Pergamon

0305-0548(93)E0023-M

Computers Ops Res. Vol. 22, No. 1, pp. 111–134, 1995

0305-0548/95 $7.00+0.00

GENETIC ALGORITHMS AND TABU SEARCH: HYBRIDS FOR OPTIMIZATION

FRED GLOVER,† JAMES P. KELLY‡ and MANUEL LAGUNA§
Graduate School of Business, CB 419, University of Colorado, Boulder, CO 80309, U.S.A.

Scope and Purpose—The development of hybrid procedures for optimization focuses on enhancing the strengths and compensating for the weaknesses of two or more complementary approaches. The goal is to intelligently combine the key elements of competing methodologies to create a superior solution procedure. Our paper explores the marriage between tabu search and genetic algorithms in the context of solving difficult optimization problems. Among other ideas, the procedure known as scatter search is revisited to create a unifying environment where tabu search and genetic algorithms can co-exist. Overall, our objective is to demonstrate that it is possible to establish useful connections between methods whose search principles may superficially appear unrelated.

Abstract—Genetic algorithms and tabu search have a number of significant differences. They also have some common bonds, often unrecognized. We explore the nature of the connections between the methods, and show that a variety of opportunities exist for creating hybrid approaches to take advantage of their complementary features. Tabu search has pioneered the systematic exploration of memory functions in search processes, while genetic algorithms have pioneered the implementation of methods that exploit the idea of combining solutions. There is also another approach, related to both of these, that is frequently overlooked. The procedure called *scatter search*, whose origins overlap with those of tabu search (and roughly coincide with the emergence of genetic algorithms) also proposes mechanisms for combining solutions, with useful features that offer a bridge between tabu search and genetic algorithms. Recent generalizations of scatter search concepts, embodied in notions of *structured combinations* and *path relinking*, have produced effective strategies that provide a further basis for integrating GA and TS approaches. A prominent TS component called *strategic oscillation* is susceptible to exploitation by GA processes as a means of creating useful degrees of diversity and of allowing effective transitions between feasible and infeasible regions. The independent success of genetic algorithms and tabu search in a variety of applications suggests that each has features that are valuable for solving complex problems. The thesis of this paper is that the study of methods that may be created from their union can provide useful benefits in diverse settings.

1. INTRODUCTION

Genetic algorithms (GAs) and tabu search (TS) methods are customarily viewed as based on different foundations and perspectives. Although significant differences between the methods exist, the two approaches share certain elements in common that often have been overlooked and left unexploited. The theme of this paper is to identify connections and contrasts between the methods that offer a fertile means for creating hybrid procedures.

---

†Fred Glover is the US West Chaired Professor in Systems Science at the University of Colorado, Boulder. He has authored or co-authored more than two hundred published articles in the fields of mathematical optimization, computer science and artificial intelligence, with particular emphasis on practical applications in industry and government. In addition to holding editorial posts for journals in the U.S. and abroad. He is co-founder of Optimization Technologies, Inc., Analysis and Research and Computation, Inc., and the nonprofit research organization Decision Analysis and Research Institute.

‡James P. Kelly is an Assistant Professor of Management Science in the College of Business and Administration and Graduate School of Business Administration at the University of Colorado in Boulder. He received his doctoral degree in Applied Mathematics and Operations Research from the University of Maryland in 1990. His current research interests are in the area of heuristic combinatorial optimization. Dr Kelly has published several papers on topics such as tabu search and simulated annealing in various journals such as *Operations Research* and the *ORSA Journal on Computing*. Currently, he is attempting to use tabu search to construct neural networks for pattern classification.

§Manuel Laguna is an Assistant Professor of Operations Management in the College of Business and Administration and Graduate School of Business Administration of the University of Colorado at Boulder. He received master's and doctoral degrees in Operations Research and Industrial Engineering from the University of Texas at Austin. He was the first U S WEST postdoctoral fellow in the Graduate School of Business at the University of Colorado. He has done extensive research in the interface between artificial intelligence and operations research to develop solution methods for problems in the areas of production scheduling, telecommunications, and facility layout. Dr Laguna co-edited the Tabu Search volume of *Annals of Operations Research*.

The development of GA/TS hybrids offers an almost untapped area for empirical research. The numerous successful applications of both GA and TS approaches strongly argue for the merit of investigating unified procedures. We suggest ways that the differing frameworks of genetic algorithms and tabu search can be advantageously reconciled, inviting research and computational studies to identify the most effective ways to integrate these methods.

The plan of this paper is as follows. We begin by sketching basic components of tabu search in Section 2, and give an overview of basic genetic algorithm ideas in Section 3. Our goal in these sections is to provide a simplified characterization of the two methods, giving a sufficient background to demonstrate some of their key features and to lay a foundation for subsequent discussions. Section 4 then describes *scatter search* ideas that establish a link between early TS and early GA ideas. In Section 5 we identify recent generalizations embodied in the notion of *structured combinations*, which enable combined solutions to take account of problem structure and satisfy properties such as feasibility. Section 6 examines the strategic oscillation component of tabu search and notes its relevance for incorporation within GA processes. Section 7 gives additional strategies for integrating solutions derived from scatter search, called *path relinking*, and Section 8 introduces *attribute creation* approaches, both with significant applications for integrating GA and TS approaches. Finally, Section 9 elaborates on the strategies that differentiate and link tabu search and genetic algorithms, and suggets additional means for their unification.

2. TABU SEARCH IN OVERVIEW

Tabu search is based on introducing flexible memory structures in conjunction with strategic restrictions and aspiration levels as a means for exploiting search spaces. Its modern form and terminology are introduced in Glover [1,2], and also owe a debt to related ideas introduced by Hansen [3]. Many researchers have made contributions to advance the field in the past several years. Table 1 identifies some of these contributions and indicates specific applications that have benefited from them.

In diverse settings, such as those represented in the table, tabu search has demonstrated the ability to generate solutions of notably high quality. For problems that are small enough or tractable enough to allow "finitely convergent' algorithms to obtain and verify optimal solutions, tabu search also typically produces solutions that are optimal or within a fraction of a percent of optimality, while requiring much less effort (in some cases, on the order of minutes versus days of computer time).

For larger and more difficult problems, of the type customarily encountered in practical settings, tabu search obtains solutions that rival and often surpass the best solutions previously found by other approaches. The applications of Table 1 include comparative studies of a number of important classes of problems where tabu search performs significantly more effectively than all other methods tested. For example, in some of the more difficult applications in scheduling, quadratic assignment and vehicle routing, tabu search has set new records for best known solutions in the literature.

To provide a starting point for subsequent discussions, we sketch some of the basic ideas of tabu search in simplified form. The following example is abridged and adapted for our present purposes from Glover and Laguna [4].

An illustrative example

Permutation problems are an important class of combinatorial optimization problems whose applications include classical traveling salesman problems, quadratic assignment problems, production sequencing problems, and a variety of design problems. As a basis for illustration, consider the problem of designing a material consisting of a number of modules to insulate against specified forms of radiation. The order in which these modules are arranged determines the overall insulating property of the resulting material, as shown in Fig. 1.

The problem is to find the ordering of modules that maximizes the overall insulating property of the composite material. Assume that seven modules are required, and that it is computationally expensive to evaluate the overall insulating property of a particular ordering. We want to find an optimal or near-optimal solution by examining only a small subset of the total possible permutations

Table 1. Some applications of tabu search [4]

Area	Brief description	Reference
Scheduling	Employee scheduling	Glover and McMillan [5]
	Flow shop scheduling	Widmer and Hertz [6]
		Taillard [7]
	Job shop scheduling with tooling constraints	Widmer [8]
	Convoy scheduling	Bovet *et al.* [9]
	Single machine scheduling	Laguna *et al.* [10]
	Just-in-time scheduling	Laguna and Gonzalez-Velarde [11]
	Multiple-machine weighted flow time problem	Barnes and Laguna [12]
	Flexible-resource job shop scheduling	Daniels and Mazzola [13]
	Job shop scheduling	Dell'Amico and Trubian [14]
	Single machine scheduling (target analysis)	Laguna and Glover [15]
	Resource scheduling	Mooney and Rardin [16]
	Sequencing jobs with deadlines and setup times	Woodruff and Spearman [17]
Transportation	Traveling salesman problem	Malek *et al.* [18]
		Glover [19]
	Vehicle routing problem	Gendreau *et al.* [20]
		Osman [21]
		Semet and Taillard [22]
Layout and Circuit Design	Quadratic assignment problem	Skorin-Kapov [23]
		Taillard [24]
		Chakrapani and Skorin-Kapov [25]
	Electronic circuit design	Bland and Dawson [26]
Telecommunications	Path assignment	Oliveira and Stroud [27]
		Anderson *et al.* [28]
	Bandwidth packing	Glover and Laguna [29]
Graphs	Clustering	Glover *et al.* [30]
		Hansen *et al.* [31]
		Dorndorf and Pesch [32]
	Graph coloring	Hertz and de Werra [33]
		Hertz *et al.* [34]
	Stable set in large graphs	Friden *et al.* [35]
	Maximum clique problem	Gendreau *et al.* [36]
Probabilistic Logic Expert Systems	Maximum satisfiability problem	Hansen and Jaumard [37]
	Probabilistic logic	Jaumard *et al.* [38]
	Probabilistic logic/expert systems	Hansen *et al.* [39]
Neural Networks	Learning in an associative memory	de Werra and Hertz [40]
	Nonconvex optimization problems	Beyer and Ogier [41]
Others	Multiconstraint 0–1 knapsack problem	Dammeyer and Voss [42]
	Large-scale controlled rounding	Kelly *et al.* [43]
	Generalized fixed charge problem	Sun and McKeown [44]

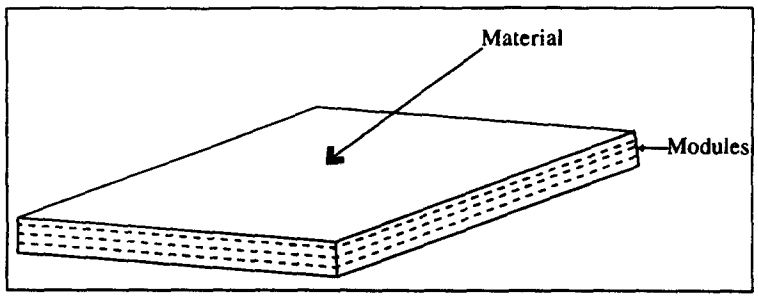


Fig. 1. Modules in an insulating material.

(which in many practical applications can be astronomical, and for this example equals 7!, i.e. 5040). Job sequencing, signal screening and chemical filtering problems also fit in this same mold.

To introduce and illustrate the basic components of tabu search, we first assume that an initial solution for this problem can be constructed in some intelligent fashion, i.e. by taking advantage of some problem-specific structure. Suppose the initial solution to our problem is the one shown in Fig. 2.

The ordering in Fig. 2 specifies that module 2 is placed in the first position, followed by module 5, etc. The resulting material has an insulating property of 10 units. (We assume this was found

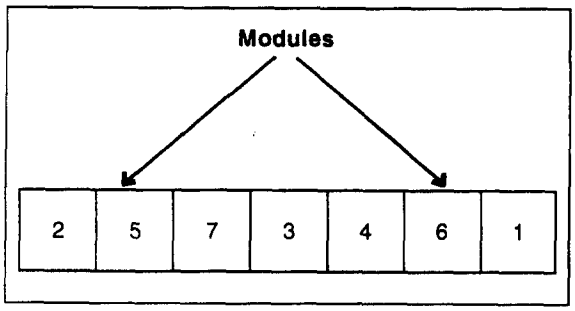


Fig. 2. Initial permutation.

