Design Process Sequencing With Competent Genetic Algorithms

Christoph Meier
Institute of Astronautics,
Technische Universität München,
85,748 Garching, Germany
e-mail: c.meier@tum.de

Ali A. Yassine[1]
Department of Industrial & Enterprise,
Systems Engineering (IESE),
University of Illinois at Urbana-Champaign,
Urbana, IL 61801
e-mail: yassine@uiuc.edu

Tyson R. Browning
M.J. Neeley School of Business,
Texas Christian University,
TCU Box 298530,
Fort Worth, TX 76129
e-mail: t.browning@tcu.edu

In product design, it is critical to perform project activities in an appropriate sequence. Otherwise, essential information will not be available when it is needed, and activities that depend on it will proceed using assumptions instead. Later, when the real information is finally available, comparing it with the assumptions made often precipitates a cascade of rework, and thus cost and schedule overruns for the project. Information flow models have been used to sequence the engineering design process to minimize feedback and iteration, i.e., to maximize the availability of real information where assumptions might otherwise be made instead. In this paper, we apply Genetic Algorithms (GAs) to an information flow model to find an optimized sequence for a set of design activities. The optimality of a solution depends on the objective of rearrangement. In an activity sequencing context, objectives vary: reducing iteration/feedback, increasing concurrency, reducing development lead-time and cost, or some combination of these. We adopt a matrix-based representation scheme, the design structure matrix (DSM), for the information flow models. Our tests indicate that certain DSM characteristics (e.g., size, sparseness, and sequencing objective) cause serious problems for simple Genetic Algorithm (SGA) designs. To cope with the SGA deficiency, we investigate the use of a competent GA: the ordering messy GA (OmeGA). Tests confirm the superiority of the OmeGA over a SGA for hard DSM problems. Extensions enhancing the efficiency of both a SGA and the OmeGA, in particular, niching and hybridization with a local search method, are also investigated. [DOI: 10.1115/1.2717224]

Keywords: engineering design process, design process optimization, genetic algorithm, design structure matrix

1 Introduction

Product design and development is recognized by many firms as a crucial activity. The ability to quickly introduce new products into the market is a key factor for determining corporate health and profitability [1]. As a result, design management researchers embarked on studying the design and development process for complex engineering products in order to simultaneously reduce development lead-times, reduce cost, and improve quality [2]. Several formal theories of engineering design emerged [3,4], as well as multiple books on product design and development [5–7]. While this literature fails to agree conclusively on a single theory of engineering design and corresponding best management practices, it seems to agree on two important facts: (1) Design iteration (hereafter, iteration) is a typical characteristic of complex design processes [8–11] and (2) it is a major source of increased product development lead-time and cost [12,13]. These facts make iteration a central issue in the management of product design and development [14,15].

Iteration represents repeated design refinements due to information feedback. Feedback occurs when a design activity proceeds with missing (or uncertain) input information; the actual information is received (or changes) at a later time. The late arrival (or future change) of vital input information could be due to detected errors, failure to meet requirements, or changing design directives, to name but a few of the common causes. There is no comprehensive classification scheme for iteration, although the literature refers to different types of iteration as planned versus unplanned, sequential versus parallel, good versus bad, and major versus minor [16]. It is clear that effective management of iteration is required to plan and control project cost, duration, technical performance (or quality), and risk. Research has shown that modeling provides an avenue to improved understanding and management of design processes [17].

Iteration can be managed in many ways, including by: improving the sequence of design activities to streamline information flows, which form the basis for task dependencies [18]; developing new engineering automation tools to perform iterations faster [19]; adding resources at bottleneck activities [20]; and possibly limiting the scope of the development effort. In this paper, we present a method for design process sequencing, which streamlines information flows between the design activities, using a digraph-based information flow model represented as a design structure matrix (DSM)[2,3]. This proposed technique is useful in many areas, such as process and project management [13,21], organization of computational subroutines for multidisciplinary design optimization [22,23], design problem decomposition [24–26], and product decomposition [27–29]. Regardless of the decomposition domain (i.e., product or process), reordering the elements within the decomposed system is necessary, either to (a) cluster components that belong to the same system together so that system boundaries could be easily identified [29], (b) determine design subproblems that can be solved independently [22], or (c) identify the feedback elements to be avoided in a project or process [13].

As shown in Fig. 1, a binary DSM is equivalent to the adjacency matrix of the underlying digraph. The *n* nodes of the digraph (representing the activities in a process) correspond to the *n* column and row headings in the matrix. The arrows, indicating a

[1]Author to whom correspondence should be addressed.

Contributed by the Design Theory and Methodology Committee of ASME for publication in the Journal of Mechanical Design. Manuscript received March 6, 2006; final manuscript received October 12, 2006. Review conducted by P. John Clarkson.

[2]The term *dependency structure matrix* is also commonly used.

[3]All of the situations we model can also be represented using directed graphs (digraphs). We choose the DSM merely as a representation scheme because of its (1) ability to visually highlight iterative loops among coupled components and (2) amenability to matrix-based analysis techniques.

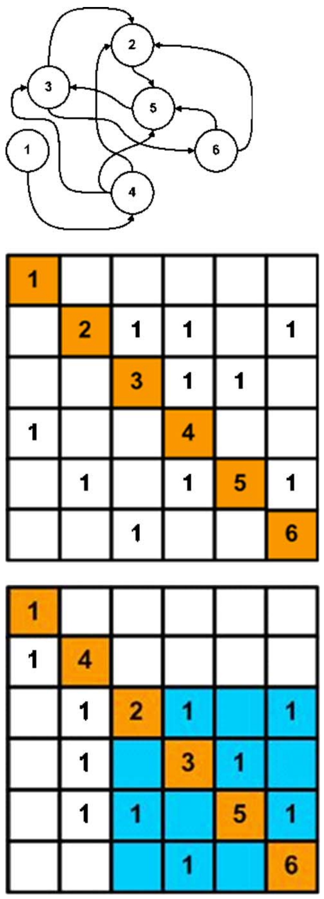


Fig. 1 A small process model shown with digraph and matrix representations

relationship between process elements, correspond to the marks inside the matrix. Therefore, a DSM is a square matrix that shows design process elements (hereafter, activities) along its diagonal, activity outputs in its columns and activity inputs in its rows. Thus, feed-forward information in the process is shown below the diagonal, and nonzero superdiagonal cells indicate the presence of feedback[4]. As the number of activities and relationships grows, DSMs provide advantages over digraphs in visualization of feedback loops, which are immediately obvious as super-diagonal marks. The design process can be sequenced (i.e., the information flow streamlined) by reordering the activities, such that feedback, and thus iteration, is minimized. An ideal sequence is defined as one with no feedback marks, represented by a lower-triangular DSM. In real and complex product development processes, an ideal sequence is unlikely to exist [19,26]. Design process optimization then becomes a problem of finding the sequence of design activities that minimizes feedback[5].

DSM sequencing can be formulated as a *quadratic assignment problem* (QAP), which is a well-known, NP-hard combinatorial optimization problem. At present no algorithm can be found to solve NP-hard problems of arbitrary size to optimality within an adequate time using limited resources. Although some algorithms (e.g., hierarchical cluster analysis [26] and partitioning [30–32]) are able to provide a block-angular form for the DSM, the optimal sequence for activities within the blocks (called coupled activities or iterative blocks) remains to be determined. Coupled activities or DSM blocks, indicated by the block in Fig. 1(*c*), can be found efficiently using partitioning algorithms [32,33]. These blocks are disjoint, so sequencing the activities within a DSM's blocks constitutes a set of disjoint QAPs. An exhaustive search algorithm is only reasonable for small DSM blocks (ensuring the detection of the global optimum); thus, other optimization techniques must be considered for medium or large blocks. Particularly for large blocks, Genetic Algorithms (GAs) [34] compare well with other popular meta-heuristic optimization techniques such as Simulated Annealing [35], Taboo Search [36,37], and Ant Colony Optimization [38].

In brief, a GA works as follows. A feasible activity sequence is encoded as a chromosome, while multiple chromosomes form a GA population. By selecting the fittest chromosomes (i.e., the ones with the fewest feedbacks) and applying the ordinary genetic operators—crossover and mutation—the population improves over time. The GA proceeds until a predefined convergence criterion is reached. In the context of DSM sequencing, the fitness of a solution depends on the objective of rearrangement. In order to test the performance of different GA designs and parameters on difficult DSM problems, we rely on artificial, deceptive objective functions.

This paper makes three contributions. First, prior to any optimization, we present an investigation of the difficulty of DSMs based on their characteristics, an aspect neglected in past literature. Our tests indicate that sparse DSM blocks, in combination with certain sequencing objectives, cause serious problems for *simple* GA (SGA) designs. The reason for this behavior is a problem called *symmetry*, where too many solutions in the search space have identical fitness values, thus providing less guidance to the best sequence. Second and third, to enhance the performance of a SGA design, we present an investigation of two improvement opportunities. Initially, we combined the GA, which usually searches more globally in the search space, with a fast local search strategy. This *hybridization* required fewer necessary resources than a SGA to obtain the same solution quality. However, the performance on hard, artificial problems remains poor. Thereafter, we applied a state-of-the-art GA design, called a *competent* GA, and extended it with a local search strategy. The performance improved significantly on difficult problems; however, the resources required to gain a certain level of accuracy were higher on easy problems (compared to the hybrid SGA).

The rest of this paper is organized as follows. In the next section, we review the DSM methodology, the various sequencing objectives that can be applied, and DSM sequencing problem complexity. We introduce the use of SGAs for DSM sequencing in Sec. 3, discussing the various genetic operators—parameters that play a role in determining the speed and quality of the DSM sequencing problem—and GA efficiency improvement techniques. Section 4 presents competent GAs. In Sec. 5, we discuss the two most important parameters to set for any GA: population size and the number of generations. To compare the performance of each GA approach, we present test results for the simple and competent GAs using deceptive functions (Sec. 6) and other typical DSM objective functions (Sec. 7). Section 8 describes potential extensions and concludes the paper.

2 Design Process Modeling Using the Design Structure Matrix (DSM)

An understanding of the design process is a necessary precursor to enhancing its performance [17]. Traditional diagrams such as PERT (Project Evaluation and Review Technique), CPM (Critical Path Method), and Gantt charts lack the ability to capture the entire design process complexity because they do not incorporate iterations and coupled relationships between activities. The DSM

[4]Some DSM literature uses the opposite convention (the transpose of the matrix), with inputs in columns and outputs in rows, and thus feedback below the diagonal; the two conventions convey equivalent information.

[5]In the rest of the paper, the terms "design process sequencing" and "DSM sequencing" are used interchangeably.

has been shown to support design process understanding and provides a more suitable model for its improvement [13,31]. The utility of a matrix representation lies in its ability to illuminate complex relationships between process elements in a compact, visual, and analytically advantageous format [39]. The cells on the diagonal of a DSM represent activities to be performed and off-diagonal marks represent information flows or dependencies among the activities. If activities are executed in the same order in which they are listed in the DSM, then subdiagonal marks represent forward dependency or information transfer to later (i.e., downstream) activities. Marks above the diagonal depict information fed back to earlier activities (i.e., feedback marks). Upstream activities, requiring unavailable feedback marks as inputs, will commence (initially) using a guess or estimate as a proxy for that missing information. Subsequent iterations of the upstream activity will be necessary if such estimates, once verified, turn out to be problematic. Moreover, such changes in an upstream activity can precipitate a cascade of changes through the downstream activities that used its outputs. Thus, feedback marks or dependencies should be avoided, in general, since they indicate the use of assumptions in lieu of actual information, and thus potential iteration and rework.

2.1 DSM Partitioning and Sequencing. We distinguish between two closely related DSM algorithms: partitioning and sequencing. Partitioning is concerned with finding a topological sort for the activities when no cycles exist. When feedback loops exist, partitioning will only identify the activities involved in these cycles but will not provide a sequence for them [19]. This is analogous to transforming a DSM into a block-angular form[6]. Sequencing algorithms, alternatively, are concerned with ordering the activities within an iterative block.

Numerous partitioning algorithms can be found in literature [30–32,40,41]. Partitioning separates a set of activities into two disjoint subsets: a set of activities with sequential or parallel relationships and a set of activities with coupled relationships. Coupled activities cannot be reordered in a way to yield no superdiagonal marks in a DSM. The identification of sequential or parallel relationships is trivial, and its worst case complexity is $O(n^3)$. In contrast, coupled relationships are identified by determining all cycles in a DSM. Some algorithms for identifying cycles, such as the powers of the adjacency matrix, are inefficient and can significantly reduce the performance of partitioning algorithms. However, for partitioning purposes, it is unnecessary to identify all cycles; it is sufficient merely to find all the *strongly connected components* (SCCs) in the DSM to identify all the coupled blocks[7]. These blocks can be identified with a linear scale-up behavior in $O(n)$ using a modified Depth First Search algorithm introduced by Tarjan [32,33][8]. This efficient algorithm makes the size of any DSM practically unimportant for identifying coupled blocks. When each coupled block is collapsed into a single representative activity, then a lower-triangular arrangement is easily obtained by any partitioning algorithm. Thus, the objective turns into how to sequence elements within a single coupled block, which can be performed concurrently on all coupled blocks of a DSM.

Many sequencing objectives have been defined for iterative blocks. Most focus on minimizing the superdiagonal marks [31,41]. Another sequencing objective, the Gebala objective [40], seeks a process with superdiagonal relationships as close to the diagonal as possible. The motivation is twofold. First, the initial guesses that these feedback marks represent would have a higher quality, since the information to be assumed is closer in time and possibly more certain. A second argument is the fact that marks closer to the diagonal cause shorter iterations in the process than marks far from the diagonal. Hence, the rework caused by the iterations would probably be reduced. McCulley and Bloebaum [22] claimed that the reduction of "crossovers" should be considered in order to minimize further iterations in the process[9]. Todd's [42] objective is to increase process concurrency by pushing marks to the lower-left corner of the DSM. Other researchers proposed a joint optimization of both iteration and concurrency [43–45]. Finally, other unique objectives, such as clustering [46,47] and maximization of the net option value of a modular design [48] have also been proposed. Table 1 provides a summary of the various sequencing objectives. Note that these proposed objectives are all proxies for the actual goals of decreasing the process's overall duration, cost, and risk.

2.2 Problem Complexity. DSM block sequencing can be formulated as a QAP [50]. The QAP in the DSM context can be described as the problem of assigning a set of activities to a set of locations with a given flow between the activities and given distances between the locations. The objective is to assign the activities on locations in such a way that the sum of the product between flows and distances is minimal. Formally, given n activities (in a DSM block) and n locations, two $n \times n$ matrices $\boldsymbol{A}=\{a_{ij}\}$ and $\boldsymbol{B}=\{b_{ru}\}$ with a_{ij} as flow between activities i and j and b_{ru} as the distance between locations r and u, the objective can be described as

$$\min_{\psi \in S(n)} \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} \cdot b_{\psi(i)\psi(j)} \tag{1}$$

where $S(n)$ represents the set of all permutations, and $\psi(i)$ gives the location of activity i in the current solution, $\psi \in S(n)$. Now, the QAP($\boldsymbol{A}, \boldsymbol{B}$) is to find a permutation $\psi \in S(n)$ that minimizes Eq. (1). The QAP formulation requires the definition of a distance matrix and a flow matrix. The DSM is essentially the flow matrix. Determination of the distance matrix depends on the DSM sequencing objective.

