \sum_{\sum_{k=1}^{j} \iota_k} \left| \mathop{\times}_{k=1}^{j} \Sigma_{\iota_k} \right| \prod_{k=1}^{j} \left(\theta_{\iota_k} (\gamma_k) \right) \right]^{|R|} \quad \textbf{(17)}$$