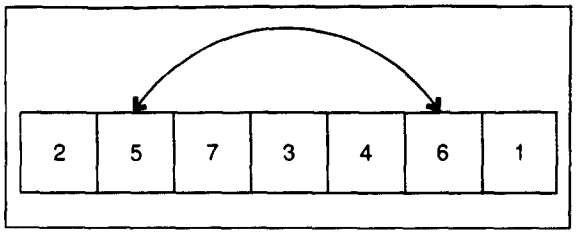


Fig. 3. Swap of modules 5 and 6.

by an accompanying evaluation routine, e.g. a simulator package for estimating the properties of a material without actually building a prototype.)

TS methods operate under the assumption that a neighborhood can be constructed to identify adjacent solutions that can be reached from any current solution. Pairwise exchanges (or swaps) are frequently used to define neighborhoods in permutation problems, identifying *moves* that lead from one solution to the next. In our problem, a swap exchanges the position of two modules as illustrated in Fig. 3. Therefore, disregarding other options such as insert moves, the complete neighborhood of a given current solution consists of the 21 adjacent solutions that can be obtained by such swaps.

Associated with each swap is a move value, which represents the change in the objective function value as a result of the proposed exchange. Move values generally provide a fundamental basis for evaluating the quality of a move, although other criteria can also be important. In large problems, it is critical to use candidate lists to screen moves for examination, both to reduce computational effort and to focus on more promising alternatives [45].

A chief way to exploit memory in tabu search is to classify a subset of the moves in a neighborhood as forbidden (or tabu). The classification depends on the history of the search, and particularly on the recency or frequency that certain move or solution components, called *attributes*, have participated in generating past solutions. For example, one attribute of a swap is the identify of the pair of elements that change positions (in this case, the two modules exchanged). To prevent the search from repeating swap combinations tried in the recent past, which could potentially reverse the effects of previous moves by interchanges that return to previous positions, we will classify as tabu all swaps composed of any of the most recent pairs of such modules; in this case, for illustrative purposes, the three most recent pairs. This means that a module pair will be kept tabu for a duration (tenure) of three iterations. A data structure such as the one shown in Fig. 4 may be used.

Each cell of the structure in Fig. 4 contains the number of iterations remaining until the corresponding modules are allowed to exchange positions again. Therefore, if the cell (3,5) has a value of zero, then modules 3 and 5 are free to exchange positions. On the other hand, if cell (2,4) has a value of 2, then modules 2 and 4 may not exchange positions for the next two iterations (i.e. a swap that exchanges these modules is classified tabu).

The move attributes illustrated are not the only ones possible for defining tabu restrictions. For example, we may refer to separate modules rather than module pairs, or to positions of modules, or to links between their immediate predecessors (or successors), and so forth. (For guidelines about good choices of attributes, see Glover and Laguna [29].)

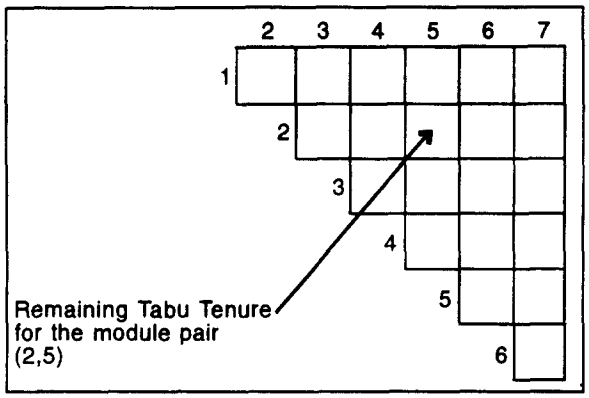
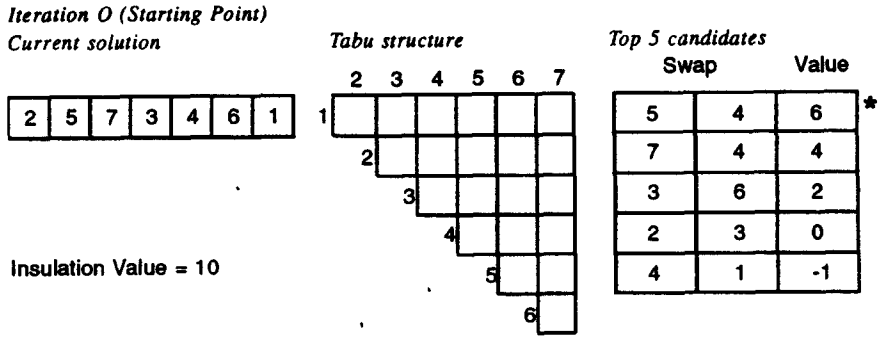


Fig. 4. Tabu data structure for attributes consisting of module pairs exchanged.



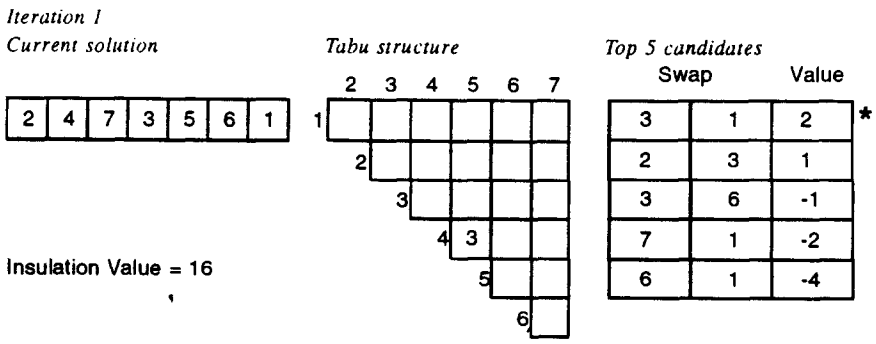
Scheme 1

Tabu restrictions are subject to an important exception. When a tabu move has a sufficiently attractive evaluation, as for example where it would result in a solution better than any visited so far, then its tabu classification may be overridden. A condition that allows such an override to occur is called an *aspiration criterion*. The following shows four iterations of the basic tabu procedure that employs the *paired module* tabu restriction and the best *solution* aspiration criterion.

The starting solution has an insulation value of 10, and the tabu data structure is initially empty (implicitly filled with zeros) indicating no moves are classified tabu at the beginning of the search. After evaluating the candidate swap moves, the top five moves (in terms of move values) are shown in the table for iteration 0 above. This information is provided by an independent evaluation subroutine designed to identify move values for this particular problem.

To locally maximize the insulating property of the material, we swap the positions of modules 5 and 4, as indicated by the asterisk. The total gain of such a move equals six units, producing the situation in the next table, labeled "Iteration 1."

The new current solution has an insulating value of 16 (i.e. the previous insulation value plus the value of the selected move). The tabu structure now shows that swapping the positions of



Scheme 2

Iteration 2

Current solution

2	4	7	1	5	6	3

Insulation Value = 18

Tabu structure

	2	3	4	5	6	7
1		3				
2						
3						
4	2					
5						
6						

Top 5 candidates

Swap		Value	
1	3	-2	T
2	4	-4	*
7	6	-6	
4	5	-7	T
5	3	-9	

Scheme 3

modules 4 and 5 is forbidden for three iterations. The most improving move at this step is to swap 3 and 1 for a gain of 2, giving the following outcome.

The new current solution becomes the best solution found so far with an insulating value of 18. At this iteration, two exchanges are classified tabu, as indicated by the nonzero entries in the tabu structure.

Note that entry (4,5) has been decreased from 3 to 2, indicating that its original tabu tenure of 3 now has two remaining iterations to go. This time, none of the candidates (including the top five shown) has a positive move value. Therefore, a nonimproving move has to be made. The most attractive nonimproving move is the reversal of the move performed in the previous iteration, but since it is classified tabu, this move is not selected. Instead, the swap of modules 2 and 4 is chosen, as indicated by the asterisk in the table for Iteration 2. This yields the following result.

Iteration 3

Current solution

4	2	7	1	5	6	3

Insulation Value = 14

Tabu structure

	2	3	4	5	6	7
1		2				
2			3			
3						
4	1					
5						
6						

Top 5 candidates

Swap		Value	
4	5	6	T*
5	3	2	
7	1	0	
1	3	-3	T
2	6	-6	

Scheme 4

The new current solution has an insulation value inferior to the two values previously obtained, as a result of executing a move with a negative move value. The tabu data structure now indicates that three moves are classified tabu, with different remaining tabu tenures. At the top of the candidate list, we find the swap of modules 4 and 5, which in effect represents the reversal of the first move performed, and is classified tabu. However, performing this move produces a solution with an objective function value that is superior to any previous insulation value. Therefore, we make use of the aspiration criterion to override the tabu classification of this move and select it as the best on this iteration.

Iteration 4

Current solution

5	2	7	1	4	6	3

Insulation Value = 20

Tabu structure

	2	3	4	5	6	7
1		1				
2			2			
3						
4	3					
5						
6						

Top 5 candidates

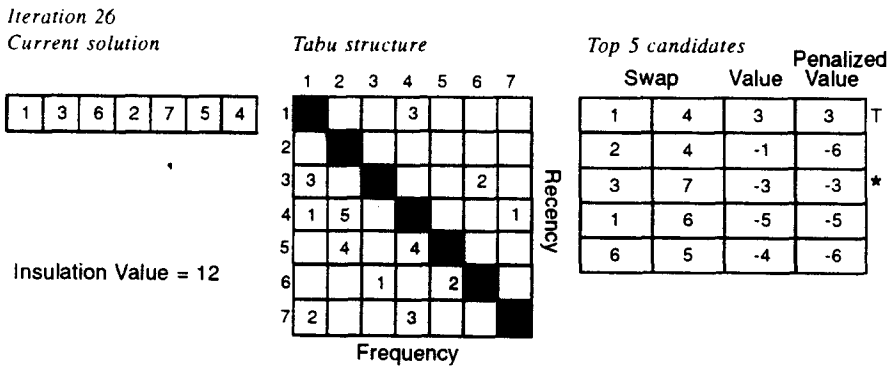
Swap		Value	
7	1	0	*
4	3	-3	
6	3	-5	
5	4	-6	T
2	6	-8	

Scheme 5

The current solution becomes the incumbent new best solution and the process continues. Note that the chosen tabu restriction and tabu tenure of three results in forbidding only three out of 21 possible swaps, since the module pair with a residual tenure of 1 always drops to a residual tenure of 0 each time a new pair with tenure 3 is introduced. (By recording the iteration when a module pair becomes tabu, and comparing this against the current iteration to determine the remaining tabu tenure, it is not necessary to change these entities at each step as we do here.)

In some situations, it may be desirable to increase the percentage of available moves that receive a tabu classification. This may be achieved either by increasing the tabu tenure or by changing the tabu restriction. For example, a tabu restriction that forbids swaps containing at least one member of a module pair will prevent a larger number of moves from being executed, even if the tenure remains the same. (In our case, this restriction would forbid 15 out of 21 swaps if the tabu tenure remains at 3.) Such a restriction is based on single module attributes instead of paired module attributes, and can be implemented with much less memory, i.e. by an array that records a tabu tenure for each module separately. Generally speaking, regardless of the type of restriction selected, improved outcomes are obtained by tabu tenures that vary dynamically.