Recognizing that DSM sequencing can be expressed as a QAP is important to determine the complexity of the problem itself. Since QAPs belong to the class of *NP-hard* problems [51], then DSM sequencing, independent of any objective, is in this complexity class as well. Partitioning is able to split the DSM into φ disjoint sets of SCCs, but the problem is still NP-hard, since each SCC still represents a QAP. Regardless of the objective, a DSM with n elements has $n!$ potential sequences. Depending on the underlying DSM parameters—such as the number of relationships between activities (density)—and sequencing objective, the feasible search space of all $n!$ process architectures looks different. This aspect has been ignored in past DSM literature, where no attempt was made to visualize the search space of a DSM based on its parameters. However, as the performance of a DSM sequencing technique will depend greatly on the difficulty of its corresponding search space, it is essential to identify critical DSM parameters. This would permit the categorization of DSMs by level of difficulty and the utilization of individual optimization strategies. The greatest obstacle is the huge number of feasible solutions ($n!$), which allows a complete visualization of the search space only for small DSMs. For example, Fig. 2 depicts three regional search space "landscapes," near the global optimum for an 8×8 DSM with no iteration (and thus an optimal objective function value of zero), according to different DSM

[6]In a fully coupled DSM, partitioning will not change the original sequence.

[7]In graph theory, sets of vertices in which every vertex can reach every other vertex are referred to as SCCs. By definition, every vertex can reach itself and forms a SCC of size 1.

[8]Precisely, the complexity is $O(n+a)$, where n is the number of activities (vertices) and a is the number of relationships (arcs) in the DSM.

[9]Crossovers in this context can be understood as the crossing of feedback of one activity with another without exchanging information through the intersection.

Table 1 An overview of sequencing objective functions in the literature

Author	Objective	Rationale	Objective function equation
Steward [31]	Minimize number of feedback marks	Reduce iteration	$f=\sum_{i=1}^{n}\sum_{j=i+1}^{n} w(i,j)$ where $w(i,j)$ is a binary value in cell (i,j)
Kusiak [41]	Minimize weighted sum of feedback marks	Reduce iteration	$f=\sum_{i=1}^{n}\sum_{j=i+1}^{n} w(i,j)$ where $w(i,j)$ is a weighted value bet. 0 and 1 in cell (i,j)
Gebala and Eppinger [40]	Minimize total feedback length; i.e., Minimize distance from diagonal	Reduce iteration length	$f=\sum_{i=1}^{n}\sum_{j=i+1}^{n} (j-i)\cdot w(i,j)$ where $w(i,j)$ is a weighted value bet. 0 and 1 in cell (i,j)
Todd [42]	Push marks to lower left side of DSM as much as possible	Increase/improve concurrency	$f=\sum_{i=1}^{n}\sum_{j=i+1}^{n} j\cdot w(i,j)$ where $w(i,j)$ is a weighted value bet. 0 and 1 in cell (i,j)
Scott [43]	Jointly reduce feedback marks and move marks to bottom left-hand corner of DSM	Jointly minimize iteration and improve concurrency	$f=\sum_{i=1}^{n}\sum_{j=i+1}^{n} \Omega(i,j)\cdot w(i,j)$ where $\Omega(i,j)=j+(n-1)^2$ if $i>j$ and $\Omega(i,j)=100[j+(n-1)^2]$ if $i<j$
McCulley and Bloebaum [22]	Minimize feedback and crossovers	Reduce number of tasks involved in an iteration loop	$F=w_f.f+w_c.c$, where f and c are the number of feedbacks and crossovers respectively. w_f and $w_c$are the associated user-defined weights.
Rogers [49]	Minimize time/cost	Traditional PM objectives	$F=1/(w_f.f+w_c.c+w_{\text{time}}.\text{time}+w_{\text{cost}}.\text{cost})^4$,where w_{time} and w_{cost} are user-defined weights for the total time and cost required to converge the process.
Altus et al. [23]	Minimize computation time of the decomposed problem	Reduce computation overhead	Problem dependent
Whitfield et al. [45]	Jointly minimize feedback marks and improve concurrency	Jointly minimize iteration and improve concurrency	Uses heuristics to identify parallel activities within a DSM
Yu et al. [46]	Clustering—improve modularity	Information theoretic measure of fitness	$f=(1-\alpha-\beta)\cdot(n_c\log n_n+\log n_n\sum_{i=1}^{n_c} cl_i)+$ $\alpha\cdot[\lvert S_1\rvert(2\log n_n+1)]+\beta\cdot[\lvert S_2\rvert(2\log n_n+1)]$ where n_c is the number of clusters in theDSM, n_n is the number of nodes, cl_i is the number of nodes in the ith cluster, and α and β are weights between 0 and 1.
Whitfield et al. [47]	Clustering—improve modularity	Get the dependencies as close to the leading diagonal as possible	$f=\sum_{i=1}^{n}\sum_{j=i+1}^{n} \lvert j-i\rvert\cdot w(i,j)$
Baldwin and Clark [48]	Clustering—improve option value	Maximize net option value	For a system of j modules the net option value is $\text{NOV}_T=\text{NOV}_1+\cdots+\text{NOV}_j$ where $\text{NOV}_i=\max\{\sigma_i(n_i)^{1/2}Q(k_i)-C_i(n_i)k_i-Z_i\}$ n_i is the number of activities inside module i, $\sigma_i$is the standard deviation of potential value, k_i is the number of competitors for module i, $Q(k_i)$ is value of the best of k designs, $C_i(n_i)$ is total development cost, and Z_i is the visibility of the module to the system.

parameters[10].The first two landscapes in Fig. 2 are based on the Kusiak objective [41] and differ only in their density, i.e., the number of current marks in the DSM a divided by the number of possible marks (given by n^2-n). Not surprisingly, the sparse DSM exhibits less discrepancy between the objective values. From an optimization point of view, it is more difficult to detect the best solution in a search space with low discrepancy, because it does not exhibit enough guidance to the global optimum. This problem is called *symmetry*; empirical tests will refer to this issue later in the paper. On the other hand, the likelihood of identifying a relatively "good" solution is rather high for a sparse DSM, as discrimination of this search space is not high and many more

[10]As a tool for visualization we used Microsoft® Excel, which has the capability to plot only 2^{15} values on the x-axis. Other tools may be used to plot larger landscapes. We used the algorithm for the generation of permutations, described in [33], because subsequent permutations differ only at one position and can thus be regarded as "neighbors" in the search space. Every permutation π can be regarded as a point in the design space Φ_d (x-axis) with a corresponding point in its objective space Φ_o (y-axis), depending on the defined objective function. Essentially, Fig. 2 plots the relationship $f(\pi):\Phi_d\rightarrow\Phi_o$, where Φ_o corresponds to all real numbers.

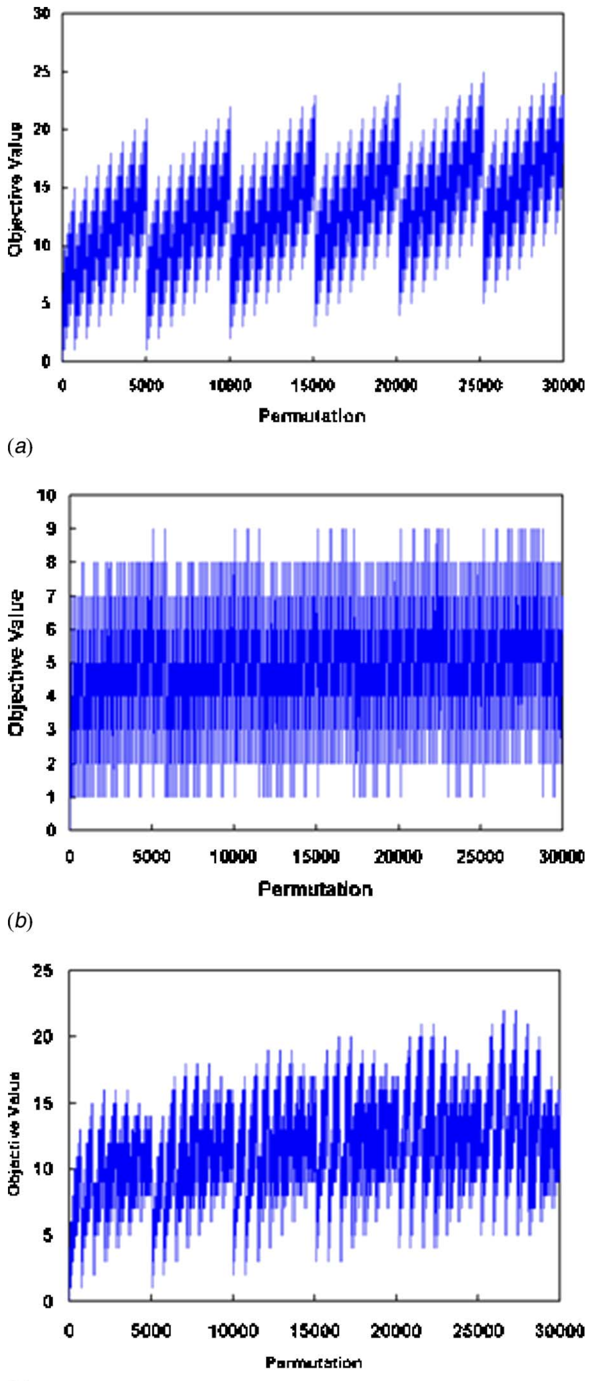


Fig. 2 Visualization of a subset of the search space for an 8 × 8 DSM block; three different landscapes with varied densities and and sequencing objectives are shown. (*a*) Kusiak objective – 50% density. (*b*) Kusiak objective – 15% density. (*c*) Gebala objective – 15% density.

second-best solutions exist than in case of the other two landscapes. Inspecting the landscape for a sparse DSM with respect to the Gebala objective [40] in Fig. 2(*c*), the appearance of the symmetry problem is much less pronounced than with the Kusiak objective. Thus, the landscape of the search space is very sensitive to changes in the problem structure and exhibits varied levels of problem complexity.

2.3 Using GAs for DSM Sequencing. The application of GAs to the sequencing of DSMs has already attracted several researchers. Rogers [49] developed a tool called DEMAID (Design Manager's Aid for Intelligent Decomposition) and extended it with a GA to support design managers' process structuring decisions. The resulting software, DEMAID/GA, is reported to successfully minimize iterative cycles in a DSM, resulting in reduced development time and cost. McCulley and Bloebaum [22] developed another software tool called GENDES (Genetic Design Sequencer) that resequences a DSM using a GA according to user-defined objectives. The authors accomplished numerous tests related to the GA parameters as well as to sequencing objectives and compared the performance of Gendes with DEMAID. A third software tool called AGENDA (A GENetic algorithm for Decomposition of Analyses [23]) supports decomposition of multidisciplinary design optimization (MDO) problems using GAs as the optimization technique and DSMs as the system model. Almost no significant differences in the description of the underlying GA and its operators, compared to DEMAID/GA and GENDES, can be regarded. In contrast, Whitfield et al. [44] mainly investigated and compared GA crossover and mutation operators for sequencing DSMs. Table 2 summarizes the DSM/GA research. These previous publications recognized the power of GAs in the field of combinatorial problems but did not exploit its entire repertoire. In particular, precise information about the setting of GA parameters and information about the motivation behind the use of certain GA operators is missing or only based on partial tests. Pivotal enhancements of GAs such as hybridization or niching techniques were not even considered. Furthermore, since a powerful class of GAs—the competent GAs—has not been applied to DSM sequencing problems, a state-of-the-art investigation is essential.

In the next two sections, we present two GA designs in detail: the SGA and the competent GA. In each case, we provide the necessary background information about the appropriate setting of parameters and the adequate choice of operators. In addition, we explore extensions of those designs through niching and hybridization strategies.

3 Process Optimization Using a Simple Genetic Algorithm

This section introduces and investigates the use of SGA designs (i.e., GAs that perform selection, crossover, and mutation) to find optimal DSM sequences for a predefined objective. This section introduces basic GA terms and information about the GA operators and the setting of its parameters. Since a thorough investigation of GA settings, and their appropriateness for and effects on various kinds of sequencing problems, has not been done in the previous literature on this subject, we feel that the exposition in this section is helpful in that regard.

As shown in Fig. 3, a SGA starts by creating an initial *population* of chromosomes. Each *chromosome* contains a collection of *genes* (each of which adopts a value called an *allele*) and represents a complete solution to the problem at hand. A *fitness* function (the objective function) measures how good each chromosome's solution is. The alleles are usually initialized to random values (within user-specified boundaries), thus providing a variety of fitness levels. Next, *selection* is performed to choose chromosomes that will have their information passed on to the next generation. Chromosome *crossover* produces new offspring chromosomes. Crossover occurs according to a user-defined probability p_c (usually high) and results in new chromosomes having characteristics taken from both of the parent chromosomes. If an offspring takes the best parts from each of its parents, the result will likely be a better solution. The genes of the offspring chromosomes are also mutated. *Mutation* occurs according to a user-defined probability p_m (usually low) and introduces variability into the gene pool. The new generation of chromosomes replaces the previous population, and the fitness of the new generation is evaluated. These steps repeat until a termination condition is met, which could simply be the number of generations or fitness convergence in the population.

Table 2 Comparison of DSM-based GA research on process sequencing

Study	GA design	Encoding	Selection operator	Crossover operator	Mutation operator	Crossover probability	Mutation probability
DEMAID/GA [49]	Simple	Integer	Tournament (Size 2)	Position-based	Order-based	Not reported	1%
GENDES [22]	Simple	Integer	Tournament (Size 2)	Position-based	Order-based	Not reported	0.1%
AGENDA [23]	Simple	Integer	Tournament (Size 2)	Position-based	Point-wise	90%	3%
Whitfield et al. [44]	Simple	Integer	Roulette-Wheel	Position-based, Version 2	Shift	60%	20%
This Paper	Competent with Hybridization	Integer and Binary	Tournament (Size 4)	Position-based, Version 2	Shift	100%	1/4l, where l is the problem length

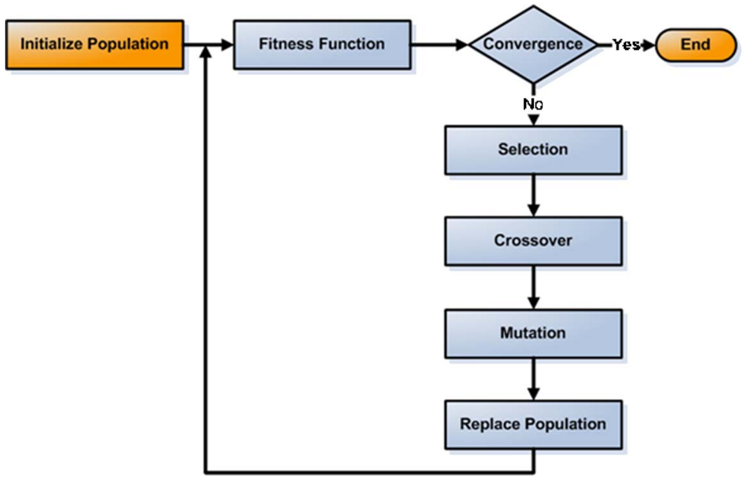


Fig. 3 SGA flowchart

The power of GAs lies in their ability to perform global search by sampling partitions of hyperplanes in the search space instead of visiting every point in the search space. The so-called *schemata* processed implicitly by a GA correspond to hyperplanes in this search space. Schemata are defined as *similarity templates* describing a subset of chromosomes with similarities at certain loci [52,53]. Using an alphabet—e.g., the binary alphabet $\{0,1\}$—as representation for chromosomes, an additional "do not care" symbol, typically represented by an asterisk ("$*$") completes the alphabet. Thus, the alphabet for schemata using bit representations is $\{0,1,*\}$[11]. As an example, consider the schema H($*$011$*$) of length $l=5$, which matches the four strings {10110, 10111, 00110, 00111}. Furthermore, this schema H is of order $O(\mathrm{H})=3$, as the order of a schema represents the number of fixed bit positions. Another property of schemata is their defining length $\delta(\mathrm{H})$, which is the distance between the first and last fixed bit positions[12].