Complementary tabu memory structures. The accompaniment of recency based memory with frequency based memory adds a component that operates over a longer horizon. To illustrate one of the useful longer term applications of frequency based memory, suppose that 25 TS iterations have been performed, and that the number of times each module pair has been exchanged is saved in an expanded tabu data structure. The lower diagonal of this structure now contains the frequency counts.



Scheme 6

At the current iteration (iteration 26), the recency memory indicates that the last three module pairs exchanged were (4,1), (6,3), and (7,4). The frequency counts show the distribution of moves throughout the first 25 iterations. We use these counts to diversify the search, driving it into new regions. This diversifying influence is restricted to operate only on particular occasions. In this case, we select those occasions where no admissible improving moves exist. Our use of the frequency information will penalize nonimproving moves by assigning a larger penalty to swaps of module pairs with greater frequency counts. (Typically these counts would be normalized, as by dividing by the total number of iterations or their maximum value.) We illustrate this in the present example by simply subtracting a frequency count from the associated move value.

The list of top candidates for iteration 26 shows that the most improving move is to swap (1,4), but since this module pair has a residual tabu tenure of 3, it is classified tabu. The move (2,4) has a value of -1, and it might otherwise be the one next preferred, except that its associated modules have been exchanged frequently during the history of the search (in fact, more frequently than any other module pair). Therefore, the move is heavily penalized and it loses its attractiveness. The swap of modules 3 and 7 thus is selected as the best move on the current iteration.

The strategy of instituting penalties only under particular conditions is used to preserve the aggressiveness of the search. Penalty functions in general are designed to account not only for frequencies but also for move values and certain influence measures.

In addition, frequencies defined over different subsets of past solutions, particularly subsets of elite solutions consisting of high quality local optima, give rise to strategies from the class of approaches that seek to exploit information from collections of good solutions. Such approaches are called *intensification* approaches. Complementary procedures to drive the search into new regions are called *diversification* approaches. Intensification and diversification strategies interact to provide fundamental cornerstones of longer term memory in tabu search.

In the following sections, we focus on the special subset of such strategies that consist of generating new solutions by reference to "combining" previous solutions, and therefore provide a link to genetic algorithm strategies. We acknowledge that in some respects the trend in this direction has already begun, as several of the basic notions of intensification and diversification introduced in tabu search are beginning to be examined in GA settings. However, we will show how it is possible to go considerably farther, making more powerful use of strategies that have been neglected or incompletely adapted to the GA setting, giving an extended bridge between TS and GA approaches.

3. GENETIC ALGORITHMS IN OVERVIEW

As described by Goldberg [46] and Davis [47], natural evolution has some general features that motivated John Holland in the 1970's to start a research effort in an area that would eventually become what is now known as genetic algorithms (GAs). These features are briefly itemized by Davis as follows:

- Evolution is a process that operates on chromosomes rather than on the living beings they encode.
- Processes of natural selection cause those chromosomes that encode successful structures to reproduce more often that those that do not.
- Mutations may cause the chromosomes of biological children to be different from those of their biological parents, and recombination processes may create quite different chromosomes in the children by combining material from the chromosomes of two parents.
- Biological evolution has no memory.

The guiding premise of GAs is that complex problems can be solved by simulating evolution via a computer algorithm. In Holland's conception, this occurs by algorithms that manipulate binary strings labeled chromosomes. As in biological evolution, the simulated evolution has the goal of finding good chromosomes by a blind manipulation of their contents. The term blind refers to the fact that the process does not have any information about the problem it is trying to solve. In the original GA conception, the process is viewed as a black box that provides evaluations of chromosomes. These evaluations are then used to bias the selection of chromosomes in a way that superior chromosomes (i.e. those with higher evaluations) will reproduce more often than inferior ones. Holland's early designs were simple, but were reported to be effective in solving a number of problems considered to be difficult at the time. The field of genetic algorithms has since evolved, chiefly as a result of innovations in the 1980's, to incorporate more elaborate designs aimed at solving problems in a wide range of practical settings.

Goldberg [46] suggests that GAs characteristically are distinguished in four ways:

- GAs work with a coding of the parameter set, not the parameters themselves.
- GAs search from a population of points, not a single point.
- GAs use payoff (objective function) information, not derivatives or other auxilary knowledge.
- GAs use probabilistic transition rules, not deterministic rules.

In the rest of this section, we discuss how these features have become incorporated in genetic algorithms at various levels, and we also review some of the more recent trends and developments in the field.

Holland's canonical GA is characterized by binary representatin of individual solutions, simple

```
Create and evaluate an initial population.
do {
    Reproduce new strings.
    Evaluate the fitness of the new offspring.
    Replace strings of the old population with the new offspring.
} while (termination criteria are not met)
```

Fig. 5. A simple GA.

problem independent crossover operators, and a proportional selection rule. To understand these concepts, consider the simple GA outlined in Fig. 5. The population members are *strings* or *chromosomes*, which as originally conceived are binary representations of solution vectors. GAs undertake to select subsets (usually pairs) of solutions from a population, called parents, to combine them to produce new solutions called children (or offspring). Rules of combination to yield children are based on the genetic notion of *crossover*, which consists of interchanging solution values of particular variables, together with occasional operations such as random value changes (called *mutations*). Children produced by the mating of parents, and that pass a survivability test, are then available to be chosen as parents of the next generation. The choice of parents to be matched in each generation is based on a biased random sampling scheme, which in some (nonstandard) cases is carried out in parallel over separate subpopulations whose best members are periodically exchanged or shared.

The purpose of parent selection in GAs is to increase the probability of reproducing members of the population that have higher evaluations. A popular way of implementing this process is called *roulette wheel parent selection*. This technique may be viewed as a roulette wheel where each member of the population is represented by a slice that is directly proportional to the member's fitness. A selection step is then a spin of the wheel, which in the long run tends to eliminate the least fit population members.

Crossover in nature is the phenomenon by which two parents exchange parts of their corresponding chromosomes to create children. In genetic algorithms a crossover recombines the genetic material encoded in two parent chromosomes to make two children. The original GA operator, modeled after processes in nature, is called *one-point crossover*. This operator randomly chooses a cutting point such that the genetic material beyond that point is exchanged between two parents to create two children. An example of one-point crossover is as follows:

Parent 1: 0 1 0 | 1 1 ⇒ Child 1: 0 1 0 0 1
Parent 2: 1 1 0 | 0 1 ⇒ Child 2: 1 1 0 1 1

Crossover is essential to genetic algorithms, and in fact many researchers feel that this component characterizes an algorithm as genetic. As the GA field has developed, the one-point crossover has been found to have restricted value, since occasions arise when it cannot combine certain features encoded on chromosomes. A popular solution to this problem has been the use of *two-point crossover*. In this case, two cutting points are selected at random and the genetic material that falls within the two points is exchanged. Although this operator is more flexible than the one-point crossover, it also is often unable to recombine parents to produce certain children that can carry important information to the next generation. This motivated Ackley [48] to develop the operator called *uniform crossover*. Starting with the first bit, a parent is selected at random to contribute its bit to the first child, while the second child is assigned the bit from the other parent. The process continues until all bits are examined.

Another important component of GAs is mutation, though it is usually conceived as a background operator. Bit mutation is applied to binary strings by randomly replacing each bit with certain probability. The probability value is known as the mutation rate, and many researchers advocate that it should generally be low (e.g. 0.01 or less). There are basically two ways of implementing mutation. The first variant always changes the bit for which the probability test has been passed. That is, if the ith bit is orignally 1 and the probability test is passed, the new string will contain a 0 in the ith position. In the second variant a new bit is randomly generated to substitute the bit for which the probability test passed. Therefore, 50% of the time the new bit will not be different than the old one, and the mutation rate will effectively be half of the one of the first variant.

The binary encoding was originally used by Holland to derive the *schema theorem*, which is considered the mathematical foundation of genetic algorithms. (More recent GAs are not limited to binary encoding, and in fact, as discussed later, other solution representations have proved somewhat more effective in applications where binary representations are not "natural" to the problem setting.) The schema theory is based on the idea that if a better understanding of the problem domain is desired, it is essential to study the similarities among subsets of strings as well as their fitness. In a sense the interest shifts from strings alone to groups of highly fit strings. As pointed out by Goldberg [46], the framework of schemata provides the tools for answering questions such as: how a string can be similar to its fellow strings? and in what ways is a string a representative of the other string classes with similarities at certain positions? A schema is defined as a template among strings. Over the binary alphabet, a schema is a string of the type:

$$(a_1, a_2, \ldots, a_n),\ a_i \in \{0, 1, \bigstar\}$$

where the "$\bigstar$" symbol represents both 0 and 1. A schema represents a sub-space of strings that match all locations i where the schema is specific. Therefore, the size of the sub-space increases with the number of "$\bigstar$" symbols in the schema. The schema theorem changes the view of the process from a search in a space of individual strings to a search through the set of schemata which the strings instantiate [49]. Since each binary string is an instantiation of 2^n possible schemata, testing the survivability of a string discloses a great deal of implicit information regarding the fitness of its corresponding schemata. This is one of the cornerstones of the idea called *implicit parallelism.*

At each step in the evolution process strings are selected from the population with a probability relative to their fitness. This selection has the effect of including in each generation representatives of a particular schemata proportionate to their average fitness. (However, the average fitness of a schemata only indicates which string templates are more promising to investigate.) Selection processes with additional controls have been made the basis for a GA convergence theory by Aarts *et al.* [50].

The schema theorem is sometimes interpreted to suggest that the implicit parallelism of GAs may be exponential. However, this is not quite correct, because certain schemata do not survive reproduction and the number of different schemata being considered depends on the population size. More precisely, if a single hyperplane is present in the initial population and is consistently above average fitness over time, then it will exponentially increase its representation over time. Holland estimated that the number of schemata being processed at each evolution generation in a population of n strings is $O(n^3)$. Such processing is often interpreted to indicate that GAs make a nearly optimal allocation of trials. Mühlenbein [51] has shown, however, that this is only true for simple optimization problems. DeJong [52] has additionally characterized related limitations when GAs take the role of function optimizers.

The binary encoding that is strongly coupled to the schemata idea has often been found unsuitable for combinatorial optimization problems. In this context, the difficulty stems from finding a binary representation such that substrings have a meaningful interpretation [53]. Therefore, there is a group of GA researchers who use encodings that are more natural representations of solutions [54–56]. Most of these encodings, however, do not support the idea of schemata. Some GA researchers feel procedures that violate some of the principles in the original theory work, "despite" their violations. In contrast, other members of the GA community suggest that the success of these procedures is precisely based on the fact that they violate the original GA principles. Regardless of how these encodings are perceived, their goal is to identify similarity features in the

solution space, while maintaining a natural connection with the solution vectors that originate them. In general, different encodings give different perspectives and resolution for inspecting a solution space.

Another relatively new trend in the GA arena is the use of local search strategies. After rather disappointing results were obtained in early applications of simple GAs to combinatorial optimization [57], researchers began to look for ways to extend GA approaches to produce better results. A leading step in this direction was taken by Mühlenbein *et al.* [58] with the development of Parallel Genetic Algorithms (PGAs) that allow individuals in the population to improve their fitness by local improvement ("hill climbing"). (PGAs also allow individuals to select their own mates by distributed processing.) Uses of local improvement operators are also introduced and successfully applied in the work of Montana and Davis [59], Huntley and Brown [60] and Ulder *et al.* [61] (who call their approach *genetic local search*).

In a sense, simple GAs with local search may be seen as sophisticated multi-start descent methods. The re-starting process is in this case governed by genetic rules, and the descent phase is performed as customary. The success of these methods may be attributed to their balance between achieving fast search and sustaining diversity to avoid premature convergence. Another approach to achieve a similar outcome within GAs is an iterative search strategy called *delta coding* [62]. Delta coding introduces diversity by generating an entirely new and random population, while preserving information from past generations by basing new encodings on previous partial solutions. The approach starts by performing an initial run of a simple GA and measuring the population diversity on each generation. When the diversity of the population (measured for example by calculating the hamming distance between parent strings) is exhausted or reduced, the best solution is saved as starting point for the next delta iteration. Delta coding preserves hyperplane sampling since each individual run is a single run of a genetic algorithm, where only the encoding strategy has changed. Since the number of bits used to encode delta values is reduced at each iteration, the solution space is reduced. Each new string, when decoded for fitness evaluation, is applied as a delta value to the string saved from previous iteration. This process translates into a search on a smaller solution space around the best initial solution.

The underlying principle of reinforcing the search in regions about high quality solutions, as noted earlier, is an instance of what is called an *intensification strategy* in TS. Intensification strategies in a TS context bear interesting connecting to their counterparts in GAs, illustrated by the *scatter search* method for generating new solutions in regions determined in relation to previous good solutions, and by the approach of reinforcing values applicable to *strongly determined* and *consistent* variables [63]. We explore such connections between TS and GA methods, and their implications for usefully blending these methods, in the following sections.

4. SCATTER SEARCH

Scatter search, which was introduced in roughly the same period as the early GA proposals, has some interesting commanalities with GA ideas, although it also has a number of quite distinct features. Several of these features, as we will show, have come to be incorporated into GA approaches after an intervening period of approximately a decade, while others remain largely unexplored in the GA context.

Scatter search is designed to operate on a set of points, called *reference points*, that constitute good solutions obtained from previous solution efforts. The approach systematically generates linear combinations of the reference points to create new points, each of which is mapped into an associated point that yields integer values for discrete variables. As originally proposed [63], the mapping consists of rounding or an associated generalized adjacency process, e.g. rounding a selected discrete variable to an integer neighbor, then determining implied value changes for other variables, and repeating.

This idea of using a systematic process to enable solution combinations to meet desired restrictions embodies the principle that such combinations should be influenced by context. (Genetic algorithm concepts proposed at this time, and advocated in some circles even today, instead embrace the theme that combinations should be generated without reference to context.) The adaptive rounding process of scatter search automatically satisfies simple constraints such as mutual exclusivity and

precedence relationships, since by standard updating, each rounding step yields only those remaining options consistent with choices made earlier. (More complex constraints sometimes also can be satisfied this way by updating a linear programming basis representation, as characteristically done in integer programming implementations.)

The vectors that result from the rounded linear combinations of the chosen reference points in turn are allowed to serve as inputs to accompanying heuristic processes. The heuristic procedures then transform these inputs into improved outcomes, thereby bringing the approach full circle. These outcomes accordingly are screened to provide a new set of reference points, and the process starts again.

By this approach, linear combinations produced at each stage are dispersed across a region whose form is biased by the distribution of reference points. Diagram 1 (from Glover [63]), illustrates a simple version of the process. Each of the points numbered 1–16 in Diagram 1 is the central point of an apparent subregion of the simplex A, B, C. The points A, B and C may or may not constitute the original reference points. (For example, the original points may consist of 6, 7 and 11, or of 4, 5, 12 and 13.) Thus, new points may be created that are not convex combinations of original points, and hence that may contain information that is not contained in these points, in the sense of bits implicit in a solution representations. (At the same time, the original points are also instances of such linear contributions, and hence they are likewise included among the candidate outcomes.)

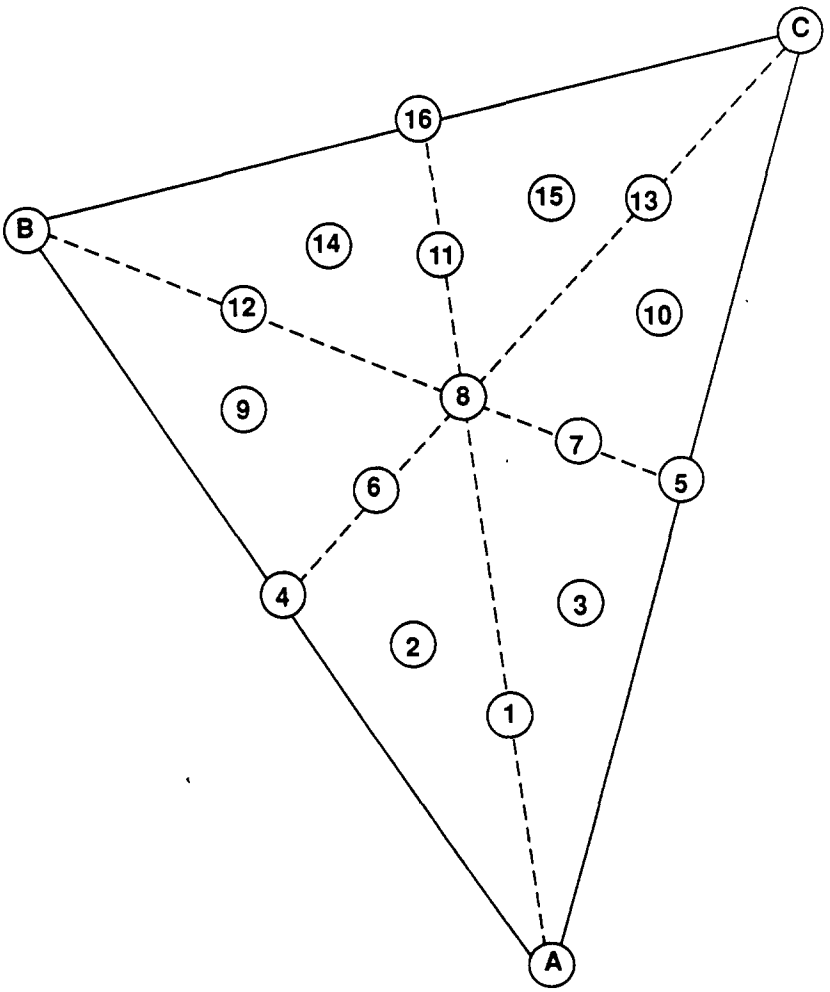


Diagram 1. An illustration of scatter search. Reproduced with permission from Glover (1977) *Decision Science* Vol. 8, pp. 156–166.

The mappings that progressively round the resulting linear combinations (modifying fractional components that are required to be discrete), can introduce additional information derived from relationships between problem variables, hence reflecting the influence of problem structures. Problem structure exerts further influence by means of the heuristic processes that take these points as inputs and produce new solutions from them.

Similarities are immediately evident between this early approach and the GA formulation of Holland [64]. Both are instances of what are sometimes called "population based" approaches, which start with some collection of elements and progressively evolve those elements to yield new ones that are subjected to the same guiding process. Both also incorporate the idea that a key aspect of producing the new elements is to generate some form of *combination* of the existing elements.

On the other hand, several contrasts between the methods also may be noted. The early GA approaches were predicated on the idea of choosing parents randomly to produce offspring, and further on introducing randomization to determine which components of the parents should be combined (by genetic crossover operations). By contrast, no corresponding recourse to randomization is made in the scatter search approach, although nothing excludes its use as a bias factor (i.e. probabilistically favoring evaluation criteria that would otherwise be applied deterministically). Attention is focused in scatter search on choosing good solutions as a basis for generating combinations, in contrast to the more democratic GA policy of allowing solutions of all types to be combined. This scatter search focus can enhance the generation of relevant outcomes without losing the ability to produce diverse solutions, due to the way the generation process is implemented. More recent "elitist" GA variants also give preference to combining good elements, but without a corresponding ability to produce combinations beyond the region in which these elements lie (except by resorting to auxiliary strategies to overcome this limitation). The different mechanisms used by the two approaches to combine solutions, consisting of rounded linear combinations on one hand and genetic crossover on the other, particularly invite examination.

Significance of rounded linear combinations

Linear combinations provide a somewhat more varied set of possibilities for creating new solutions than crossover as initially introduced in GAs. They also avoid the artificiality of resorting to the binary representations which were the foundation of the original genetic encoding and crossover motions. To see the relevance of this, consider the goal of creating integer solutions that are combinations of the two solutions $x=9$ and $x=26$. Rounded linear combinations can generate every integer point from minus to plus infinity on the line joining $x=9$ and $x=26$, hence in this case yielding every value x may feasibly be assigned. On the other hand, when these solutions are given a binary representation, (1 0 0 1 0) for $x=9$, and (0 1 0 1 1) for $x=26$ (encoding x as a bit string by the identity $x=1x_1+2x_2+4x_3+8x_4+16x_5$, where $x_1, x_2, \ldots, x_5$ are zero–one integer variables), then the possible outcomes are substantially more limited. In particular, the only ways to create rounded linear combinations of the binary vectors (1 0 0 1 0) and (0 1 0 1 1) yield the collection of binary vectors (★ ★ 0 1 ★) where the "★ elements" can be 0 or 1. Hence instead of producing all possible integer points these combinations produce only the integer values of x satisfying $8 \leqslant x \leqslant 11$ and $24 \leqslant x \leqslant 27$.

The possible outcomes are more limited still if classical forms of genetic crossover are used. The only vectors that can be created by the proposals of Holland are the four vectors (0 0 0 1 0), (0 1 0 1 0), (1 0 0 1 1), (1 1 0 1 1), corresponding to $x=8$, 10, 25, 27. When attention is restricted to binary vectors, which clearly is inappropriate, rounded linear combinations in fact give the same set of possibilities as the "uniform" crossover operator proposed 10 years later by Ackley [48], although the randomized means of generating these possibilities in the GA setting contrasts with the strategic rounding theme of scatter search. By further employing generalized adjacency rounding, where values of some variables may change as a result of modifying others, additional possibilities result.

The significance of rounding to account for interactions between variables is illustrated by the following example [65]. Consider the simple integer programming problem

Minimize

$$9x_1+4x_2+8x_3$$

subject to

$$9x_1 - 8x_2 - x_3 \geq 7$$
$$-6x_1 + 7x_2 - 2x_3 \geq 6$$
$$-x_1 - x_2 + 5x_3 \geq 9$$
$$x_1, x_2, x_3 \geq 0 \text{ and integer.}$$

The linear programming (LP) solution to this problem, which disregards the integer requirement for the variables, gives a solution vector $x=(x_1, x_2, x_3)=(24.43, 25.14, 11.71)$. Successive rounding that respects interactions between the variables (implied by the inequality constraints above, and manifested in the structure of the LP basis inverse), yields a solution vector $x=(29, 30, 14)$, which turns out to be optimal for this problem. Evidently, the integer values of this final vector could not be anticipated without accounting for the interdependencies among the problem variables. Advanced manifestations of such phenomena and processes for exploiting them are given in Glover [66, 67], and represent the types of processes that are accommodated naturally within the scatter search framework.