By looking at the fundamental theorem of GAs, the *Schema Theorem* [34], we obtain a lower bound on the change of the sampling rate for a single schema from generation t to $t+1$. As a result, schemata with above average fitness, short defining length, and low order survive with higher probability than those with low fitness, long defining length, or high order. These useful schemata are called *building blocks* (BBs) [54]. In fact, it was shown that an exponential number—about n^3—of these useful schemata are implicitly sampled by the GA while processing n chromosomes on the surface [34]. BBs can be understood as portions of a solution contributing much to the global optimum. The constant juxtaposition and selection of BBs forms better and better solutions over time, leading to a global optimum in a search space [54]. This is referred to as the *Building Block Hypothesis*.

3.1 Components: Data Structure and Fitness Function. As an input to the GA, a particular sequence of activities must be represented as a *chromosome*, which constitutes the data structure in the GA. Various representation models for encoding permutations as chromosomes in GAs exist, but the most common is the natural encoding by integer numbers. In general, for permutation problems, a specific element has to be assigned exactly once to a locus in the permutation; otherwise, the permutation is not feasible. As shown in Fig. 4, each activity of the DSM is represented exactly once in the chromosome.

Another encoding possibility is the use of binary representation. An earlier approach to binary representation was the use of binary matrices describing the relative order of the elements within a permutation. However, this approach fails to exclude nonexistent sequences, so it is not ideal [55]. Nevertheless, we will describe in Sec. 4.1 how binary encoding can be used for combinatorial problems using *random keys* [56].

3.2 Mechanics

3.2.1 Selection. To find a good selection operator, the *selection pressure* (SP) is decisive. SP is defined as the number of

[11]However, it is worth noting that the representation is not limited by a binary alphabet and schemata representation is thus different.
[12]Hence, the schema H($*$011$*$) has a defining length $\delta(\mathrm{H})=4-2=2$.

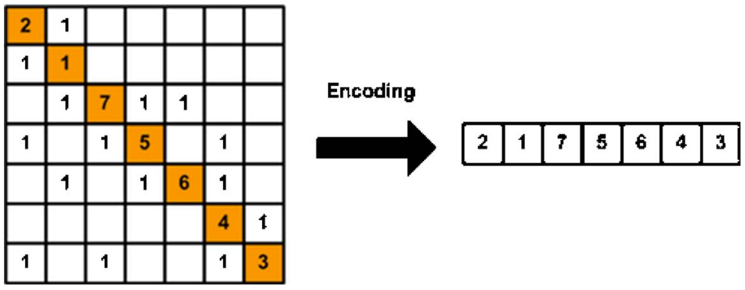


Fig. 4 Encoding a DSM as a chromosome via integer representation

expected copies of the best chromosome in the next generation [57]. In determining this value, one has to consider that SP decreases over time as the structure of chromosomes (and their fitness) becomes more and more similar and finally (depending on the probability of mutation) only the best chromosome's genotype will remain at the end of a GA run. When SP is low, a *genetic drift* occurs, causing the GA to converge arbitrarily on a solution [58]. In contrast, an SP that is too high results in *premature convergence*[13] and *cross-competition*[14]. Pushy selection schemes with adequate, high SP must be preferred initially but relaxed later in a run.

There are two popular types of selection schemes: fitness-proportionate selection schemes (e.g., roulette wheel selection [54] and stochastic universal sampling [59]) and ordinal-based selection schemes (e.g., tournament selection [60] and truncation selection [61]). Fitness-proportionate schemes may often fail to provide adequate SP when fitness variance in the population is very high or very low. For instance, in a population with high fitness variance, only a certain portion of the population with high fitness is selected, leading to a premature extinction of other BBs that could be necessary for detection of the global optimum. In order to reduce extreme fitness variance in the population, *fitness scaling* methods can balance the fitness of all chromosomes prior to selection via fitness-proportionate schemes. Examples of such scaling mechanisms include the Sigma Scaling [63], linear/nonlinear fitness scaling [54], and the Boltzmann selection operator using an annealing schedule [64,65]. While these scaling operators reduce the problems of fitness-proportionate selection, they cannot completely eliminate it.

We favor ordinal-based selection schemes because of their ability to ensure an adequate SP independent of a specific fitness structure within the population. Tournament selection is very popular [60]. In Tournament Selection, a certain number of chromosomes is randomly selected, depending on the tournament size s. The best chromosome wins the tournament with probability p and overcomes the selection phase. Here, a chromosome's fitness *rank* within the tournament is decisive rather than the magnitude of its fitness value. While tournament selection can be implemented either with (TWR) or without replacement (TWOR)[15], Sastry [52] found TWOR to be superior because it is less noisy and needs a lower population size to achieve a given level of accuracy. The SP in the important, early generations of the GA run (when the proportion of best individuals is insubstantial), using TWOR, is $\mathrm{SP}=s\cdot p$ [62]. In the remainder of this paper we assume a probability of 1.0 that the best chromosome of a tournament will be selected, and hence $\mathrm{SP}\approx s$ holds in the early generations.

3.2.2 Crossover. Crossover enables a global exploration of the search space. Since crossover operators must consider the underlying encoding of the chromosomes to generate feasible solutions, special operators are necessary when dealing with an integer encoding. Following Whitfield et al. [44], who compared several crossover strategies for DSM sequencing, we implement a version[16] of position-based crossover [66] that works as follows. First, two chromosomes are chosen randomly from the "mating pool" (the chromosomes that passed the selection stage), one of which is randomly designated as the "primary" parent. These two chromosomes undergo crossover according to the *crossover probability* p_c. Each of the primary parent's genes has a 50% chance of being inherited by the offspring at the same locus in the chromosome. To fill the offspring's remaining loci, it inherits the remaining genes from the second parent to create a feasible solution. That is, the secondary parent is scanned from left to right, and genes not present in the offspring are inherited in the order in which they appear in the second parent's chromosome. Figure 5 provides a graphical example of this process. The offspring first inherits four genes from parent 1 at loci (1,2,3,6) and then the remaining genes from parent 2 at loci (4,5,7,8).

Although the Schema Theorem [34] predicts BB growth for low p_c and high SP, the previous section described the problem of high SP leading to premature convergence. Similarly, a low p_c will inhibit the BBs from mixing and forming higher-order BBs. Thus, a reasonable combination of p_c and SP must be found. *Control maps* provide information about the relationship between the performance of a GA at a certain p_c and SP [67]. A theoretical control map for an easy[17] problem is depicted in Fig. 6(*a*) and was empirically verified by Goldberg, as shown in Fig. 6(*b*) [68]. Inside the map, an area where the GA performs well (i.e., where the GA can reliably identify the global optimum—the so-called "sweet spot") is limited by three bounds that must be overcome: the (genetic) drift boundary, the mixing boundary, and the cross-competition boundary. These control maps indicate that the sweet spot is surprisingly large for easy problems. p_c has to be increased with increasing SP, but the sweet spot can be hit with $p_c=1.0$ even for an extremely high SP of approximately 20. In contrast, the sweet spot is smaller for a more difficult problem [69].

A linear relationship between the logarithm of SP and p_c can be observed in these control maps and can also be forecast theoretically. For this purpose, Thierens [69] defined a relationship between the two fundamental mechanics in a GA: selection and crossover (i.e., mixing of BBs). For a successful GA, the mixing time must be less than the selection time[18]. Using takeover-time

[13]At premature convergence, the *takeover time*—the time at which an individual dominates the population—is less than the *innovation time*—the time for any operator to achieve a solution better than any other solution at this point [62].

[14]Here, good schemata have to compete against each other due to the high SP, causing incomplete schema coverage and greatly reducing the probability of finding a good solution.

[15]In TWR, all individuals who participated in a tournament are also candidates for the following tournament without any constraints. Consequently, the best string is expected to obtain s copies on average. In contrast, TWOR requires s "rounds" of selection, where individuals may participate in a tournament only if all other chromosomes have been part of a tournament in the same round. Thus, the best individual gets exactly s copies for further processing.

[16]Murata and Ishibuchi [37] distinguish between two versions of position-based crossover.

[17]The problem is the so-called One-Max Problem, described in [70].

[18]Mixing time is the number of generations until a population contains $n-1$ copies of the best individual. Selection time is the expected number of generations to obtain one mixing event, where a mixing event occurs if the crossover operator generates an offspring with more BBs of equal size than each parent [71].

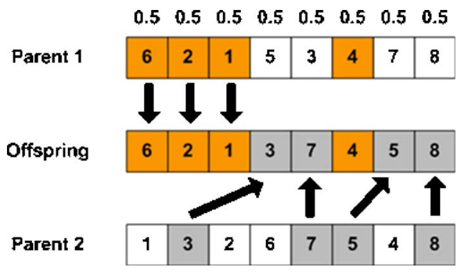
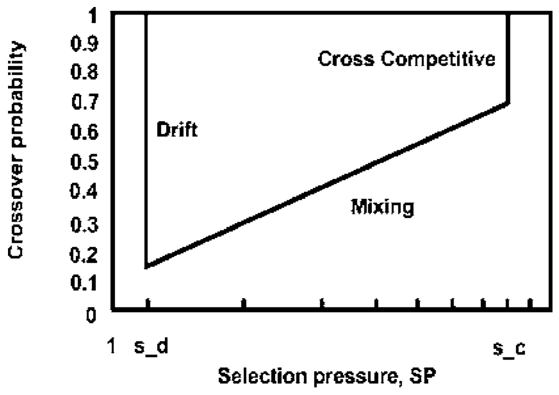
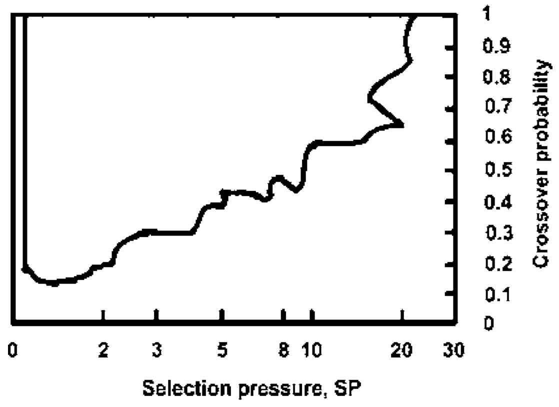


Fig. 5 Illustration of position-based crossover, version 2



(*a*)



(*b*)

Fig. 6 Theoretical and empirical predictions of the GA boundaries [67] for an easy problem. (*a*) Theoretical prediction, according to [67]. (*b*) Empirical results, according to [67].

models [62] for the selection time and a definition for mixing time [69], we obtain the following equation inter-relating the two

$$n_{\text{pop}}[\ln n_{\text{pop}} + \ln(\ln n_{\text{pop}})] > c\frac{2^{m+\mu k}\ln 2^m s}{p_c m^{2.5}} \tag{2}$$

where k is the BB order/size, m is the number of BBs, n_{pop} is the population size, μ is a calibration coefficient, and c is a constant. Since in tournament selection $\text{SP}\approx s$ in the early generations, we see the linear relationship between ln SP and p_c as well as the fact that a high p_c is expected to be least harmful. Although the BB structure of the problem is usually unknown a priori, a p_c of 1.0 in combination with a SP between 2 and 6 performs well. However, the dramatic increase of the mixing boundary for hard problems (with a large number and order of BBs; i.e., high k and m) is a serious problem for simple GAs that cannot be handled despite the choice of p_c or SP [58].

3.2.3 Mutation. In contrast to crossover, mutation exploits a certain region of the search space and can be understood as a local search. Mutation can be helpful when the effects of crossover diminish, diversity slowly disappears, and the GA begins to converge. Unlike crossover, mutation is able to produce new chromosomes, thereby maintaining diversity and possibly yielding better solutions. In this way, the GA proceeds with the search instead of converging prematurely. However, early work by De Jong [70] and Brindle [60] shows that high mutation rates will lead the GA to a random search. Furthermore, Goldberg and Segrest [72] point out that high mutation rates decelerate the convergence of a SGA. Thus, some authors [70,72–74] recommend a low, fixed *mutation probability*, p_m, for instance in the interval [0.001,0.01]. Furthermore, a p_m based on chromosome length, such as $p_m = 1/l$ [75], can be regarded as more reasonable than static probabilities, since the impact of mutation is coupled with chromosome length.

The traditional bit-flip mutation cannot be applied to integer-encoded chromosomes since feasibility must be ensured. Similar to crossover, various mutation operators for integer encoding were constructed to search in the near neighborhood of a permutation. Probably the easiest to implement is the single exchange of two randomly chosen genes, known as order-based mutation [76]. However, empirical results [44] suggest the superiority of a shift mutation operator [66], which works as follows. A random gene is chosen at loci A and B_1. A then exchanges the locus with B_1 and all x genes from B_1 to A (B_1 through B_m) are shifted accordingly one locus to the right or left. Figure 7 shows an example where the gene at locus 6 undergoes mutation and the second gene randomly chosen was gene number 3.

3.3 SGA Efficiency Enhancement Techniques. Several GA efficiency enhancement techniques exist, including parallelization [77–80], hybridization [81–83], time continuation [84,85], evaluation-relaxation schemes [86–89], and niching techniques [90]. We will introduce two of these techniques in more detail as they contribute to the GA performance of our test cases in Sec. 7[19].

3.3.1 Niching Techniques. Niching techniques are helpful as they restrict the likelihood of generating solutions in crowded regions and increase the likelihood of generating solutions in underpopulated regions. The evolutionary process maintains diversity by allowing species to occupy a separate niche. Individuals belonging to the same niche compete for limited resources whereas dissimilar individuals occupy different niches. In GAs, niching techniques maintain stable subpopulations within the GA by comparing chromosomes that are similar enough to occupy the same niche. Two main niching techniques exist: *crowding* methods and *fitness sharing* techniques. Fitness sharing has enjoyed the most success so far [90] and was therefore implemented in this paper. This technique alters the fitness of strings depending on the number of "similar enough" chromosomes in the population.

3.3.2 Hybridization Techniques. The second efficiency enhancement technique we explore is the *hybridization* of a GA with a local search strategy. The motivation for hybridization is the fact that GAs search more globally (high p_c and low p_m), thus potentially underexploiting certain interesting regions in the search space. With hybridization, the GA identifies promising regions and the local searcher "hill climbs" the optima within these fitness regions. Hybridization can significantly enhance the performance

[19]Depending on the underlying interests of the tests, the importance of investigating any particular efficiency enhancement technique varies. For instance, we do not want to primarily study CPU run times for a GA but to investigate its scale-up behavior in terms of objective function evaluations. Hence, we do not focus on parallelization strategies in this section; the reader may refer to the references quoted.

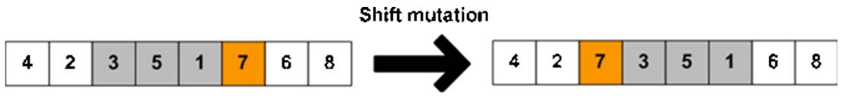


Fig. 7 Illustration of shift mutation

of GAs and plays a pivotal role in successful GA designs.

Several hybrid GAs can be found in the literature. Fleurant and Flerand [81] combined a GA with robust taboo search [91]. Ahuja [92] applied greedy GAs for large-scale QAPs with promising results. Evolution strategies and simulated annealing are also viable options. However, when Merz [83] compared different local searchers for the QAP, he chose the Genetic Local Search strategy applying the "2-opt heuristic" as a local searcher. The motivations for this choice were the simplicity of implementation and the speed of the 2-opt heuristic, while delivering good results. Speed in recognizing enhancements of the chromosome is an important advantage for the QAP. The 2-opt heuristic searches all of the possible permutations formed by exchanging two elements of a chromosome. This is essentially the repeated application of order-based mutation [76]. It is sufficient to search until the first improved permutation is found, thus saving computational time [83]. The number of possible exchanges for a chromosome of length n is given by $(n^2-n)/2$.