Without contradicting the importance of randomization in GA processes, the fact that scatter search seeks to create new points strategically rather than randomly may represent a useful feature in some settings. The points of Diagram 1, for example, may be generated and scanned in their indicated numerical order, under the condition where this order reflects a ranking determined by the objective function, or more generally by a feasible direction gradient. Scatter search does not prespecify the number of points it will generate or retain, since this can be established adaptively by considering the quality or structure of solutions produced in such a systematic generation.

Scatter search and early GA approaches may also be distinguished by the fact that the reference points are supplied by and in turn supply another heuristic process. This is an orientation that has lately gained strong proponents among a core of researchers in the area of optimization who are seeking to modify GA proposals to make them more effective, particularly as advocated in the PAG approach of Mühlenbein *et al.* [58] and the related genetic local search approach of Ulder *et al.* [61].

Finally, we observe that the allowance for real-valued weights and vector components (for variables or parameters that are not discrete) anticipates the developing trend in some parts of the GA community to embrance "real-coded" (or floating-point) genes, as represented by the work of Davis [47], Bäck *et al.* [68] and Eschelman and Schaffer [69]. Additional connections with current GA developments are provided in the study of Michalewicz *et al.* [70], which introduces a *nonstandard GA* approach using linear combinations in place of genetic crossover.

In this way, the philosophical themes of scatter search and genetic algorithms are being brought closer together by modern efforts to create GA variants with an improved ability to solve optimization problems. The potentially useful implications of this are compounded by the recent introduction of advanced processes for combining solutions, as described in the next section.

5. STRUCTURED COMBINATIONS

Operations used to define linear combinations can be replaced with *structured transformations* of vectors that preserve specified discrete relationships and associated feasibility conditions. This approach is based on three properties of the reference vectors from which the weighted combinations are created.

Property 1 (Representation property)

Each vector represents a set of votes for particular decisions (e.g. the decision of assigning a specific value to a particular variable, or of assigning a specific facility to a particular location etc.). Standard solution vectors for many problems can directly operate as such voting vectors, or can be expanded in a natural way to create such vectors. For example, a solution vector for a job shop scheduling problem can be interpreted as a set of 0–1 votes for predecessor decisions in scheduling specific jobs on particular machines.

Property 2 (Trial solution property)

The set of votes prescribed by a vector translates into a trial solution to the problem of interest by a well defined process. A set of votes for items to include in a knapsack, for example, can be translated into a trial solution by processing the votes sequentially in terms of benefit-to-weight ratios for votes in particular intervals, until either the knapsack is full or votes at all intervals are considered. (Note that the vectors may not represent feasible solutions to the problems considered, or even represent solutions in a customary sense at all.)

Property 3 (Update property)

If a decision is made according to the votes of a given vector, a clearly defined rule exists to update all vectors for the residual problem so that Properties 1 and 2 continue to hold. (For example, upon assigning a specific value to a particular variable, all votes for assigning different values to this variable are cancelled, and the remaining updated votes of each vector retain the ability to be translated into a trial solution for the residual problem in which the assignment has been made.)

The three preceding properties allow a constructive method to create structured combinations of vectors [71]. Voting schemes for evaluation in related contexts have been proposed by Glover and McMillan [5, 72] and Ackley [48], though in a less general form and without a rigorous design for exploiting and preserving the three fundamental properties. The properties imply that the decisions voted upon can be made in a sequential order, which is established either *a priori*, or as a function of the decision theory. Decision steps also may be linked by solving associated optimization problems. (For example, an assignment problem to initiate the decision steps for a traveling salesman tour can be created by combining the votes of permutation vectors with original distance data.) This type of linking is often performed effectively in settings that do not provide for generating combined solutions, and the ability to incorporate it as part of the process of creating new solutions from combinations of others is a useful feature.

The sequential decision structure implied by the preceding properties is sufficiently comprehensive to apply to optimization problems of nearly unlimited generality, and can directly incorporate standard neighborhood processes for transforming one solution into another. Going a step farther, it is possible to specify processes to score and evaluate vectors, taking account of different types of objectives, that enable reference points (or "parents" in GA terminology) to be combined according to any desired emphasis on their relative contributions. For example, motivated by the concept of weighting solutions as in scatter search, two solutions can be combined in various mixes such as 70–30, 60–40 or 50–50, to produce new outcomes that embody different degrees of contributions of the original solutions. This can be done either probabilistically or deterministically by employing appropriate mechanisms to exploit Properties 1–3.

Specific rules make it possible to generate a variety of such combined solutions that represent different relative emphasis on the parents, while simultaneously satisfying problem constraints. This may be conveniently illustrated for a permutation problem (which has very simple constraints embodied in sequential restrictions). Consider starting with the following two permutations:

1 2 3 4 5 6 7 8 9

1 8 6 4 9 3 5 2 7

Placing a two to one emphasis on favoring the first permutation over the second, the following three permutations are generated by rules identified in Glover [71]:

1 2 7 8 9 3 4 5 6

1 2 8 3 4 6 5 7 9

1 2 6 4 5 7 3 8 9

The differences in these three permutations do not result by changing their rules of combination, but simply by applying the rules to different objectives, respectively associated with traveling salesman problems, scheduling problems and facility location problems. (The indicated permutations are not the only ones generated by the rules for these three cases.)

The analogy to the application of different types of nongenetic crossover is apparent in this example. However, in addition to allowing different relative weights on the parents, there are two key distinctions. First, the rules follow directly from a general framework that yields these varied outcomes, each responsive to a particular objective, without having to invent different types of nonstandard crossover operations for each separate case. Second, and more importantly, the rules apply to problems with many different kinds of constraint structures, allowing feasibility to be handled automatically. A special class of transformations, called *adaptive structured combinations*, further enhances the process by allowing the problem objective to determine the relative mix as well as the priority system for combining solutions.

The structured combination ideas provide a useful extension of the linear combination ideas of scatter search along several dimensions. In creating structured combinations, additional opportunities for design and control are provided as the process unfolds. We will argue later for the importance of this reference to structure (from which these types of combinations derive their name), in the context of identifying additional connections to recent trends in GA methods.

6. STRATEGIC OSCILLATION AND GENETIC ALGORITHM LINKAGES

In the tabu search framework, strategic oscillation is a process used to vary the direction of search and the region visited by controlling the admissibility or evaluations of moves in some neighborhood of the current solution. One of its more useful applications is to create expanded search processes that can admit infeasible solutions during the search. Infeasible solutions are permitted, but they are penalized for their infeasibility. The "oscillation" component of the procedure varies this penalty over time. A typical oscillation may be in the form of a sine curve, although many other types of oscillation patterns have been successfully used. The common theme among all of the oscillation patterns used is that the penalty alternates between values that encourage or discourage infeasible solutions (in the latter case, sometimes seeking to drive deeper within a feasible region).

There are at least three reasons for considering the use of this form of stragetic oscillation when solving optimization problems [43, 45]. One, if the feasible solution space of a problem consists of nonconvex or disjoint components, then strategic oscillation provides a mechanism for crossing regions of infeasibility in the course of searching for the optimal solution. In some problems, where in fact the feasible region may be connected, an attempt to reach an optimal solution may require a long tortuous path through feasible space, while if a solution path is allowed to enter infeasible regions, then an optimum may be easily found.

A second reason for using strategic oscillation is that in some problems the goal of obtaining a feasible solution is as hard as obtaining an optimal solution—i.e. from the standpoint of theoretical complexity, both pose models that are NP-complete. For this class of problems, strategic oscillation can be used to locate high quality, feasible solutions in a way that is analogous to primal–dual optimization approaches, and in fact can be used to coordinate alternating relaxation and restriction phases in an optimization setting.

A third desirable attribute of strategic oscillation is an indirect effect. As we have noted, any heuristic search that aspires to find optimal solutions must ensure sufficient diversity in the search. Strategic oscillation provides diversity by emphasizing different parts of the problem over time, and thus improves the robustness of the method.

Typically, genetic algorithms do not allow cross-over operations that would produce infeasible solutions. In restricted instances where infeasible solutions are permitted, they are usually transformed into feasible solutions before they are placed back into the population. Strategic oscillation provides a means to allow infeasible offspring to exist in the population. By enriching the population with infeasible solutions, a genetic algorithm coupled with strategic oscillation gains the power to operate with increased diversity, which may improve its chances of finding optimal solutions.

A straightforward way to incorporate strategic oscillation into a genetic algorithm is as follows. First, a cross-over operation that admits infeasible solutions is employed. Second, a feasibility measure is introduced. Typically, this measure quantifies the amount by which a solution violates its constraints. For example, a heuristic for a Traveling Salesman Problem that produces a solution

with four subtours may be assigned an infeasibility measure of three (4 − 1), perhaps refined by weighting each subtour as an inverse function of its size. Third, a suitable set of penalty factors must be obtained, to scale the infeasibility measure relative to the objective function. The fitness of a solution may then be calculated as follows:

$$Fitness = Objective\ Value + Penalty(t) \bigstar Infeasibility\ Measure.$$

The penalty and infeasibility measure terms may be conceived as vectors, hence producing a weighted sum of values, whose component terms are either linear or nonlinear. If the problem is a minimization problem, then "better" fitness levels are given by smaller *Fitness* values. *Penalty*(t) is controlled by strategic oscillation as a sequence of penalty factors that oscillate between large and small values (where t denotes the control parameter for the oscillation). The best strategy for manipulating t is probably problem dependent, although as in tabu search, general strategies may emerge that are relevant for special classes of problems. As changes to t cause *Penalty*(t) to decrease, infeasible solutions are permitted to enter the population, which then may in turn be gradually driven out by changes that cause *Penalty*(t) to increase. In this way the population will alternate between populations with different mixes of feasible and infeasible members.

We emphasize that other ways of incorporating the strategic oscillation approach within genetic algorithms are also possible. For example, such an approach has been applied in tabu search to control alternation between constructive and destructive steps for graph problems, and in general the feasibility–infeasibility dichotomy can be replaced by a variety of other dichotomies to induce the search to oscillate relative to a selected strategic framework. The usefulness of strategic oscillation as a component of tabu search suggests that its incorporation within genetic algorithms warrants examination.

Next we identify a type of strategy from tabu search founded on a spatial analogy and guided by reference to attributes of solutions, as normally used to define tabu restrictions in TS procedures. Our description in the next two sections parallels that of Glover and Laguna [29].

7. PATH RELINKING

Path relinking is an approach for pursuing both intensification and diversification concerns in tabu search. Employing the generalized concept of combination implicit in our preceding discussions, this approach also can be viewed as a way of combining solutions.

The process of path relinking is initiated by selecting two solutions x' and x'' from a collection of eilite solutions (high quality local optima produced during previous search phases). A path is then generated from x' to x'', producing a solution sequence $x' = x(1), x(2), \ldots, x(r) = x''$. A standard neighborhood structure is used to define available moves for transitioning from one solution to the next, and $x(i+1)$ is created from $x(i)$ at each step by choosing a move that leaves the fewest number of moves remaining to reach x'', or an approximation to this criterion. The path generation process may be applied to various choices of pairs, and as in scatter search. Upon completion one or more of the solutions $x(i)$ is selected to initiate a new search phase with a chosen heuristic procedure.

It may or may not be the case that x' and x'' were previously joined by a search trajectory as a result of some sequence of moves applied at an earlier stage. If they were joined in this manner, however, the new trajectory produced by path relinking is likely to be somewhat different, representing a "more direct" route between these solutions. An illustration of this is provided in Diagram 2.

A number of alternative moves typically will qualify to produce a next solution from $x(i)$ at each step by variations of the "fewest remaining moves" criterion, consequently allowing a variety of possible paths from x' to x''. Let $c(x)$ denote an objective function which is to be minimized (by picking an appropriate x over some feasible region). Selecting unattractive moves, relative to $c(x)$ at each step will tend to produce a final series of strongly improving moves, while selecting attractive moves will tend to produce lower quality moves at the end. (The last move, however, will be improving, or leave $c(x)$ unchanged, since x'' is a local optimum.) Thus, choosing best, worst or average moves, provide options that produce contrasting effects in generating the indicated sequence. (An aspiration criterion may be used to override choices in the last two cases if a sufficiently

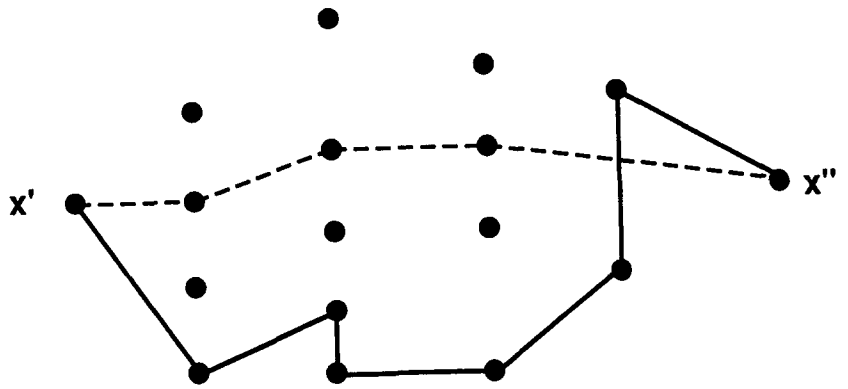


Diagram 2. Path relinking.

attractive solution is available. In general, it appears reasonable to select best moves at each step, and then repeat the process by interchanging x' and x''.)

The choice of one or more solutions $x(i)$ to launch a new search phase preferably should depend not only on $c[x(i)]$ but also on the values $c(x)$ of those solutions x that can be reached by a move from $x(i)$. In particular, when $x(i)$ is examined to move to $x(i+1)$, a number of candidates for $x = x(i+1)$ will be presented for consideration. The process additionally may be varied to allow solutions to be evaluated other than those that yield $x(i+1)$ closer to x''.

Let $x^*(i)$ denote a neighbor of $x(i)$ that yields a minimum $c(x)$ value during an evaluation step, excluding $x^*(i) = x(i+1)$. [If the choice rules do not automatically eliminate the possibility $x^*(i) = x(h)$ for $h < i$, then a simple tabu restriction can be used to do this.] Then the method selects a solution $x^*(i)$ that yields a minimum value for $c(x^*(i))$ as a new point to launch the search. If only a limited set of neighbors of $x(i)$ are examined to identify $x^*(i)$, then a superior least cost solution $x(i)$, excluding x' and x'', may be selected instead. Early termination may be elected upon encountering an $x^*(i)$ that yields $c[x^*(i)] < \min[c(x'), c(x''), c[x(p)]$, where $x(p)$ is the minimum cost $x(h)$ for all $h \leqslant i$. [The procedure continues without stopping if $x(i)$, in contrast to $x^*(i)$, yields a smaller $c(x)$ value than x' and x'', since $x(i)$ effectively adopts the role of x'.]

Variation and tunneling

A variant of the path relinking approach starts at both endpoints x' and x'' simultaneously, producing two sequences $x' = x'(1), \ldots, x'(r)$ and $x'' = x''(1), \ldots, x''(s)$. The choices are designed to yield $x'(r) = x''(s)$, for final values of r and s. To progress toward this outcome when $x'(r) \neq x''(s)$, either $x'(r)$ is selected to create $x'(r+1)$, by the criterion of minimizing the number of moves remaining to reach $x''(s)$, or $x'(s)$ is chosen to create $x''(s+1)$, by the criterion of minimizing the number of moves remaining to reach $x'(r)$. From these options, the move is selected that produces the smallest $c(x)$ value, thus also determining which of r or s is incremented on the next step.

The path relinking approach can benefit by a tunneling strategy that allows a different neighborhood structure to be used than in the standard search phase. In particular, it can be desirable to periodically allow moves for path relinking that normally would be excluded due to creating infeasibility. Such a practice is less susceptible to becoming "lost" in an infeasible region than other ways of allowing periodic infeasibility, since feasibility evidently must be recovered by the time x'' is reached. This tunneling effect therefore offers a chance to reach solutions that might otherwise be bypassed. In the variant that starts from both x' and x'', at least one of $x'(r)$ and $x''(s)$ may be kept feasible.

Greater emphasis on intensification or diversification is achieved in this approach by choosing x' and x'' to share more or fewer attributes in common. For example, choosing x' and x'' from a clustered set of elite solutions will stimulate intensification, while choosing them from two widely separated sets will stimulate diversification.

Extrapolated relinking

The path relinking approach extends naturally beyond consideration of points "between" x' and x'' in the same way that linear combinations extend beyond points that are expressed as convex combinations of two endpoints. For simplicity, suppose we begin at x' and seek a path that continues beyond x''. The ability to go beyond this endpoint results by a method for approximating the criterion of choosing a move that leaves the fewest moves remaining to reach x''. Specifically, let $A(x)$ denote the set of solution attributes associated with ("contained in") x, and let *A_drop* denote the set of solution attributes that are dropped by moves performed to reach the current solution $x'(i)$. Such attributes may actually be components of the x vectors themselves, or may be related to these components of the x vectors themselves, or may be related to these components by appropriately defined mappings [4].

Define a *to-attribute* of a move to be an attribute of the solution produced by the move, but not an attribute of the solution that initiates the move. Similarly, define a *from-attribute* to be an attribute of the initating solution but not of the new solution produced. Then we seek a move at each step to maximize the number of *to-attributes* that blong to $A(x'') - A[x(i)]$, and subject to this to minimize the number that belong to $A\_drop - A(x'')$. Such a rule generally can be implemented very efficiently by appropriate data structures.

Once $x(r) = x''$ is reached, the process continues by modifying the choice rule as follows. The criterion now selects a move to maximize the number of its *to-attributes* not in *A_drop* minus the number of its *to-attributes* that are in *A_drop*, and subject to this to minimize the number of its *from-attributes* that belong to $A(x'')$. (The combination of these criteria establishes an effect analogous to that achieved by the standard algebraic formula for extending a line segment beyond an endpoint. However, the secondary minimization criterion is probably less important.) The path then stops whenever no choice remains that permits the maximization criterion to be positive.

For neighborhoods that allow relatively unrestricted choices of moves, this approach yields an extension beyond x'' that introduces new attributes, without reincorporating any old attributes, until no move remains that satisfies this condition. The ability to go beyond the limiting points x' and x'' creates a form of diversification not available to the path that "lies between" these points. At the same time the exterior points are influenced by the trajectory that links x' and x''.

8. CREATING NEW ATTRIBUTES

The last concept from tabu search we will discuss involves the creation of new attributes out of others. We focus on creating new attributes by reference to a process called vocabulary building, related to concept formation.

Vocabulary building is based on analyzing a chosen set S of solutions to discover attribute combinations its members share in common with each other and with various feasible solutions in general. Attribute combinations that emerge as significant enough to qualify as units of vocabulary, by a process to be described below, are treated as new attributes capable of being incorporated into tabu restrictions and aspiration conditions of the type illustrated in Section 2, and of additional more general types illustrated in tabu search references previously cited. In addition, the combination can be directly assembled into larger units as a basis for constructing new solutions.

Collections of attributes may be represented for our present purposes by encoding them as assignments of values to variables, which we denote by $y_j = p$, to differentiate the vector y from the solution vector x which possibly may have a different dimension and encoding. Normally we suppose a y vector contains enough information to be transformed into a unique x, to which it corresponds, but this assumption can be relaxed to allow more than one x to yield the same y.

Let $Y(S)$ denote the collection of y vectors corresponding to the chosen set S of x vectors. In addition to assignments of the form $y_j = p$ which define attributes, we allow each y_j to receive the value $y_j = \bigstar$, in order to generate subvectors that identify specific attribute combinations. Such a value represents an *attribute clash*, as contrasted with a *wild card* for an element of a bit string, as in the case of a "$\bigstar$ value" in GAs. In the present setting, an attribute combination will be implictly determined by the non-$\bigstar$ values of y.

The approach to generate vocabulary units will be to compare vectors y' and y'' by an intersection

operator, $Int(y', y'')$, to yield a vector $z = \text{Int}(y', y'')$ by the rule: $z_j = y'_j$ if $y'_j = y''_j$, and $z_j = \bigstar$ if $y'_j \neq y''_j$. By this definition we also obtain $z_j = \bigstar$ if either $y'_j = \bigstar$ or $y''_j = \bigstar$. *Int* is associative, and the intersection $Int(y: y \in Y)$, for an arbitrary Y, yields a z in which $z_j = y_j$ if all y_j have the same value for $y \in Y$, and $z_j = \bigstar$ otherwise.

Accompanying the intersection operator, we also define a relation of containment, by the stipulation that y'' contains y' if $y'_j = \bigstar$ for all j such that $y'_j \neq y''_j$. Accompanying this relation, we identify the *enclosure* of y' (relative to S) to be the set $Y(S: y') = \{y \in Y(S): y \text{ contains } y'\}$, and define the enclosure value of y', $enc\_value(y')$, to be the number of elements in this set. Finally, we refer to the number of non-$\bigstar$ components of y' as the *size* of the vector, denoted $size(y')$. [If $y \in Y(S)$, the size of y is the same as its dimension.]

As a general tendency, with some exceptions, the greater $size(y')$ to becomes, the smaller $enc\_value(y')$ becomes. Thus for a given size s, we seek to identify vectors y' with $size(y') \geq s$ that maximize $enc\_value(y')$, and for a given enclosure value v to identify vectors y' with $enc\_value(y') \geq v$ that maximize $size(y')$. Such vectors are included among those regarded to quality as vocaculary units.

Similarly we include reference to weighted enclosure values, where each $y \in Y(S)$ is weighted by a measure of attractiveness [such as the value $c(x)$ of an associated solution $x \in S$], to yield $enc\_value(y')$ as a sum of the weights over $Y(S: y')$. Particular attribute values likewise may be weighted to yield a weighted value for $size(y')$, equal to the sum of weights over non-$\bigstar$ components of y'.

At a higher level, we seek vectors as vocabulary units that give rise to aggregate units called *phrases* and *sentences* with certain properties of consistency and meaning, characterized as follows. Each y_j is allowed to receive one additinal value, $y_j = blank$, which may be interpreted as an empty space free to be filled by another value (in contrast to $y_j = \bigstar$, which may be interpreted as a space occupied by two conflicting values). We begin with the collection of vectors created by the intersection operator *Int*, and replace the $\bigstar$ value with *blank* values in these vectors. We then define an extended intersection operator *E_Int*, where $z = E\_Int(y', y'')$ is given by the rules defining *Int* if y'_j and y''_j are not *blank*. Otherwise $z_j = y'_j$ if $y''_j = blank$, and $z_j = y''_j$ if $y'_j = blank$. *E_Int* likewise is associative. The vector $z = E\_Int(y: y \in Y)$ yields $z_j = \bigstar$ if any two $y \in Y$ have different non-*blank* values y_j, or if some y has $y_j = \bigstar$. Otherwise z_j is the common y_j value for all y with y_j non-blank (where $z_j = blank$ if $y_j = blank$ for all y).