Since DSM sequencing can be defined as a QAP, the impact of an exchange on the fitness of a chromosome can be determined without calling the (computationally time-consuming) fitness function (assuming a deterministic fitness function). The formula for the QAP [93] can be adapted for DSM sequencing (by neglecting reflexive relationships between activities), yielding the following difference for exchanging ψ_x and ψ_y

$$\begin{aligned}\Delta(\psi,x,y) &= b_{xy}(a_{\psi(y)\psi(x)} - a_{\psi(x)\psi(y)}) + b_{yx}(a_{\psi(x)\psi(y)} - a_{\psi(y)\psi(x)}) \\ &\quad + \sum_{i=1,i\neq x,y}^{n} [b_{ix}(a_{\psi(i)\psi(y)} - a_{\psi(i)\psi(x)}) \\ &\quad + b_{iy}\cdot(a_{\psi(i)\psi(x)} - a_{\psi(i)\psi(y)}) \\ &\quad + b_{xi}\cdot(a_{\psi(y)\psi(i)} - a_{\psi(x)\psi(i)}) + b_{yi}\cdot(a_{\psi(x)\psi(i)} - a_{\psi(y)\psi(i)})]\end{aligned} \tag{3}$$

where a_{ij} is the flow between activities i and j and b_{ru} is the distance between activities r and u. If the outcome is negative, the swap has led to an improved permutation. This operation requires only $O(n)$ operations instead of the $O(n^2)$ for calling any of the presented fitness functions. A further improvement is the possibility of using dynamic programming. First, one step is computed in $O(n)$ using the above equation. Subsequent steps can then be executed based on prior results in a constant time complexity $O(c)$ using the following equation [94]. Let ψ be the placement caused by an exchange of x and y in placement ϕ. Exchanging units u and v (with $\{u,v\}\cap\{x,y\}=0$) can then be determined as follows if one stores the value of(ϕ,x,y)

$$\begin{aligned}\Delta(\psi,u,v) &= \Delta(\phi,x,y) + (b_{xu} - b_{xv} + b_{yv} - b_{yu}) \\ &\quad \cdot(a_{\phi(y)\phi(u)} - a_{\phi(y)\phi(v)} + a_{\phi(x)\phi(v)} - a_{\phi(x)\phi(u)}) \\ &\quad + (b_{ux} - b_{vx} + b_{vy} - b_{uy}) \\ &\quad \cdot(a_{\phi(u)\phi(y)} - a_{\phi(v)\phi(y)} + a_{\phi(v)\phi(x)} - a_{\phi(u)\phi(x)})\end{aligned} \tag{4}$$

Hence, a complete investigation of the neighborhood[20]. with size $(n^2-n)/2$ for each chromosome requires a first iteration of complexity $O(n^3)$[21] using Eq. (3) and succeeding iterations of $O(n^2)$[22] using Eq. (4). Moreover, these operations can be implemented simultaneously since the local search of one chromosome has no influence on the results of the local search of another. According to Merz [83], a local search is performed after the initialization phase and after every crossover and mutation, respectively.

4 Process Optimization Using a Competent GA: The Ordering Messy GA (OmeGA)

To cope with the difficulties facing SGAs—namely, the dramatic increase of the mixing boundary leading to exponential increases of population size necessary to reach certain accuracy—in this section we present an investigation of a class of GAs called competent GAs. Competent GAs solve hard problems of bounded difficulty quickly, accurately, and reliably [58]. Instead of exhibiting the exponential scale-up behavior of function calls and population size with problem size, the objective of designing competent GAs was to obtain a polynomial scale-up behavior (subquadratic desired). Usually, competent GAs are presented as selecto-recombinative GAs. Nevertheless, mutation can be still applied to them.

A special type of a component GA developed particularly for combinatorial optimization problems is the Ordering messy GA (OmeGA). Essentially, OmeGA is the combination of the fast messy GA [53] with *random keys* to represent permutations. Empirical tests by Knjazew [95] on relative ordering problems show a promising, subquadratic scale-up behavior of function calls $\approx O(l^{1.4})$. A competent GA or OmeGA has not previously been applied or tested for sequencing DSMs.

4.1 Data Structure. To permit genes to move around the genotype, messy GAs use a different technique than SGAs to represent a chromosome. Each gene is tagged with its location via the pair representation <locus, allele>. For example, the two messy chromosomes shown in Fig. 8 both represent the permutation (12-5-8-13).

OmeGA uses a binary coding representation, where messy chromosomes are encoded through random keys. This encoding has the advantage that any crossover operator can be used, since random keys always produce feasible solutions for combinatorial problems. Typically, floating point numbers are used as ascending sorting keys to encode a permutation. These numbers are initially determined randomly and alter only under the influence of mutation and crossover. Accordingly, a permutation of length l consists of a vector $r=(r_1,r_2,\ldots,r_l)$ with $r\in[0,1]^l$. A permutation sequence is determined by sorting the genes in ascending order of their associated random keys, as demonstrated by the example in Fig. 9.

Messy chromosomes may also have a variable length through being over- or underspecified. Consider the chromosomes in Fig. 10. As the problem length is 4 in this example, the first chromosome is *overspecified* since it contains an extra gene. The second chromosome is *underspecified* since it contains only one gene. To handle overspecification, Goldberg [53] proposes a gene expres-

[20]Our definition of the term neighborhood is based on the 2-opt heuristic. Using another definition for this term, e.g., a 3-opt neighborhood, would yield a different neighborhood size and computational complexity.

[21]This complexity results from the linear complexity of Eq. (3), which has to be executed $(n^2-n)/2$ times.

[22]This complexity results from the constant complexity of Eq. (4), which has to be executed $(n^2-n)/2$ times.

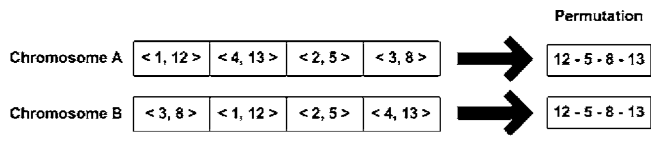


Fig. 8 Illustration of a messy chromosome

sion operator that employs a first-come-first-served rule on a left-to-right scan. In the example of Fig. 10, the gene assigned to locus 2 occurs twice in chromosome A. Thus, the left-to-right-scan drops the second instance, obtaining the valid permutation (12-5-8-13) instead of (12-11-8-13). In the case of underspecification, the unspecified genes are filled in by using a *competitive template*, which is a fully specified chromosome from which any missing genes are directly inherited. At the start of OmeGA, the genes of the competitive template are randomly chosen in consideration of feasibility issues. For example, using the competitive template shown in Figure 11, the underspecified chromosome B is completed by inheriting genes 1 through 3 from the competitive template.

4.2 Mechanics. Figure 12 shows the flow of OmeGA (and messy GAs in general). It consists of *epochs*, each of which contains two main loops: an inner and an outer. Each iteration of the outer loop corresponds to one building block (BB) level and is called an *era*[23]. Unlike in SGAs, the population is replaced after each era, not after each generation within an era. (A generation in the SGA corresponds to an era in OmeGA.) An inner loop runs for several generations (as they are defined for the OmeGA) within each era. The inner loop can be divided into three phases: (1) initialization phase, (2) BB filtering / primordial phase, and (3) juxtapositional phase. To overcome local optima, the OmeGA uses a technique called *level-wise processing*. After each era (i.e., each level), the best solution found so far is stored and used as the competitive template for the next era. The OmeGA processes until any convergence criterion is fulfilled. In general, any criterion proposed for the SGA can be adopted by the OmeGA.

4.2.1 Initialization Phase. The original messy GA [96] claimed that a population size with a single copy of all substrings of order/length k (ensuring the presence of all BBs) is necessary. In this case one can expect that the good BBs grow and are re-

[23]A BB "level k" denotes the processing of BBs of maximum size k within era k.

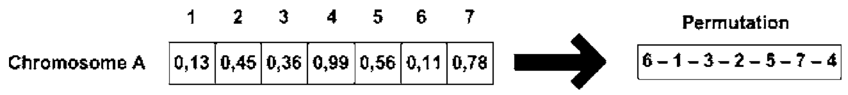


Fig. 9 Demonstration of random keys

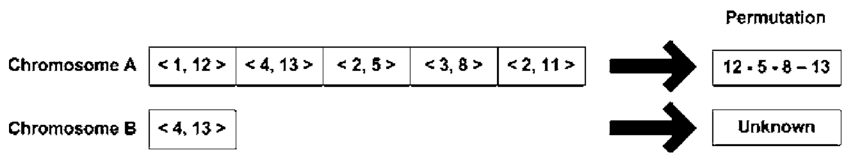


Fig. 10 Messy chromosomes may have variable length

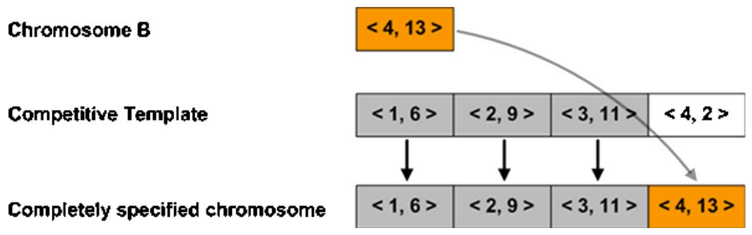


Fig. 11 Use of a competitive template on underspecified chromosomes

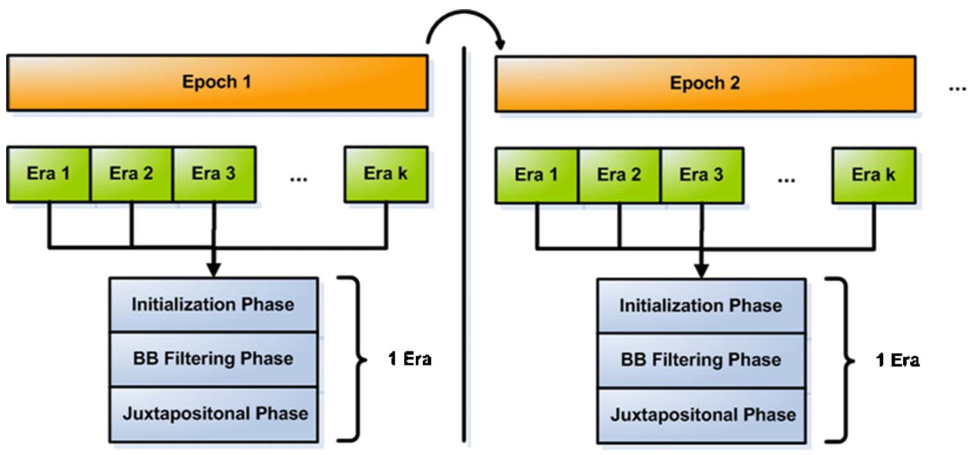


Fig. 12 OmeGA flowchart

combined in a way to create almost global optima. But the population size needed to guarantee such a copy is extremely high

$$n_{\text{pop}} = 2^k \binom{l}{k} \tag{5}$$

Equation (5) stems from the number of ways to choose order-k BBs from a string of length l, multiplied by the number of different BB combinations (assuming a binary alphabet), which is 2^k. Frequent initialization of such a large population causes a problem called *initialization bottleneck*. The fast messy GA "fixes" this problem as it was verified that a probabilistic complete initialization is sufficient for success [53]. For this purpose, Goldberg [53] developed an initial formula for BB supply (although population size is often determined empirically in practice). The length of the chromosomes can be chosen arbitrarily between k and l. Note that for each BB level (i.e., for each era) a completely new population is initialized. The size of the population s_t at level (or era) t, assuming a starting level 1, is given by: $\Sigma_{i=1}^{t} s_i$. Thus, the overall population size is the sum of population sizes for each era.

4.2.2 Building Block Filtering Phase. In order to identify highly fit BBs and to eliminate genes not belonging to BBs, the OmeGA uses repeated selection and random deletion of genes. At the beginning of the BB filtering phase, chromosomes arrive from the initialization phase with a length almost equal to the problem length. After this phase, chromosomes will be reduced to a length between 1 and the length of the current BB level in the epoch. For instance, if the first era of an epoch has a population size of 500, the second era has 750, and the third era has 1000, then the initialization phase of era 3 delivers 2250 chromosomes. During the BB filtering phase, 500 chromosomes are cut to a length of 1, 750 to a length of 2, and 1000 to a length of 3. At the beginning of the next epoch, the BB length is reset to 1.

To ensure an adequate supply of BBs, repeated selection is performed. Afterwards, genes are randomly deleted. The reason for a random deletion is the assumption that selection provides enough good genes so that a random deletion of some BBs is acceptable. In this way, genes not belonging to BBs can be filtered out. Figure 13 illustrates the BB filtering phase where the BBs are lightly shaded. The complete mathematical formulas of the BB filtering schedule, including equations for the generations necessary for the deletion and selection phase, can be found in [95,97]. In general, any selection scheme can be used, but typically TWOR with adequate tournament size is used due to the reasons mentioned in Sec. 3.2.1.

In addition to a selection operator, the original OmeGA paper [95] described the use of a *thresholding operator*. Thresholding works like niching to maintain diversity in the population and overcome the problem of cross-competition. That is, two chromosomes must have a certain number of genes in common before they can be compared during tournament selection. Traditional niching techniques, which work sufficiently fast, could be used as well. However, both approaches have the same intention. Thus, it seems fair to compare test results between the original OmeGA and a SGA extended with a niching operator.

4.2.3 Juxtapositional Phase. To recombine the identified BBs, messy GAs incorporate a juxtapositional phase. Instead of ordinary crossover operators, messy GAs use cut and splice operators [97], as demonstrated in Fig. 14. The *cut operator* breaks a chromosome into two parts with cut probability $p_{\text{cut}}=p_\kappa(l-1)$, where l is the current length of a chromosome and p_κ is a specified bitwise cut probability. The effects of the cut operator become more evident in later Juxtapositional Phases when the chromosomes become longer and therefore have a higher probability of getting cut. Knjazew [95] suggests keeping the length of a chromosome less than or equal to $2n$, gaining a cut probability set to the reciprocal of half of the problem length: $p_\kappa=2/n$. In contrast to the cut operator, the *splice operator* connects two chromosomes with

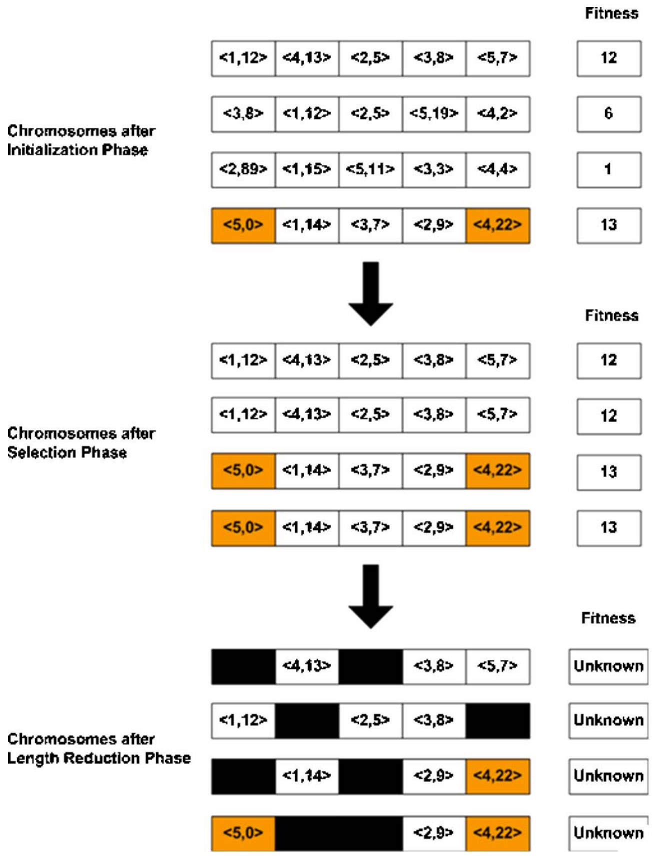


Fig. 13 Illustration of the BB filtering phase

probability p_s, usually chosen rather high. OmeGA thereby combines the identified BBs to find the optimum in exactly the same manner suggested by the BB Hypothesis. Note that the population size for each Juxtapositional Phase is held constant. At the end of this phase, the fitness of all chromosomes is evaluated.