The y vectors created by *E_Int* are those we call *phrases*. A sentence is a phrase that has no *blank* values. We call a phrase or sentence *grammatical* (logically consistent) if it has no $\bigstar$ values. Grammatical sentences thus are y vectors lacking both *blank* values and $\bigstar$ values, constructed from attribute combinations (subvectors) derived from the original elements of $Y(S)$. Finally we call a grammatical sentence y *meaningful* if it corresponds to, or maps into, a feasible solution x. (Sentences that are not grammatical do not have a form that permits them to be translated into an x vector, and hence cannot be meaningful.)

The elements of $Y(S)$ are all meaningful sentences, assuming they are obtained from feasible x vectors, and the goal is to find other meaningful sentences obtained from grammatical phrases and sentences constructed as indicated. More precisely, we are interested in generating meaningful sentences (hence feasible solutions) that are not limited to those that can be obtained from $Y(S)$, but that also can be obtained by one of the following strategies:

(S1) Translate a grammatical phrase into a sentence by filling in the blanks (by the use of neighborhoods that incorporate constructive moves);

(S2) Identify some set of existing meaningful sentences (e.g. derived from current feasible x vectors not in S), and identify one or more phrases, generated by *E_Int* over S, that lie in each of these sentences. Then, by a succession of moves from neighborhoods that preserve feasibility, transform each of these sentences into new meaningful sentences that retain as much of the identified phrases as possible;

(S3) Identify portions of existing meaningful sentences that are contained in grammatical phrases, and transform these sentences into new meaningful sentences (using feasibility-preserving neighborhoods) by seeking to incorporated additional components of the indicated phrases.

The foregoing strategies can be implemented by incorporating standard tabu search incentive and penalty mechanisms for choosing moves. We assume in these strategies that neighborhood operations on x vectors are directly translated into associated changes in y vectors. In the case of (S1) there is no assurance that a meaningful sentence can be achieved unless the initial phrase itself is meaningful (i.e. is contained in at least one meaningful sentence) and unless the constructive process is capable of generating an appropriate completion. Also, in (S3) more than one grammatical phrase can contain a given path (subvector) of a meaningful sentence, and it may be appropriate to allow the targeted phrase to change according to possibilities consistent with available moves.

Specific instances of vocabulary building processes can profit from special algorithms for linking vocabulary units into sentences that are both meaningful and attractive, in the sense of creating good $c(x)$ values. An example of this is provided by vocabulary building approaches for the traveling salesman problem described in [19], where vocabulary units can be transformed into tours by specialized shortest path procedures. A number of combinatorial optimization problems are implicit in generating good sentences by these approaches, and the derivation of effective methods for handing these problems in various settings, as in the case of the traveling salesman problem, may provide a valuable contribution to search procedures generally.

9. GA COMPARISONS AND FOUNDATIONS FOR HYBRIDS

We have already intimated several ways that ideas from tabu search share a kindred philosophy with certain ideas from genetic algorithms. We expand these observations with comments about fundamental mechanisms, and their embodiment in strategies previously described, that give additional possibilities to establish useful links between the methods.

Starting at a very basic level, the alleles of genetic algorithms, which correspond to values of variables in a binary solution vector, may be compared at a first level of approximation to attributes in tabu search. Introducing memory in GAs to track the history of alleles over subpopulations therefore would provide an immediate and natural way to create a hybrid with TS. Mühlenbein [51] has similarly observed the relevance of applying TS memory structures in GAs.

Some important differences between alleles and attributes are worth noting, however. Differentiation of attributes into *from* and *to* components, each having different memory functions, does not have a counterpart in genetic algorithms. This results because of the GA tradition of operating without reference to neighborhood structures and associated moves (although, strictly speaking, combination by crossover can be viewed as a special type of move). Another distinction derives from differences in the use of coding conventions. Although an attribute change in tabu search, from a state to its complement, can be encoded in a zero–one variable, such a variable does not necessarily provide a convenient or useful representation for the transformations provided by moves. We have already noted limitations of binary representations in Sections 3 and 4. Tabu restrictions and aspiration criteria handle the binary aspects of complementarity without requiring explicit reference to a zero–one x vector or two-valued functions. Adopting a similar orientation (relative to the special class of moves embodied in crossover) might yield benefits for genetic algorithms in dealing with issues of genetic representation, which currently pose difficult questions [73].

Another contrast between genetic algorithms and tabu search that deserves further emphasis occurs in the treatment of context, i.e. in the consideration given to structure inherent in different problem classes. For tabu search, context is fundamental, embodied in the interplay of attribute definitions and the determination of move neighborhoods, and in the choice of conditions to define tabu restrictions. Context is also implicit in the identification of amended evaluations created in association with longer term memory.

At the opposite end of the spectrum, GA methods have customarily stressed the freedom of their rules from the influence of context. Crossover, by this view, is a *context neutral* operation, which assumes no reliance on conditions that solutions must obey in a particular problem setting, just as genes make no reference to the environment as they follow their instructions for recombination. As noted in Section 3, however, practical application generally renders this an inconvenient assumption, making solutions of interest difficult to find. Consequently, a good deal of effort in GA implementation is devoted to developing "special crossover" operations that compensate for the difficulties created by context, effectively reintroducing it on a case by case basis.

The chief method by which modern genetic algorithms handle structure is by relegating its treatment to some other method, as in the case of PGAs and genetic local search. Genetic algorithms of this type combine solutions by their parent–children processes at one level, and then a descent method takes over to operate on the resulting solutions to produce new solutions. These new solutions in turn are submitted to be recombined by the GA processes. Viewed from a slightly different perspective, GAs that rely on other processes to exploit structure already are hybrid methods, and can just as well make use of tabu search designs for such exploitation, hence giving a natural basis for marrying GA and TS procedures. But genetic algorithms and tabu search also can be joined in a more fundamental way.

Specifically, tabu search strategies for intensification and diversification are based on the question: how can information be extracted from a set of good solutions to help uncover additional (and better) solutions? From one point of view, GAs provide an approach for answering this question, consisting of putting solutions together and interchanging components (in some loosely defined sense, if traditional crossover is not strictly enforced). Tabu search, by contrast, seeks an answer by utilizing processes that specifically focus on problem structure and that incorporate it as fully as possible into their design.

Augmented by historical information, properties of solution neighborhoods are used in tabu search as a basis for applying penalties and incentives to induce attributes of good solutions to become incorporated into current solutions. Consequently, although it may be meaningless to interchange or otherwise incorporate a set of attributes from one solution into another in a wholesale fashion, as attempted in recombination operations, a stepwise approach to this goal through the use of neighborhood structures is entirely practicable. This concept is transformed into explicit formulations for achieving meaningful combinations of solutions by the derivation of structured combinations, as described in Section 5, and by the introduction of path relinking, as described in Section 7. Instead of being compelled to create new types of crossover to remove deficiencies of standard operators upon being confronted by changing contexts, this approach addressed context directly and makes it an essential part of the design for generating combinations. Proposals advanced by Mühlenbein [16] characterize a segment of the GA field that is increasingly compatible to adopting such an approach, and this could provide a basis for a signficant hybrid combination of genetic algorithm and tabu search ideas.

Finally, we observe that the realm of attribute creation, making use of vocabulary building processes as described in Section 8, offers the possibility for an analogous genetic interpretation. Vocabulary units may suggestively be given the alternate name of "genetic material". By this means, such units may be viewed as substrings of genes, created by a process that selectively extracts them to establish a substring pool. (This suggests a natural connection with schemata, although as previously noted, the components of an attribute vector in vocabulary building are somewhat different than the components of schemata.) As elements are accumulated from different sources within such a pool, and progressively reintegrated to form *phrases* and *sentences* by vacabulary processes, a genetic parallel may be conceived of incorporating substring templates to guide construction of new genes.

Perhaps the use of such evolving substring pools, as opposed to the exclusive focus on parents and children which supposedly allow substrings to evolve indirectly, would prove useful in genetic algorithms. A step in this direction has been initiated with the "messy" genetic algorithms of Goldberg *et al.* [74]. But there are limiting factors, since the TS processes for creating vocabulary are based on conscious and strategic reconstruction, and hence from this standpoint do not much resemble genetic processes. To preserve the genetic metaphor, one may imagine relying on *intelligent enzymes*, operating as special subroutines to cut out appropriate components and then recombine them according to systematic principles. If this is not stretching analogy too far, the outcome may qualify as an interesting hybrid of the GA and TS approaches.

The fundamental concept of attribute based memory in tabu search, whether applied to attributes that are created or that are determined by prior analysis, yields a means of controlling search by rules that take advantage of interdependencies among subsets of elements. This theme of exploiting memory in relation to structure, which is characteristic of organisms that respond to their surroundings at higher levels than manifested in simple genetic recombination, provides a promising avenue for integrating tabu search and genetic algorithms. The relevance of this possibility has also been observed by Mühlenbein [51].

The foregoing opportunities for harmonizing what may at first appear to be somewhat disparate approaches, and to take advantage of features that may be mutually reinforcing, offer stimulating possibilities for future research. If the GA emphasis on the value of "combination" as a basis for improvement is valid, then we may look forward to new advances by linking the ideas of these two fields.

Acknowledgement—This research is supported in part by the Joint Air Force Office of Scientific Research and Office of Naval Research Contract No. F49620-90-C-0033, at the University of Colorado.

REFERENCES

1. F. Glover, Future paths for integer programming and links to artificial intelligence. *Computers Ops Res.* **5**, 533–549 (1986).
2. F. Glover, *Tabu search—part I. ORSA Jl Comput.* **1**, 190–206.
3. P. Hansen, The steepest ascent mildest descent heuristic for combinatorial programming. *Numerical Methods in Combinatorial Optimization*, Capri, Italy (1986).
4. F. Glover and M. Laguna, Tabu search. In *Modern Heuristic Techniques for Combinatorial Problems* (Edited by C. Reeves). Blackwell Scientific, Oxford (1993).
5. F. Glover and C. McMillan, The general employee scheduling problem: an integration of management science and artificial intelligence. *Computers Ops Res.* **15**, 563–593 (1986).
6. M. Widmer and A. Hertz, A new method for the flow sequencing problem. *Eur. J. Ops Res.* **41**, 186–193 (1989).
7. E. Taillard, Some efficient heuristic methods for the flowshop sequencing problem. *Eur. J. Ops Res.* **47**, 65–74 (1990)
8. M. Widmer, Job shop scheduling with tooling constraints: a tabu search approach. *J. Opl Res. Soc.* **24**, 75–82 (1991).
9. J. Bovet, C. Constantin and D. de Werra, A convoy scheduling problems. *Discrete Appl. Math.* (in press).
10. M. J. Laguna, W. Barnes and F. Glover, Tabu search methods for a single machine scheduling problem. *J. Intell. Manufac.* **2**, 63–74 (1991).
11. M. Laguna and J. L. Gonzalez-Verlarde, A search heuristic for just-in-time scheduling in parallel machines'. *J. Intell. Manufac.* **2**, 253–260 (1991).
12. J. W. Barnes and M. Laguna, Solving the multiple-machine weighted flow time problem using tabu search. *IIE Trans.* **25**, 121–130 (1993).
13. R. L. Daniels and J. B. Mazzola, A tabu search heuristic for the flexible-resource flow shop scheduling problem. *Ann. Ops Res.* **41**, 207–230 (1993).
14. M. Dell'Amico and M. Trubian, Applying tabu search to the job-shop scheduling problem. *Ann. Ops Res.* **41**, 231–252 (1993).
15. M. Laguna and F. Glover, Integrating target analysis and tabu search for improved scheduling systems. *Expert Systems Applic.* **6**, 287–298 (1993).
16. E. L. Mooney and R. L. Rardin, Tabu search for a class of scheduling problems. *Ann. Ops Res.* **41**, 253–278 (1993).
17. D. L. Woodruff and M. L. Spearman, Sequencing and batching for two classes of jobs with deadlines and setup times. *Prod. Ops Mangmnt* **1**, 87–102 (1992).
18. M. Malek, M. Guruswamy, M. Pandya and H. Owens, Serial and parallel simulated annealing and tabu search algorithms for the travelling salesman problem. *Ann. Ops Res.* **21**, 59–84 (1989).
19. F. Glover, Ejection chains, reference structures, and alternating path methods for the traveling salesman problem. Graduate School of Business and Administration, University of Colorado at Boulder (1992).
20. M. Gendreau, A. Hertz and G. Laporte, A tabu search heuristic for the vehicle routing problem. *Mangmnt Sci.* (in press).
21. I. H. Osman, Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Ann. Ops Res.* **41**, 421–452 (1993).
22. F. Semet and E. Taillard, Solving real-life vehicle routing problems efficiently using taboo search. *Ann. Ops Res.* **41**, 469–488 (1993).
23. J. Skorin-Kapov, Tabu search applied to the quadratic assignment problem. *ORSA J. Comp.* **2**, 33–45 (1990).
24. E. Taillard, Taboo search for the quadratic assignment problem. *Parallel Comput.* **17**, 443–455 (1991).
25. J. Chakrapani and J. Skorin-Kapov, Massively parallel tabu search for the quadratic assignment problem. *Ann. Ops Res.* **41**, 327–342 (1993).
26. J. A. Bland and G. P. Dawson, Tabu search and design optimization. *Computer-Aided Des.* **23**, 195–202 (1991).
27. S. Oliveira and G. Stroud, A parallel version of tabu search and the path assignment problem. *Heuristics Combinat. Optimiz.* **4**, 1–24 (1989).
28. C. A. Anderson, K. F. Jones, M. Parker and J. Ryan, Path assignment for call routing: an application of tabu search. *Ann. Ops Res.* **41**, 301–312 (1993).
29. F. Glover and M. Laguna, Bandwidth packing: a tabu search approach. *Mangmnt Sci.* **39**, 492–500 (1993).
30. F. Glover, C. McMillan and B. Novick, Interactive decision software and computer graphics for architectual and space planning. *Ann. Ops. Res.* **5**, 557–573 (1985).
31. P. Hansen, B. Jaumard and Da Silva, Average linkage divisive hierarchical clustering. *J. Classif.* (in press).
32. U. Dorndorf and E. Pesch, Fast clustering algorithms. *ORSA J. Comput.* **6**, 141–153 (1994).
33. A. Hertz, D. de Werra, Using tabu search techniques for graph coloring. *Computing* **29**, 345–351 (1987).
34. A. Hertz, B. Jaumard and M. Poggi di Aragao, Topology of local optima for the K-coloring problem. *Discrete Appl. Math.* **49**, 257–280 (1994).
35. C. Friden, A. Hertz and D. de Werra, Stabulus: a technique for finding stable sets in large graphs with tabu search. *Computing* **42**, 35–44 (1989).
36. M. Gendreau, P. Soriano and L. Salvail, Solving the maximum clique problem using a tabu search approach. *Ann. Ops Res.* **41**, 385–404 (1993).
37. P. Hansen and B. Jaumard, Algorithms for the maximum satisfiability problem. *Computing* **44**, 279–303 (1990).
38. B. Jaumard, P. Hansen and M. Poggi di Aragao, Column generation methods for probabilistic logic. *ORSA J. Comput.* **3**, 135–148 (1991).

39. P. Hansen, B. Jaumard and M. Poggi di Aragao, Mixed integer column generation algorithms and the probabilistic maximum satisfiability problem. *Proc. 2nd Integer Programming and Combinatorial Optimization Conf.* Carnegie Mellon (1992).
40. D. de Werra and A. Hertz, Tabu search technique: a tutorial and an applications to neural networks. *OR Spectrum* **11**, 131–141 (1989).
41. D. Beyer and R. Ogier, Tabu learning: a neural network search method for solving nonconvex optimization problems. *Proc. Int. Joint Conf. Neural Networks.* IEEE and INNS, Singapore (1991).
42. F. Dammeyer and S. Voss, Dynamic tabu list management using the reverse elimination method. *Ann. Ops Res.* **41**, 31–46 (1993).
43. J. P. Kelly, B. L. Golden and A. A. Assad, Large-scale controlled rounding using tabu search with strategic oscillation. *Ann. Ops Res.* **41**, 69–84 (1993).
44. M. Sun and P. G. McKeown, Tabu search applied to the general fixed charge problem. *Ann. Ops Res.* **41**, 405–420 (1993).
45. F. Glover, E. Taillard and D. de Werra, A user's guide to tabu search. *Ann. Ops Res.* **41**, 3–28 (1993).
46. D. E. Goldberg, *Genetic Algorithms in Sarach, Optimization, and Machine Learning.* Addison-Wesley, Reading, MA (1989).
47. L. Davis (Editor), *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, New York (1991).
48. D. Ackley, *A Connectionist Model for Genetic Hillclimbing.* Kluwer Academic, London (1987).
49. Y. Davidor, An intuitive introduction to genetic algorithms as adaptive optimizing procedures. Technical Report CS90-07, The Weizmann Institute of Science (1990).
50. E. H. L. Aarts, A. E. Eiben and K. M. van Hee, A general theory of genetic algorithms. *Computing Science Notes, 89/8.* Eidenhoven University of Technology (1989).
51. H. Mühlenbein, Parallel genetic algorithms in combinatorial optimization. *Computer Science and Operations Research* (Edited by Osman Balci), Pergamon Press, Oxford (in press).
52. K. A. DeJong, Genetic algorithms are NOT function optimizers. Computer Science Department, George Mason University (1992).
53. U. Dorndorf and E. Pesch, Evolution based learning in a job shop scheduling environment. Technical Report, University of Limberg, The Netherlands (1992).
54. L. Davis, Job shop scheduling with genetic algorithms. *Conf. Genetic Algorithms,* Carnegie Mellon University (1985).
55. H. Mühlenbein, M. Gorges-Schleuter and O. Krämer, New solutions to the mapping problem of parallel systems—the evolution approach. *Parallel Comput.* **6**, 269–279 (1987).
56. D. Whitley, Starkweather and Fuquay, Scheduling problems and traveling salesmen: the genetic edge recombination operator. ICGA, Morgan Kaufman (1989).
57. J. J. Grefenstette, R. Gopal, B. Rosmaita and D. Van Gucht, Genetic algorithms for the traveling salesman problem. *Proc. 1st Int. Conf. Genetic Algorithms and Their Applications,* pp. 160–168. Lawrence Erlabum, Hillsdale, NJ (1985).
58. H. Mühlenbein, M. Gorges-Schleute and O. Krämer, Evolution algorithms in combinatorial optimization. *Parallel Comput.* **7**, 65–88 (1988).
59. D. J. Montana and L. Davis, Training feedforward neural networks using genetic algorithms. *Proc. 1989 Int. Joint Conf. Artificial Ingelligence,* San Mateo, CA, Morgan Kaufmann (1989).
60. C. L. Huntley and D. E. Brown, Parallel genetic algorithms with local search. The Institute for Parallel Computation and The Department of Systems Engineering.
61. N. L. J. Ulder, E. Pesch, P. J. M. van Laarhoven, H. J. Bandelt and E. H. L. Aarts, Genetic local search algorithm for the traveling salesman problem. *Parallel Problem Solving from Nature* (Edited by R. Maenner and H. P. Schwefel), *Lectures in Computer Science,* Vol. 496, pp. 109–116. Springer, Berlin.
62. D. Whitley, K. Mathias and P. Fitzhorn, Delta coding: an iterative search strategy for genetic algorithms. *Proc. 1991 Int. Conf. Genetic Algorithms* (1991).
63. F. Glover, Heuristics for integer programming using surrogate constraints. *Decision Sci.* **8**, 156–166 (1977).
64. J. H. Holland, *Adaptation in Natural and Artificial Systems.* The University of Michigan Press, Ann Arbor, MI (1975).
65. F. Glover, A bound escalation method for the solution of integer linear programs. *Cahiers Rech. Op.* **6**, 131–168 (1964).
66. F. Glover, Integer programming over a finite additive group. *SIAM Jl Control* **7**, 213–231 (1969).
67. F. Glover, Cut search methods in integer programming. *Math. Progr.* **3**, 86–100 (1972).
68. T. Bäck, F. Hoffmeister and H. Schwefel, A survey of evolution strategies. *Proc. 4th Int. Conf. Genetic Algorithms,* pp. 2–9. Morgan Kaufmann, San Mateo, CA (1991).
69. L. J. Eschelman and J. D. Schaffer, Real-coded genetic algorithms and interval-schemata. Technical Report, Phillips Laboratories (1992).
70. Z. Michalewicz, G. A. Vignaux and M. Hobbs, A non-standard genetic algorithm for the nonlinear transportation problem. *ORSA Jl on Computing* **3**, 307–316 (1991).
71. F. Glover, Tabu search for nonlinear and parametric optimization (with links to genetic algorithms). *Discrete Appl. Math.* **49**, 231–256 (1994).
72. D. E. Goldberg, B. Korb and K. Deb, Messy genetic algorithms: motivation, analysis and first results. *Complex Systems* **3**, 493–530 (1989).
73. G. Liepins and M. D. Vose, Representational issues in genetic optimization. *J. Exp. Theoret. Artific. Intell.* **2**, 101–105 (1990).
74. U. Faigle and W. Kern, Some convergence results for probabilistic tabu search. *ORSA Jl Comput.* **4**, 32–37 (1992).
75. J. P. Kelly, M. Laguna and F. Glover, A study of diversification strategies for the quadratic assignment problem. *Computers Ops Res.* **21**, (1994).
76. M. Laguna, J. P. Kelly, J. L. Gonzalez-Verlarde and F. Glover, Tabu search for the multilevel generalized assignment problem. Graduate School of Business and Administration, *Eur. J. Ops Res.* (in press).
77. C. Reeves, Improving the efficiency of tabu search for machine sequencing problems. *J. Ops Res. Soc.* (in press).
78. G. Syswerda, Uniform crossover in genetic algorithms. (Edited by J. David Schaffer), *Proc. 3rd Int. Conf. Genetic Algorithms,* San Mateo, CA, Morgan Kaufamann Publishers (1989).
79. D. Whitley, GENITOR: a different genetic algorithm. *Proc. Rocky Mountain Conf. Artificial Intelligence,* Denver, Colorado (1988).
80. D. L. Woodruff and E. Zemel, Hashing vectors for tabu search. *Ann. Ops Res.* **41**, 123–138 (1993).