4.3 Efficiency Enhancement Techniques for a Competent GA. In general, all of the proposed enhancement techniques presented in Sec. 3.3 can also be applied to the OmeGA. However, none of these techniques was applied and investigated in the original studies by Knjazew [95], although he did note the pivotal role of efficiency enhancement techniques. Therefore, we extended the OmeGA with respect to hybridization, using the 2-opt heuristic as a local searcher. In SGAs, local search is applied after the initialization phase, crossover, and mutation. However, in the OmeGA, no ordinary recombination operator exists after which a local search could be performed. Instead, cut and splice operators work continuously for some generations within an era. Performing a local search after an initialization of a population must be considered with care as the population is initialized several times in contrast to the SGA. Thus, available computational resources determine if such an implementation is reasonable.

Imitating our design of a simple global-local GA in Sec. 3, we apply the 2-opt local search to the entire population at the end of each era. In this way, the best chromosome after local search will be assigned as the competitive template. The OmeGA calls the fitness function not only at the end of each era but also during the BB filtering phase and the juxtapositional phase. Thus, local search is just applied after many more function calls compared to the hybrid SGA, implying less impact than a local search applied in the SGA. Therefore, it is important to study the implementation of other local search techniques in the OmeGA, such as incorporating the local search into the fitness function.

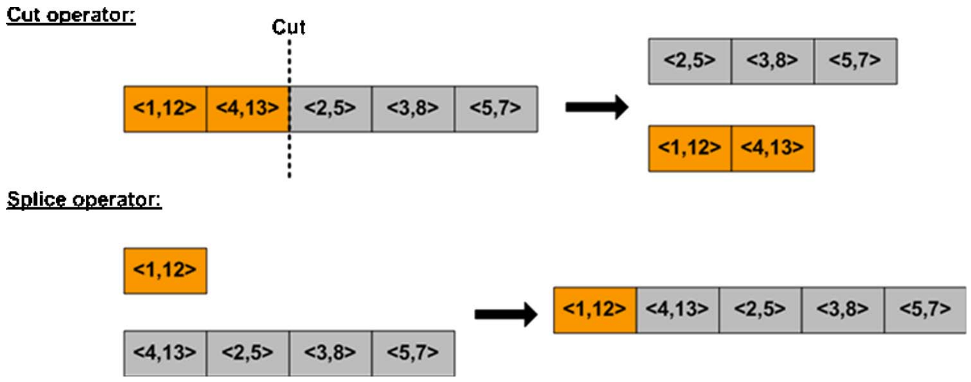


Fig. 14 Demonstration of the cut and splice operations

5 Run Duration Theory and Adjustment of Population Size

In this section, we briefly review two of the most important parameters to set for a GA: the population size and the number of generations to run. Multiple population size models assuming selectorecombinative GAs exist [61,70,98–100]. Harik [100] introduced a model, derived from the gambler's ruin problem, which is often used in the GA literature. Here, a decision between the right and wrong BB is incorporated over time (rather than solely in the first generation). The population size is calculated in a manner that the selection error is not more than a value α

$$n_{\text{pop}} = -\ln(\alpha)2^{k-1}\frac{\sigma}{2d}\sqrt{\pi \cdot m} \tag{6}$$

where n_{pop} is the population size, m is the number of BBs, d is the signal difference between the best and the second-best BB, σ^2 is the average fitness variance, and k is the order of the BBs. Although this provides a bound for an adequate population size, it requires problem-specific knowledge. In real-world problems, such parameters as the size and number of BBs are usually not available. Besides, the BB size could be nonuniform; i.e., BBs may have different lengths, or they could overlap. Therefore, we determined the population size for all of our subsequent GA tests empirically. Nevertheless, these models illuminate an important characteristic: when problems become more difficult, population size must be increased exponentially for high accuracy.

In contrast, convergence time models [59,71] predict an elongation of convergence time with an increase in problem size. These models are based on the evolution of average fitness of the population. Therefore, they use the *selection intensity* I instead of selection pressure. Mühlenbein [61] defined I as the expected average fitness of a population after performing selection. Values of I for different selection schemes can be found in the studies of Goldberg [58] and Miller [71], wherein the authors distinguished ordinal-based and proportionate selection schemes for their analysis of the run duration necessary for a successful GA convergence. As a result, the convergence time with ordinal-based schemes (e.g., Tournament Selection) is much faster than with fitness-proportionate schemes. According to [58], the convergence time t_c for a successful GA on a problem of length n can be estimated as

$$t_c = \frac{n\sqrt{3}\ln 2}{I} \tag{7}$$

6 Deceptive Functions

Artificial *deceptive functions* are often used to investigate the behavior of a GA in difficult search spaces. A deceptive function has a global optimum surrounded by misleading suboptimal attractors and far removed from many local optima. As opposed to the DSM-related fitness functions, problem difficulty can be scaled. In addition, the optimal solution is known in advance. Typically these deceptive functions are constructed by combining several deceptive subfunctions. The length of the subfunctions defines the *order* (or length) of a deceptive function.

The order of a deceptive function can also be understood as the size of the BBs for the combinatorial problem. In a DSM sequencing context, these BBs can be regarded as small subpermutations contributing much to the best overall process architecture. For both relative and absolute ordering problems, two deceptive functions of order 4 were introduced in [101]. The most recently constructed deceptive subfunction of order 6 is described in [95]. However, for the tests described later in this paper, we used the order-4 deceptive function. By combining k subfunctions of order m into a permutation of length l, only one global optimum exists, but $l!/(k!)^m$ local optima exist [95]. Thus, the creation of such deceptive functions results in an extremely difficult search space for the GA if the global optimum must be identified.

Using different problem codings provides a way to increase the difficulty of deceptive functions: the greater the defining length of the BBs, the greater the likelihood that operator disruption will occur, and thus the greater the problem difficulty. In the subsequent tests, we applied both tight and loose codings. Tight coding means that the genes belonging to one BB/subfunction are placed side-by-side in the chromosome. Consequently, tightly coded BBs have a defining length of $m-1$. In contrast, loose coding can be understood as a coding scheme whereby the defining length of BBs is maximal. The maximal defining length of k BBs for a problem of length l is given by $l-k$. Assuming four BBs of order 4, Table 3 illustrates both codings. Note that the optimal sequence (1-2-3-$\cdots$-16) is identical for both codings. Also note that messy codings provide high flexibility, as loosely coded BBs are treated in the same way as tightly coded ones.

7 Testing the SGA and the OmeGA

We now present test results for the SGA and the OmeGA, including their extensions through niching and hybridization. We performed tests for DSM blocks with different characteristics as well as for artificial deceptive functions. A main problem for the tests was the determination of potential sources of difficulty in a DSM block. We expected the block size to be a problem: the larger the block size, the greater the search space, and thus the

Table 3 Illustration of tight and loose codings

BB number	Tight coding	Loose coding
1	1-2-3-4	1-5-9-13
2	5-6-7-8	2-6-10-14
3	9-10-11-12	3-7-11-15
4	13-14-15-16	4-8-12-16

Table 4 Test parameters

SGA		OmeGA	
p_c:	1.0	p_{cut}:	$2/n$
p_m:	$1/4n$	p_s:	1.0
Selection scheme:	TWOR with/without continuously updated sharing[a] and tournament size 4	p_m:	$1/4n$
Crossover operator:	Position-based crossover, Version 2	Selection scheme:	TWOR with tournament size 4
Mutation operator:	Shift mutation	Number of eras per epoch:	4
		Ratio of population size in eras 1-4:	1:1:2:6

[a]This means that we made tests with tournament selection without a niching operator and some with niching operator.

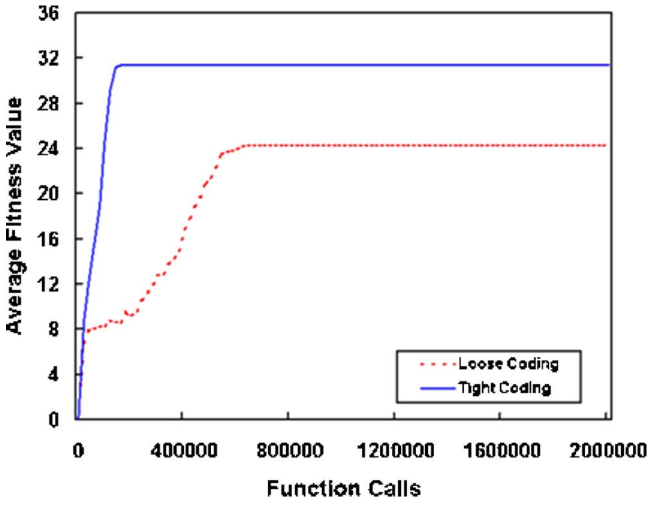
greater the difficulty. We also expected the density of a DSM block to play an important role [22]. The solution landscapes for certain sequencing objectives, in combination with block density (proposed in Sec. 2.3), show a quite symmetric landscape for the reduction of superdiagonal marks in sparse blocks. Hence, such a combination should be problematic for GAs, as symmetry is claimed to be one problem characteristic [102]. However, the extent to which linkages between elements in a DSM block influence the GA was not considered. We suggest that the difficulty in identifying BBs varies for differently linked DSM blocks of equal size. Thus, a small DSM block could be much harder to optimize than a large one. However, finding a systematic schema to scale the difficulty or to predict it in advance seems to be almost impossible. Each of the DSM blocks in the following tests was constructed with a global optimum known a priori.

While numerous tests could be accomplished to detect the best GA parameters, the number of possible test cases is quite large. Therefore, we set most of the GA parameters based on the discussions in the previous sections, and as shown in Table 4.

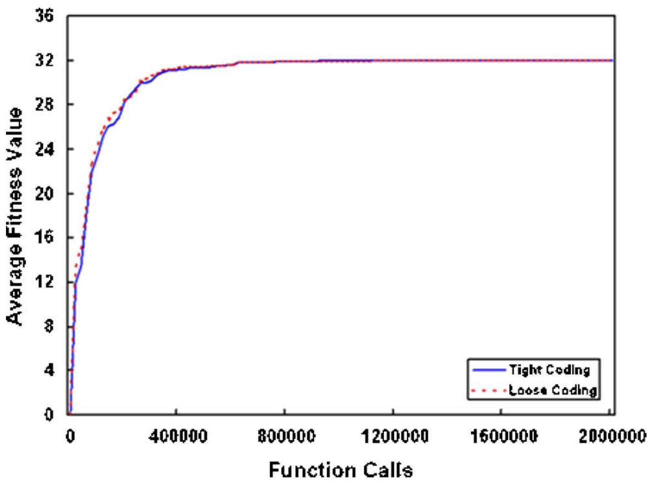
To empirically ascertain the desired population size for a specific test, we established that the optimal solution must be identified in 90% of all tests[24]. Empirically determining an adequate population size is an optimization problem of its own that can take a long time. Since time is often at a premium, a good population size for DSMs of arbitrary size can be predicted from the scale-up functions in the subsequent tests. In terms of run duration, we determined that some improvement (in the fitness of the best chromosome) had to occur within a certain window of generations. To calculate convergence time, and thus to estimate a good window of improvement, we used the convergence time model in [58], which includes Eq. (7). Assuming a maximal problem length of 100 and tournament selection of size 4 without replacement implied a prediction of 134 generations for successful convergence. Hence, a window of improvement of 100 generations seemed to be large enough. Since GA convergence time grows linearly with problem length [58], the problem lengths of our test cases equal the number of generations in each window; e.g., a problem of length 80 would have a window of 80 generations. Computational time and computational resources are affected mainly by the evaluation of the fitness function and population size. All other GA mechanics can be implemented in $O(n^2)$. Thus, we judge the scale-up of GAs and the quality of a GA design on the basis of the number of fitness function evaluations and the required population size. We intentionally did not perform tests with time as the unit of measurement, since time depends on individual computer configurations, the chosen programming language for the GA implementation, the size of the population, and other factors.

[24]We did not use different population sizes for different runs of a specific test scenario.

7.1 Tests on Deceptive Functions. To investigate the



(*a*)



(*b*)

Fig. 15 Test results for a tightly and loosely coded deceptive function of length 32. (*a*) SGA. (*b*) OmeGA.

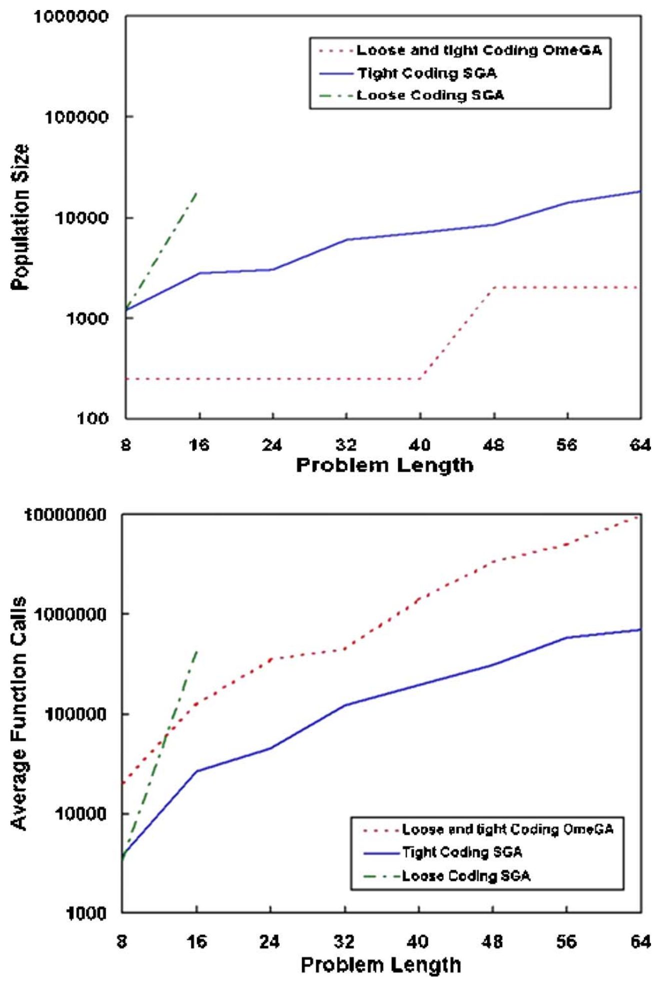


Fig. 16 Comparative performance of the OmeGA and the SGA on deceptive functions

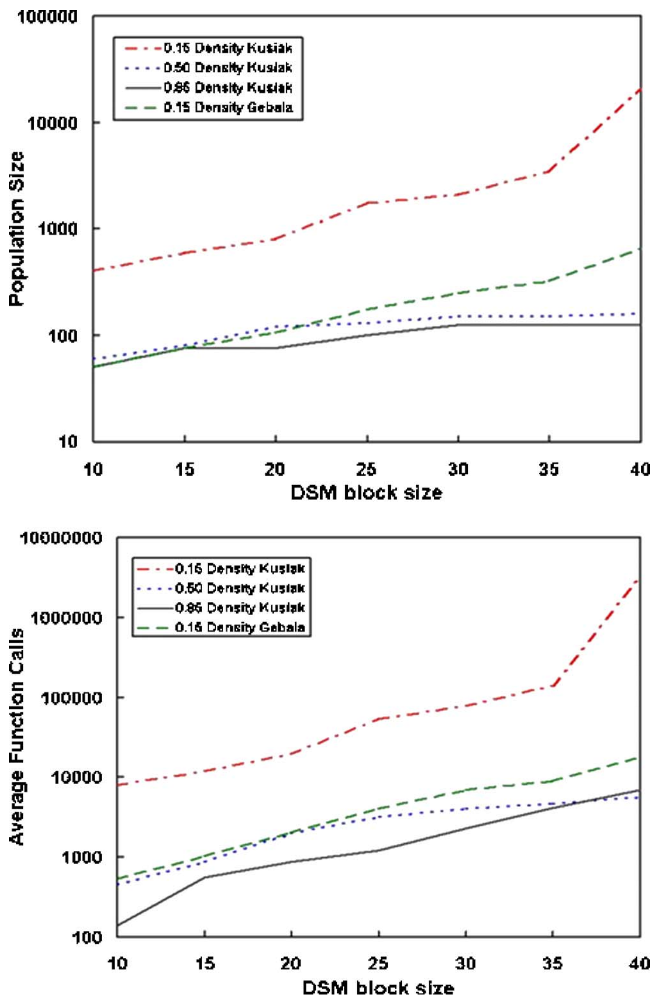


Fig. 17 Comparative SGA performance on DSM blocks with varied densities and objective functions

scale-up behavior of both GAs for hard problems, we first considered tests for deceptive functions with tight and loose coding. A SGA design is supposed to perform better for tightly coded functions, but this desired coding of BBs cannot be assumed in practice. For this reason it is important to investigate both coding extremes. In contrast, the OmeGA is expected to perform equally well for different codings of deceptive functions due to the advantages of messy codings. The tests of this section are based on the deceptive function of order 4, proposed in [101].

As a first test, we examined the course of the SGA and the OmeGA on both codings for a deceptive function of fixed length 32. The corresponding results, averaged over ten independent runs (an average of the best fitness values), are shown in Fig. 15. The SGA's performance differs significantly for different codings, whereas the OmeGA scales up equally regardless of BB coding, which is a significant advantage.

In tests with different lengths of the deceptive function, the OmeGA design scales up in both function calls and population size far better on loose coding than a SGA (Fig. 16). For tight coding, however, a SGA requires fewer function calls, although the population size is higher, because the OmeGA processes many more function calls per era than a SGA does in a generation to detect BBs. The charts point out that loose-coded problems require a hyper-exponential number of function calls and population size. In contrast, tightly coded BBs require a smaller population size while still exhibiting exponential scale-up behavior. Note that for the SGA design, only loose-coded deceptive functions of order 16 could be successfully tested, since the required population size to meet our accuracy criterion would have required a prohibitive amount of computational resources.

7.2 DSM-Related Tests. The test results related to artificial problems indicated that the OmeGA performs equally well on difficult and easy problems but was outperformed by the SGA on easy problems. Therefore, it is interesting to examine the performance of both approaches on practical DSMs, since the difficulty of a DSM sequencing problem is not known for real-world design problems. Thus, we prefer an optimization technique that performs almost equally well regardless of problem complexity. Since the reduction of feedback marks and their distance to the diagonal attracted the most interest in past DSM literature, we primarily tested the GAs with these fitness functions[25]. The DSM blocks (strongly connected components) for our tests were created in the following manner. First we created a SCC of size n by sequentially linking all activities; i.e., all cells (i,j) with $i=j+1$ yield a mark, followed by a mark in cell $(1,n)$. Depending on the underlying size and density of the tested DSM block, subdiagonal cells were randomly filled up with marks[26]. For a certain size and density only one specific DSM block was generated in the de-

[25]The mathematical equations of both objectives can be found in Table 1 (Gebala and Eppinger [40] and Kusiak [41]).

[26]Obviously, these relationships belong to the constructed SCC as well.

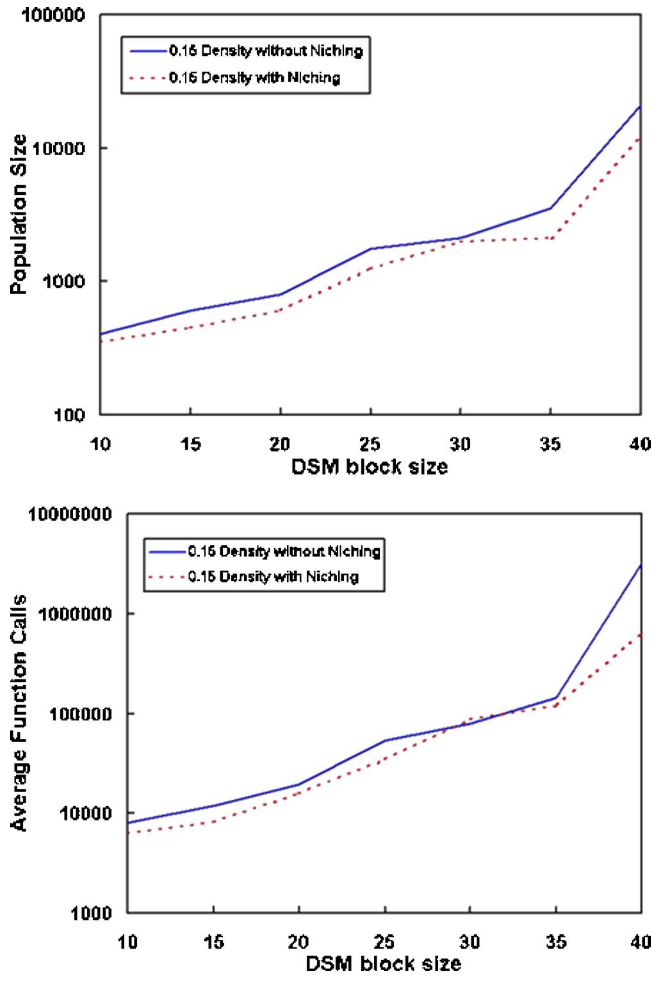


Fig. 18 Demonstration of niching to improve SGA performance

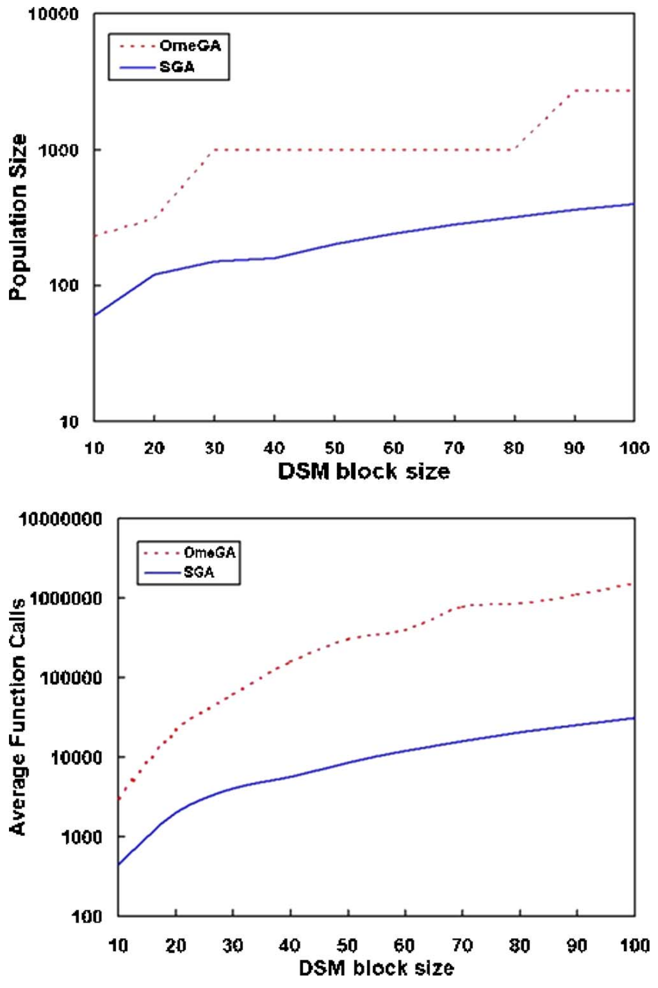


Fig. 19 Comparative performance of the OmeGA and the SGA with niching

scribed way.

7.2.1 Normal SGA and OmeGA Design. Our first test bank sought to investigate the scale-up behavior for DSM blocks using a SGA with varied densities (Fig. 17), since McCulley and Bloebaum [22] reported an exponential increase of population size required to reach certain accuracy as the density *decreases*, although they did not mention a reason for this behavior. With the plot of solution landscapes for different problem structures (Fig. 2) and the established problem characteristic of symmetry, a synchronization problem caused by symmetry seems to be a good explanation. Therefore, in addition to the Kusiak objective, we also tested a sparse DSM with the Gebala objective, since the solution landscape for the Gebala objective in Fig. 2 looks similar to the landscape for high-density blocks with the Kusiak objective. As a result, the performance of the SGA on a sparse DSM applying the Gebala objective differs substantially from the performance on the same DSM with the Kusiak objective. In general, it is very tough for a GA to identify a single global optimum in a symmetric landscape, since many non-inferior BBs exist that do not guide to the optimum.

Since reducing feedback marks is an important aspect of enhancing a process represented by a DSM, and sparse DSMs are common in the real world, one would like to obtain the same performance for this problem as for DSMs with high density or other sequencing objectives. van Hoyweghen [102] describes the symmetry problem and proposes several ways to ameliorate it. One way is to maintain diversity to ensure the growth of non-inferior BBs in the same proportion (niching). In fact, tests using continuously updated sharing [103] (Fig. 18) show a significant SGA performance enhancement (e.g., for a sparse 40×40 DSM block, the required population size was 12,000 chromosomes with niching, compared to 21,000 without). Nevertheless, niching cannot mitigate the exponential increase of population size and function calls.

One challenge in applying the OmeGA is the uncertainty about the BB structure: the BB size, the scaling of BB size, and the information about overlapped BBs are unknown a priori. Hence, the following tests assume a "conservative" problem difficulty with BB size 4; tests with more eras in each epoch might lead to better results. In the design of the OmeGA, the thresholding operator reduces the number of function calls in the same way niching does for a SGA. Therefore, we compare a SGA with niching to the OmeGA.

Tests for DSM blocks of density 0.5 (Fig. 19) indicate that the OmeGA requires many more function calls than the SGA. The scale-up of the OmeGA also seems to be slightly poorer. Although these facts seem to be disappointing, they are explained in that the OmeGA is at a disadvantage for easy problems[27], because its identification of BBs requires many more function calls. For easy problems, the explicit identification of BBs is not as crucial as in the case of difficult search spaces and can be regarded as a waste of resources. Invoking the fitness function is the main driver of computational duration, so one can expect linear growth in overall

[27]These DSM blocks can be regarded as "easy" since the SGA was able to identify the optimum with low population sizes.

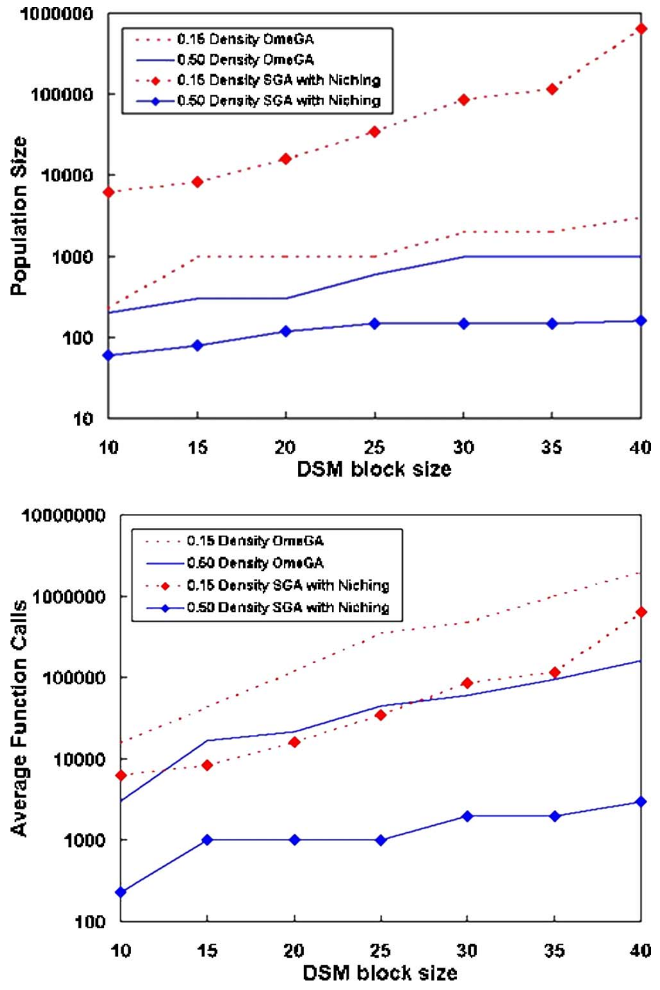


Fig. 20 Comparative performance of the OmeGA and SGA with niching as DSM block density varies

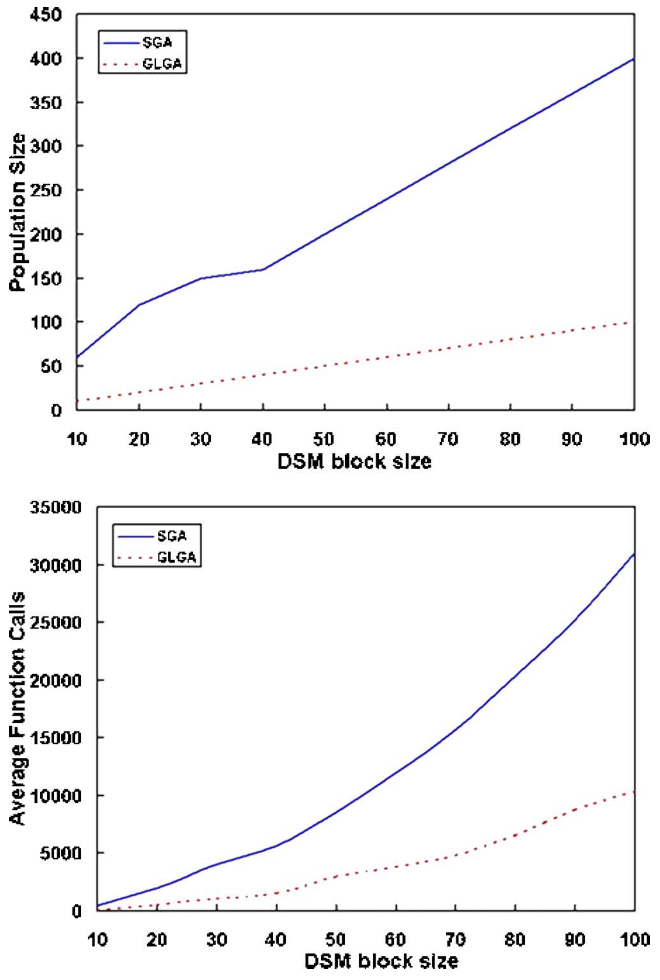


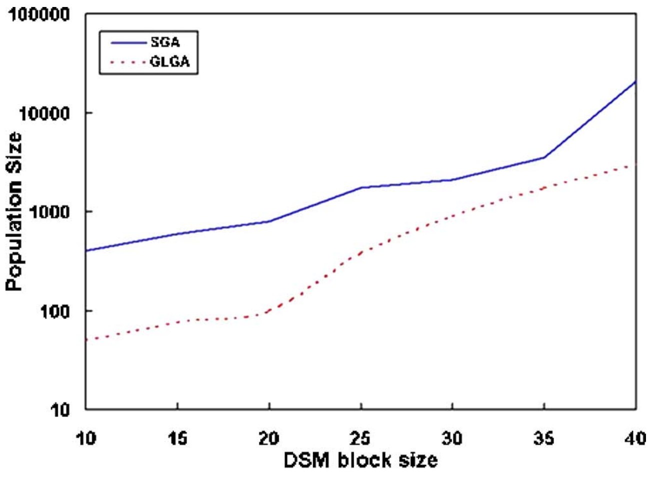
Fig. 21 Comparative performance of the regular SGA and the global-local SGA (block density=0.5)

GA time with linear growth in the number of fitness function evaluations. For instance, a GA that calls the fitness function 20 times is expected to take twice as long as one that performs ten evaluations of the same fitness function.

However, the test results for low-density DSMs ("hard" problems) in combination with a symmetric objective indicate that the OmeGA provides an advantage over the SGA. The curves of the OmeGA in Fig. 20 exhibit similar scale-up behaviors for both sparse and dense DSM blocks. In contrast, the scale-ups differ significantly for a SGA. Although the presence of many noninferior BBs in symmetry-related problems reduces the benefits of BB identification, the OmeGA has another advantage; the repeated initialization of the population. In SGAs, the correct BBs must be present in the initial population. Otherwise, only mutation is able to introduce new alleles and BBs. The complexity for a successful mutation is exponential: $O(m^k \log_{10} m)$ [61]. But multiple re-initializations provide many chances for the correct BBs to appear—effectively "restarting" the complexity for a successful competent GA convergence, given by $O(2^k l)$ if we combine Eqs. (6) and (7). However, this also implies a disadvantage leading to increased function calls: the search for BBs is started repeatedly. Accordingly, the absolute number of function calls for the OmeGA was higher for all tested DSM blocks. However, due to its slower scale-up behavior, the OmeGA is more economical for large DSM blocks. In particular, the smaller population size (associated with less computer memory) makes using the OmeGA more practical for large DSM blocks.

7.2.2 Hybrid SGA and OmeGA. Hybridization of the SGA and the OmeGA with a local searcher is expected to reduce the requisite population size and function calls for identical problems. Thus, the next tests address a "global-local GA" (GLGA) using the 2-opt heuristic as local search strategy. The test program was chosen identical to the tests on a normal SGA (in particular, we used identical DSM blocks for this test section as in case of the normal SGA). The first test dealt with DSM blocks having a density of 0.5 and a Kusiak objective. Although the SGA performed well, the GLGA demonstrated dramatically improved performance in terms of population size and function calls (Fig. 21), requiring only about a third of the resources. Even for a 100×100 DSM block, a population size of 100 was sufficient to detect the global optimum in about 100 generations. The scale-up behavior also improves, but remains slightly exponential. Unfortunately, the hybrid SGA cannot fix the symmetry problem in low-complexity DSM blocks. Figure 22 shows its dramatic increase in function calls and the continued, strongly exponential scale-up behavior.

Reducing the number of function calls must be the focus in the OmeGA instead of reducing the population size. Thus, we present only the test results for the hybrid OmeGA with respect to function calls. The impact of the implemented local search strategy cannot be expected to be as significant as with the SGA. Because $c_p = 1.0$, each chromosome in the SGA undergoes at least one local search after one evaluation of the fitness function. The local search for chromosomes in the OmeGA begins only after the execution of many more function calls at the end of an era, as



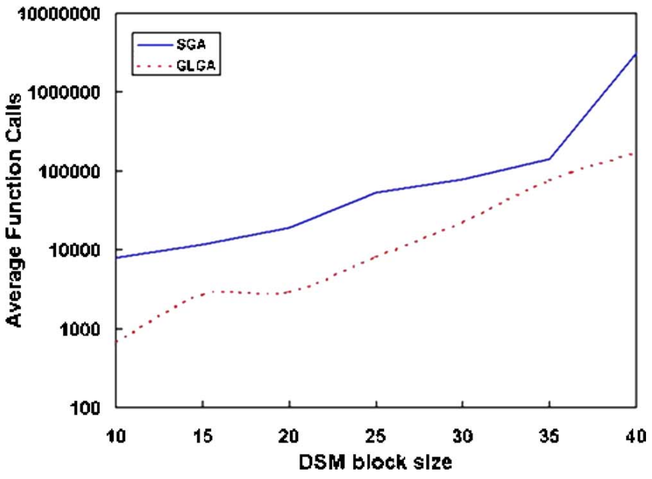
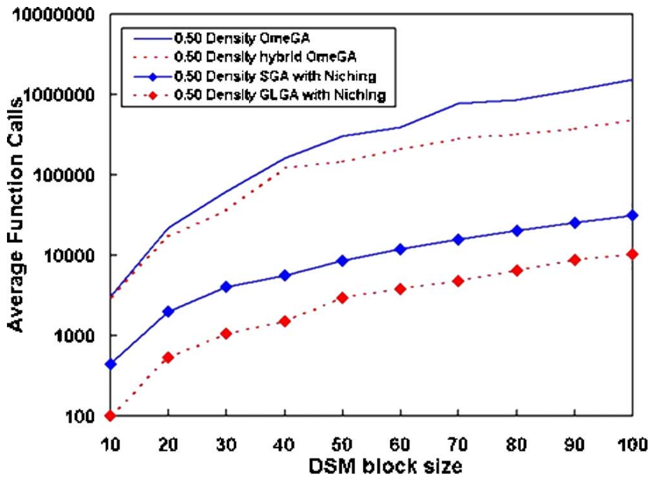


Fig. 22 Comparative performance of the regular SGA and the global-local SGA (block density=0.15)



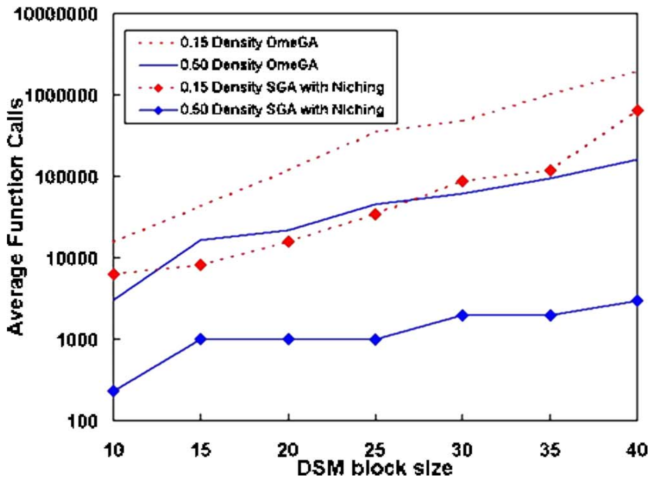


Fig. 23 Comparative performance of the regular SGA, the global-local SGA, the OmeGA, and the hybrid OmeGA on DSM blocks with varied densities

confirmed by corresponding test results. Indeed, the extended OmeGA achieved slightly fewer function calls on sparse and dense DSM blocks (Fig. 23). Nevertheless, the great discrepancy in the absolute number of function calls required for the hybrid SGA becomes obvious. Thus, we conclude that a local search should be used in the OmeGA. Further studies might consider the incorporation of a local searcher in the fitness function itself, which would decrease the discrepancy in performance between the hybrid SGA and the hybrid OmeGA.

8 Conclusions and Future Work

In this paper, we used GAs to sequence design processes. Such sequencing corresponds to an NP-hard QAP. We used the DSM merely as a representation scheme, and we focused on the individual sets of coupled or iterative blocks (i.e., strongly connected components) that may be extracted from a DSM. Through visualization of some DSM search landscapes, we demonstrated that the difficulty of sequencing DSMs differs depending on DSM characteristics such as size, density, and objective function.

We have demonstrated that, depending on the underlying fitness function and density of the DSM, a symmetry problem might occur during the sequence optimization, thus providing less guidance to the global optimum. Since linkages between DSM activities are responsible for the landscape of the search space, this issue significantly influences the degree of optimization difficulty. Tests confirmed this to some extent, as in some cases small DSMs required more function evaluations than large DSMs to obtain the same accuracy. Nevertheless, we could not find a general method to determine the difficulty of optimization a priori based on the linkages between the activities.

We conducted a number of tests on a SGA, a hybrid SGA, a competent GA, and a hybrid competent GA. A state-of-the-art competent GA (OmeGA) performs particularly well on very difficult problems wherein the search spaces were constructed artificially. The advantage of this competent GA is the almost identical performance independent of problem difficulty. Such an approach outperforms SGA designs for difficult problems, as a SGA can fix complexity and difficulty of search spaces only by an exponential increase of population size. Particularly for real-world problems with uncertain information about difficulty of the search space, the performance of the hybrid competent GA is beneficial. We have shown that the performance of SGAs degrades significantly on hard combinatorial problems. Therefore, we described, extended, and tested the OmeGA. Nevertheless, the explicit identification of BBs requires many function calls, and therefore hybrid SGA designs scale up better and outperform on easy problems.

Table 5 summarizes the results of our experiments with both GA designs. It is important to mention that this table does not present conclusive evidence about the choice of the best GA design and operators for certain DSM characteristics. Since all DSMs with certain characteristics (like density, sequencing objective, or size) cannot be tested and optimization difficulty depends on many factors, we cannot preclude, for instance, the superiority

Table 5 Comparison of DSM characteristics with GA settings according to test results

DSM density	Kusiak objective [41]	Gebala objective [40]
Sparse (<0.15)	Search Space: Symmetric GA type: Hybrid OmeGA Encoding: Binary Selection: TWOR, size 4 Crossover: Cut and splice operators Mutation: Bit flip	Search Space: Nonsymmetric GA type: Hybrid SGA Encoding: Integer Selection: TWOR, size 4 Crossover: Position-based, version 2 Mutation: Shift Mutation
Dense (>0.15)	Search Space: Nonsymmetric GA type: Hybrid SGA Encoding: Integer Selection: TWOR, size 4 Crossover: Position-based, version 2 Mutation: Shift	Search Space: Nonsymmetric GA type: Hybrid SGA Encoding: Integer Selection: TWOR, size 4 Crossover: Position-based, version 2 Mutation: Shift

of a competent GA design for some dense DSMs—although our tests indicate better scale-up behavior for a SGA on such DSMs. In fact, *the recommendation of our studies in applying GAs for design process optimization is the use of a competent GA extended by a local search strategy*. While a hybrid SGA might perform best at times, the difficulty of practical problems is typically unknown, and hence it is more rational to use a competent GA.

A great focus of future work should be on the classification of DSM difficulty prior to an optimization. This work has laid some of the groundwork by detecting a symmetry problem for certain DSM parameters and objective functions. Although it might be highly unlikely to forecast search space complexity depending on the linkages between activities, further studies might result in useful insights and observations. In addition, since local search strategies represent an important element for the solution quality of GAs, techniques other than the 2-opt heuristic should be examined. The performance of local searchers, depending on DSM parameters, is also interesting. For instance, robust taboo search [91] is likely to outperform the 2-opt heuristic for symmetric search spaces. Dorigo [38] illustrated with comparison of local search strategies for Ant Colony Systems that performance differences exist in the same problem domain. By deciding between different techniques, a good tradeoff must be found between the speed of local search and the quality obtained.

As the OmeGA is currently the only existing competent GA applied for combinatorial problems, other competent GAs, particularly the Bayesian Optimization Algorithm (BOA) [104], could also be examined with respect to their suitability for combinatorial problems. In general, the random key representation can also be utilized for the BOA, thereby overcoming the problem of constructing feasible solutions. A comparison of the OmeGA and any other competent GAs in terms of scale-up behavior for artificial functions and in terms of DSMs or QAP instances could also deliver interesting results.

Nomenclature

BB	=	building block
DSM	=	design structure matrix
QAP	=	quadratic assignment problem
SCC	=	strongly connected components of a graph
SP	=	selection pressure
a	=	number of arcs/edges/relationships in a digraph; number of off-diagonal marks in a DSM
c	=	constant factor
d	=	signal difference between the best and the second-best BB
I	=	selection intensity
k	=	order or size of BBs
l	=	current chromosome length
m	=	number of BBs
n	=	number of activities in a design process; number of vertices or nodes in a graph; DSM block size
n_{pop}	=	population size
p	=	probability to pick the best fit chromosome of a tournament during the tournament selection phase
p_c	=	crossover probability
p_{cut}	=	cut probability in the OmeGA
p_κ	=	bitwise cut probability in the OmeGA
p_m	=	mutation probability
p_s	=	slice probability in the OmeGA
s	=	size of tournament used in the Tournament Selection phase
t_c	=	convergence time
α	=	selection error
φ	=	number of SCCs in a graph or iterative/coupled blocks in a DSM
μ	=	calibration coefficient
σ^2	=	average fitness variance
π	=	Ludolph's number
ψ	=	DSM sequence
ϕ	=	DSM sequence

Acknowledgment

The authors would like to thank the *Heinrich und Lotte Mühlfenzel* foundation, which supported the first author. We would also like to thank Dr. Ken English (University at Buffalo, State University of NY) and Dr. Xavier Llora (Illini Genetic Algorithms Laboratory) for their insightful feedback and reviews on earlier drafts of this manuscript.

References

[1] Clark, K., and Fujimoto, T., 1991, *Product Development Performance*, Harvard Business School Press, Boston, MA.
[2] Yassine, A., Joglekar, N., Eppinger, S. D., and Whitney, D., 2001, "Performance of Coupled Product Development Activities With a Deadline," Manage. Sci., **47**(12), pp. 1605–1620.
[3] Suh, N., 1990, *The Principles of Design*, Oxford University Press, New York.
[4] Braha, D., and Maimon, O., 1998, *A Mathematical Theory of Design: Foundations, Algorithms and Applications*, Kluwer Academic Publishers, Dordrecht.
[5] Pahl, G., and Beitz, W., 1995, *Engineering Design: A Systematic Approach*, 2nd ed., Springer-Verlag, London.
[6] Ulman, D., 2003, *The Mechanical Design Process*, 3rd ed., McGraw-Hill, New York.
[7] Ulrich, K., and Eppinger, S. D., 2004, *Product Design and Development*, 3rd ed., McGraw-Hill, New York.
[8] Finger, S., and Dixon, J., 1989, "A Review of Research in Mechanical Design. Part I: Descriptive, Prescriptive, and Computer-Based Models of Design Processes," Res. Eng. Des., **1**(1), pp. 51–67.
[9] Eppinger, S. D., Whitney, D. E., Smith, R. P., and Gebala, D. A., 1994, "A Model-Based Method for Organizing Tasks in Product Development," Res. Eng. Des., **6**(1), pp. 1–13.
[10] Fricke, E., Gebhard, B., Negele, H., and Igenbergs, E., 2000, "Coping With

Changes: Causes, Findings, and Strategies," J. Syst. Eng., **3**(4), pp. 169–179.
[11] Safoutin, M. J., 2003, "A Methodology for Empirical Measurement of Iteration in Engineering Design Processes," Doctoral thesis (Mech. E.), University of Washington, Seattle, WA.
[12] Cooper, K. G., 1993, "The Rework Cycle: Part 1: Why Projects are Mismanaged," IEEE Eng. Manage. Rev., **21**(3), pp. 4–12.
[13] Browning, T. R., and Eppinger, S. D., 2002, "Modeling Impacts of Process Architecture on Cost and Schedule Risk in Product Development," IEEE Trans. Eng. Manage., **49**(4), pp. 428–442.
[14] Osborne, S. M., 1993, "Product Development Cycle Time Characterization through Modeling of Process Iteration," Master's thesis (Mgmt), MIT, Cambridge, MA.
[15] Smith, R. P., and Eppinger, S. D., 1997, "Identifying Controlling Features of Engineering Design Iteration," Manage. Sci., **43**(3), pp. 276–293.
[16] Safoutin, M., and Smith, R., 1998, "Classification of Iteration in Engineering Design Processes," *Proceedings of ASME Design Engineering Technical Conference (DETC98)*, Atlanta, GA, Sept. 13–16.
[17] Whitney, D. E., 1990, "Designing the Design Process," Res. Eng. Des., **2**(1), pp. 3–14.
[18] Yassine, A., and Braha, D., 2003, "Four Complex Problems in Concurrent Engineering and the Design Structure Matrix Method," Concurr. Eng. Res. Appl., **11**(3), pp. 165–176.
[19] Yassine, A., Whitney, D., Lavine, J., and Zambito, T., 2000. "Do-It-Right-First-Time (DRFT) Approach to Design Structure Matrix Restructuring," *Proceedings of the 12th International Conference on Design Theory and Methodology*, Baltimore, MD, September 10–13.
[20] Yassine, A., Joglekar, N., Eppinger, S. D., and Whitney, D., 2003, "Information Hiding in Product Development: The Design Churn Effect," Res. Eng. Des., **14**(3), pp. 145–161.
[21] Kehat, E., and Shacham, M., 1973, "Chemical Process Simulation Programs-2," Process Technology International, **18**(3), pp. 115–118.
[22] McCulley, C., and Bloebaum, C. L., 1996, "A Genetic Tool for Optimal Design Sequencing in Complex Engineering Systems," Struct. Optim., **12**(2), pp. 186–201.
[23] Altus, S. S., Kroo, I. M., and Gage, P. J., 1996, "A Genetic Algorithm for Scheduling and Decomposition of Multidisciplinary Design Problems," ASME J. Mech. Des., **118**(4), pp. 486–489.
[24] Kusiak, A., and Wang, J., 1993, "Decomposition of the Design Process," ASME J. Mech. Des., **115**(12), pp. 687–695.
[25] Michelena, N., and Papalambros, P., 1995, "Optimal Model-Based Decomposition of Powertrain System Design," ASME J. Mech. Des., **117**(4), pp. 499–505.
[26] Chen, L., Ding, Z., and Li, S., 2005, "A Formal Two-Phase Method for Decomposition of Complex Design problems," ASME J. Mech. Des., **127**(3), pp. 184–195.
[27] Wang, B., and Antonsson, E., 2004, "Information Measure for Modularity in Engineering Design," *Proceedings of the ASME 2004 Design Engineering Technical Conferences*, Salt lake City, Utah, Sept. 28–Oct. 2.
[28] Whitfield, R., Smith, J., and Duffy, A., 2002, "Identifying Component Modules," *Seventh International Conference on Artificial Intelligence in Design*, Cambridge, UK, July 15–17.
[29] Sosa, M., Eppinger, S. D., and Rowles, C., 2003, "Identifying Modular and Integrative Systems and Their Impact on Design Team Interactions," ASME J. Mech. Des., **125**(6), pp. 240–252.
[30] Warfield, J. N., 1973, "Binary Matrices in System Modeling," IEEE Trans. Syst. Man Cybern., **3**(5), pp. 441–449.
[31] Steward, D. V., 1981, *Systems Analysis and Management: Structure, Strategy and Design*, Petrocelli Books, New York.
[32] Tarjan, R., 1972, "Depth-First Search and Linear Graph Algorithms," SIAM J. Comput., **1**(2), pp. 146–160.
[33] Sedgewick, R., 1992, *Algorithms in C*, Addison-Wesley, Reading, MA.
[34] Holland, J. H., 1975, *Adaptation in Natural and Artificial Systems*, The University of Michigan Press, Ann Arbor, MI.
[35] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P., 1983, "Optimization by simulated annealing," Science, **220**(4598), pp. 671–680.
[36] Glover, F., 1989, "Tabu Search: Part 1," ORSA J. Comput., **1**(3), pp. 190–206.
[37] Glover, F., 1990, "Tabu search: Part 2," ORSA J. Comput., **2**(1), pp. 4–32.
[38] Dorigo, M., and Stützle, T. 2004, *Ant Colony Optimization*, MIT Press, Cambridge, MA.
[39] Browning, T. R., 2001, "Applying the Design Structure Matrix to System Decomposition and Integration Problems: A Review and New Directions," IEEE Trans. Eng. Manage., **48**(3), pp. 292–306.
[40] Gebala, D. A., and Eppinger, S. D., 1991, "Methods for Analyzing Design Procedures," *Proceedings of the ASME Third International. Conference On Design Theory and Methodology*, Miami, FL, September.
[41] Kusiak, A., and Wang, J., 1993, "Decomposition of the Design Process," ASME J. Mech. Des., **115**(4), pp. 687–695.
[42] Todd, D., 1997, Multiple Criteria Genetic Algorithms in Engineering Design and Operation, Ph.D. thesis, Engineering Design Centre, University of Newcastle upon Tyne, UK.
[43] Scott, J. A., 1998, "A strategy for Modeling the Design-Development Phase of a Product," in Department of Marine Engineering, University of Newcastle upon Tyne, Newcastle upon Tyne.
[44] Whitfield, R. I., Duffy, A. H. B., Coates, G., and Hills, W., 2003, "Efficient Process Optimization," Concurr. Eng. Res. Appl., **11**(12), pp. 83–92.
[45] Whitfield, R. I., Duffy, A. H. B., Gartzia-Etxabe, L. K., and Kortabarria, 2005, "Identifying and Evaluating Parallel Design Activities using the Design Structure Matrix," *International Conference on Engineering Design 2005*, Melbourne, August 15–18.
[46] Yu, T., Yassine, A., and Goldberg, D., "A Genetic Algorithm for Developing Modular Product Architectures," *Proceedings of the ASME 2003 International Design Engineering Technical Conferences*, Chicago, Sept. 2–6.
[47] Whitfield, R. I., Smith, J. S., and Duffy, A. H. B., 2002, "Identifying Component Modules," *Seventh International Conference on Artificial Intelligence in Design*, Cambridge, UK, July 15–17.
[48] Baldwin, C. Y., and Clark, K., 2000, *Design Rules: The Power of Modularity*. MIT Press, Cambridge.
[49] Rogers, J. L., 1996, "DeMAID/GA: An Enhanced Design Manager's Aid for Intelligent Decomposition," NASA TM-11024.
[50] Koopmans, T. C., and Beckmann, M. J., 1957, "Assignment Problems and the Location of Economic Activities," Econometrica, **25**, pp. 53–76.
[51] Sahni, S., and Gonzalez, T., 1976, "P-complete approximation problems," J. ACM, **23**(3), pp. 555–565.
[52] Sastry, K., and Goldberg, D. E., 2001, "Modeling Tournament Selection With Replacement Using Apparent Added Noise," Intelligent Engineering Systems Through Artificial Neural Networks, Vol. **11**, pp. 129–134.
[53] Goldberg, D. E., Deb, K., Kargupta, H., and Harik, G., 1993, "Rapid, Accurate Optimization of Difficult Problems Using Fast Messy Genetic Algorithms," *Proceedings of the Fifth International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA, pp. 56–64.
[54] Goldberg, D. E., 1989, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, New York.
[55] Whitley, D., and Yoo, N.-W., 1994, "Modeling Simple Genetic Algorithms for Permutation Problems," in *Foundations of Genetic Algorithms III*, Morgan Kaufmann, San Mateo, CA, pp. 163–184.
[56] Bean, J. C., 1994, "Genetic Algorithms and Random Keys for Sequencing and Optimization," ORSA J. Comput., **6**(2), pp. 154–160.
[57] Bäck, T., 1994, "Selective Pressure in Evolutionary Algorithms: A Characterization of Selection Mechanisms," *Proceedings of the First IEEE Conference on Evolutionary Computation*, Vol. 1, pp. 57–62.
[58] Goldberg, D. E., 2002, *The Design of Innovation*, Kluwer Academic Publishers Group, Norwell, MA.
[59] Baker, J. E., 1987, "Reducing bias and inefficiency in the selection algorithm," *Proceedings of the Second, International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA.
[60] Brindle, A., 1981, "Genetic Algorithms for Function Optimization," Doctoral dissertation, University of Alberta, Edmonton, Canada.
[61] Mühlenbein, H., 1992, "How Genetic Algorithms really work: Mutation and hillclimbing," in *Parallel Problem Solving from Nature II*, Springer-Verlag, Berlin, pp. 15–25.
[62] Goldberg, D. E., and Deb, K., 1991, "A Comparative Analysis of Selection Schemes Used in Genetic Algorithms," in *Foundations of Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA, pp. 69–93.
[63] Chambers, L., 2001, *The Practical Handbook of Genetic Algorithms: New Frontiers*. Chapman and Hall, Boca Raton, FL.
[64] Dukkipati, A., Murty, M. N., and Bhatnagar, S., 2004, "Cauchy Annealing Schedule: An Annealing Schedule for Boltzmann Selection Scheme in Evolutionary Algorithms," *Proceedings of the Congress on Evolutionary Computation (CEC'2004)*, Portland, OR, June 19–23, pp. 55–62.
[65] Mahfoud, S. W., and Goldberg, D. E., 1995, "Parallel Recombinative Simulated Annealing: A Genetic Algorithm," Parallel Comput., **21**(1), pp. 1–28.
[66] Murata, T., and Ishibuchi, H., 1994, "Performance Evaluation of Genetic Algorithms for Flow Shop Scheduling Problems," *Proceedings of the First IEEE Conference on Genetic Algorithms and their Applications*, Orlando, FL, June 27–29, pp. 812–817.
[67] Goldberg, D. E., Deb, K., and Thierens, D., 1993, "Toward a Better Understanding of Mixing in Genetic Algorithms," Journal of the Society of Instrument and Control Engineers, **32**(1), pp. 10–16.
[68] Bäck, T., 1995, "Generalized Convergence Models for Tournament- and (μ-l)-Selection," *Proceedings of the Sixth International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA, pp. 2–8.
[69] Thierens, D., 1995, "Mixing in Genetic Algorithms," Doctoral dissertation, Katholieke Universiteit Leuven, Belgium.
[70] De Jong, K. A., 1975, An Analysis of the Behavior of a Class of Genetic Adaptive Systems, Doctoral dissertation, University of Michigan.
[71] Miller, B. L., and Goldberg, D. E., 1995, "Genetic Algorithms, Tournament Selection, and the Varying Effects of Noise," Evol. Comput., **4**(2), pp. 113–131.
[72] Goldberg, D. E., and Segrest, P., 1987, "Finite markov chain analysis of genetic algorithms," *Proceedings of the Second International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA, pp. 1–8.
[73] Grefenstette, J. J., 1986, "Optimization of Control Parameters for Genetic Algorithms," Int. J. Numer. Methods Fluids, **16**(1), pp. 122–128.
[74] Schaffer, J. D., Caruana, R. A., Eshelman, L. J., and Das, R., 1989, "A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization," *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA, pp. 51–60.
[75] Bremermann, H. J., 1962, "Optimization Through Evolution and Recombination," in *Self-Organizing Systems*, Spartan Books, Washington, D.C.
[76] Syswerda, G., 1991, "Schedule Optimization Using Genetic Algorithms," in *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York.
[77] Cantu-Paz, E., 2000, *Efficient and Accurate Parallel Genetic Algorithms*, Kluwer Academic, Boston.
[78] Gorges-Schleuter, M., 1989, "Asparagos: An Asynchronous Parallel Genetic

Optimization Strategy," *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA.
[79] Grefenstette, J. J., 1981, "Parallel Adaptive Algorithms for Function Optimization," (Tech. Rep. No. CS-81-19), Vanderbilt University, Computer Science Department, Nashville, TN.
[80] Tanese, R., 1989, "Distributed Genetic Algorithms for Function Optimization," Doctoral dissertation, University of Michigan.
[81] Fleurant, C., and Ferland, J. A., 1994, "Genetic Hybrids for the Quadratic Assignment Problem," in *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, Vol. 16, pp. 173–187.
[82] Hinton, G. E., and Nowlan, S. J., 1987, "How Learning Can Guide Evolution," Complex Syst., **1**(3), pp. 495–502.
[83] Merz, P., and Freisleben, B., 1997, "A Genetic Local Search Approach to the Quadratic Assignment Problem," *Proceedings of the Seventh International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA, pp. 465–472.
[84] Goldberg, D. E., 1999, "Using Time Efficiently: Genetic-Evolutionary Algorithms and the Continuation Problem," IlliGAL Report No. 99002, University of Illinois at Urbana-Champaign, Il.
[85] Srivastava, R., and Goldberg, D. E., 2001, "Verification of the Theory of Genetic and Evolutionary Continuation," *Proceedings of the Genetic and Evolutionary Computation Conference*, San Francisco, CA, July 7–11, pp. 623–630.
[86] El-Beltagy, M., Nair, P., and Keane, A., 1999, "Metamodeling Techniques for Evolutionary Optimization of Computationally Expensive Problems: Promises and Limitations," *Proceedings of the Genetic and Evolutionary Computation Conference*, Orlando, FL, July 13–17, pp. 196–203.
[87] Jin, Y., Olhofer, M., and Sendhoff, B., 2000, "On Evolutionary Optimization With Approximate Fitness Functions," *Proceedings of the Genetic and Evolutionary Computation Conference*, Las Vegas, NV, July 10–12, pp. 786–792.
[88] Ratle, A., 1998, "Accelerating the Convergence of Evolutionary Algorithms by Fitness Landscape Approximation," in *Parallel Problem Solving from Nature V*, Springer-Verlag, Berlin, pp. 87–96.
[89] Sastry, K., Goldberg, D. E., and Pelikan, M., 2001, "Don't Evaluate, Inherit," *Proceedings of the Genetic and Evolutionary Computation Conference*, San Francisco, CA, July 7–11.
[90] Mahfoud, S. W., 1995, "Niching Methods for Genetic Algorithms," Doctoral dissertation, University of Illinois at Urbana - Champaign.
[91] Taillard, E. D., 1991, "Robust Taboo Search for the Quadratic Assignment Problem," Parallel Comput., **17**, pp. 443–455.
[92] Ahuja, R. K., Orlin, J. B., and Tivari, A., 1995, "A Greedy Genetic Algorithm for the Quadratic Assignment Problem," Working paper 3826-95, Sloan School of Management, MIT, Cambridge, MA.
[93] Burkard, R. E., and Rendl, F., 1984, "A Thermodynamically Motivated Simulation Procedure for Combinatorial Optimization Problems," Eur. J. Oper. Res., **17**(2), pp. 169–174.
[94] Stützle, T., and Dorigo, M., 2002, "Local Search and Metaheuristics for the Quadratic Assignment Problem," Unpublished paper.
[95] Knjazew, D., 2002, OmeGA: A Competent Genetic Algorithm for Solving Permutation and Scheduling Problems. Kluwer Academic Publishers Group, Norwell, MA.
[96] Goldberg, D. E., Korb, B., and Deb, K., 1989, "Messy Genetic Algorithms: Motivation, Analysis, and First Results," Complex Syst., **3**(5), pp. 493–530.
[97] Goldberg, D. E., Deb, K., and Korb, B., 1990, "Messy Genetic Algorithms Revisited: Studies in Mixed Size and Scale," Complex Syst., **4**(4), pp. 415–444.
[98] Goldberg, D. E., and Rudnick, M., 1991, "Genetic Algorithms and the Variance of Fitness," Complex Syst., **5**(3), pp. 265–278.
[99] Goldberg, D. E., Sastry, K., and Latoza, T., 2001, "On the supply of building blocks," Evol. Comput. Proceedings of the Genetic and Evolutionary Computation Conference, San Francisco, CA, July 7–11.
[100] Harik, G., Cantu-Paz, E., Goldberg, D. E., and Miller, B. L., 1999, "The Gambler's Ruin Problem, Genetic Algorithms, and the Sizing of Populations," Evol. Comput., **7**(3), pp. 231–253.
[101] Kargrupta, H., Deb, K., and Goldberg, D. E., 1992, "Ordering genetic algorithms and deception," in *Parallel Problems Solving from Nature II*, Springer-Verlag, Berlin, pp. 47–56.
[102] van Hoyweghen, C., Naudts, B., and Goldberg, D. E., 2002, "Spin-Flip Symmetry and Synchronization," Evol. Comput., **10**(4), pp. 317–344.
[103] Oei, C. K., Goldberg, D. E., and Chang, D. J., 1994, "Tournament Selection, Niching, and the Preservation of Diversity," IlliGAL Report No. 91011, University of Illinois at Urbana-Champaign, Il.
[104] Pelikan, M., Goldberg, D. E., and Cantu-Paz, E., 1999, "BOA: The Bayesian Optimization Algorithm," *Proceedings of the Genetic and Evolutionary Computation Conference*, Orlando, FL, July 13–17, pp. 525–532.