The Edge-Window-Decoder Representation for Tree-Based Problems

Sang-Moon Soak, David. W. Corne, and Byung-Ha Ahn

***Abstract*—We describe the "edge-window-decoder" strategy (EWD), a decoder-based redundant encoding strategy for tree-based combinatorial problems, and explore its performance on three well-known tree-based network optimization problems: the degree constrained minimum spanning tree problem (DCMST), the optimum communication spanning tree problem (OCST), and the quadratic minimum spanning tree problem (q-MST). Each is an NP-hard problem, and in each case the best previously published approach (for obtaining fast solutions on large instances) is an evolutionary algorithm (EA), characterized by a specialized encoding. EWD simply encodes a tree as a list of nodes (in which nodes may be repeated), and a tree is built from this list via a "tree construction routine" (TCR). A particular instantiation of EWD involves choosing a specific TCR, and choosing specific designs for the mutation and crossover operators (which work only at the genotype (list of nodes) level). Different combinations of TCRs and operators are described and explored and found to be suited to different classes of problem. We compare these several instantiations of EWD with other encodings (including the previous best reported for each problem) over several benchmark instances, as well as randomly generated instances. We find that instantiations of EWD generally perform favorably, clearly outperforming the best comparative approach in two of the three problem classes, and close to best in the third. Some analyses of EWD in terms of its locality, heritability, and bias are provided. These indicate that EWD shows high locality and heritability, and an intermediate level of bias toward the MST of the problem under consideration. In particular, the basic EWD encoding strategy is unbiased, but its good performance on a range of tree-based problems appears to be explainable in terms of the bias inherent in the associated operators. Further, experiments show that EWD strategies using one particular biased operator are able to apply a consistent intermediate level of bias "pressure" over time, thus allowing the search to range more freely in the early stages over the less represented areas of the landscape, rather than (as is the case with the other biased encoding/operator combinations compared against) quickly becoming committed to MST-like regions.**

***Index Terms*—Degree constrained minimum spanning tree problem, evolutionary algorithm (EA), optimum communication spanning tree problem (OCST), quadratic minimum spanning tree problem (q-MST), tree representation.**

Manuscript received December 10, 2004; revised October 18, 2005. This work was supported in part by the Brain Korea 21 Project and in part by the Korea Science and Engineering Foundation (KOSEF).

S.-M. Soak is with the Gwangju Institute of Science and Technology (GIST), Gwangju 500-712, Korea (e-mail: soakbong@hotmail.com).

D. W. Corne is with Heriot-Watt University, Edinburgh EH14 4AS, U.K. (e-mail: dwcome@macs.hw.ac.uk).

B.-H. Ahn is with the Department of Mechatronics, Gwangju Institute of Science and Technology, Gwangju, Korea 500-712 (e-mail: bayhay@gist.ac.kr).

Digital Object Identifier 10.1109/TEVC.2006.871250

I. Introduction

IN NETWORK-ORIENTED optimization problems, such as those involving transportation networks, local area networks (LANs), communication networks, and optical networks, the design of the network topology is one of the most important issues, having significant impact on the network's performance. In the context of this topology issue, the minimum spanning tree (MST) problem has often been used as the basic model for constructing a backbone of the network under consideration, making use of algorithms developed to search efficiently for the MST. Such algorithms for finding MSTs are well-known, the most common being Prim's algorithm [35] and Kruskal's algorithm [24]. Both can construct an MST of a network in polynomial time. Generally, an MST is identified by repeatedly using the shortest remaining edges to gradually build a tree, while ensuring that no cycle is introduced.

However, in realistic network topology design problems, various constraints apply, such as a degree constraints on each node or specific user traffic requirements; these are imposed over the underlying MST problem, changing it into a much more complex challenge for algorithmic search. Most of these are NP-hard variants of the MST problem, generally classed as "constrained" spanning tree problems [8], [14]. Among these, the degree constrained minimum spanning tree problem (DCMST), optimum communication spanning tree problem (OCST) and quadratic minimum spanning tree problem (q-MST) are dealt with in this paper. Each of these has a different objective function and set of constraints, arising from the nature of the classes of applications which motivate them.

Generally, finding constrained MSTs in a graph is often of importance in communication network design and other network-related problems [12]. Several approaches have been studied over the years. These include exact approaches (usually based on a branch and bound algorithm [27]), approximation algorithms [32], [54], [55], and heuristic approaches [3], [5], [27], [56]. In more recent years, evolutionary algorithms (EAs) have been investigated very much in this context. It turns out that comparative analysis of different approximation or heuristic algorithms has seldom been done comprehensively in the constrained MST problem literature, however, in the recent EA-oriented literature, it is found that an EA can regularly find the best known solutions on the standard benchmarks. In particular, on DCMST problems, an EA using the *edge-sets* (ES) encoding [38], [41] can regularly find the best known solutions on the standard benchmarks. On the other hand, EAs using the *link and node biased* encoding (LNB) [28], [29] and the *network random key* encoding (NetKey) [40] have been found similarly

appropriate and effective (together with the ES) encoding [41], [42]) for the OCST problem, although they have not so far been compared with each other directly on this problem. Meanwhile, in the case of the q-MST problem, it has been found that an EA using the *Prüfer number* encoding [57], [58] was superior to the previously applied heuristics [3], [56]. However, the often-noted poor locality of the Prüfer encoding leads us to suspect that other approaches, such as those mentioned, would outperform it on the q-MST, but such experiments have not yet been published.

Given the vigorous recent history and success of EA approaches to constrained spanning tree problems, we focus on an EA approach in this paper, and in particular, we develop a new flexible encoding strategy which combines a high level of fitness bias with a high level of exploration, and which can be modified using domain knowledge to increase its suitability to particular classes of problems. The general issue of *encoding* is of course one of the critical steps in designing an EA approach to any application area, and the progress so far in the area of EAs applied to spanning tree problems can be seen as the continual development of ingenious new encodings (and associated genetic operators) [1], [7], [14], [19]–[21], [23], [34], [38], [40], [45], [58].

In general, we can see all of these attempts as finding a way to strike a balance between locality (the degree to which small perturbations in genotype affect perturbations in phenotype), heritability (related to locality—the degree to which the encoding allows children to usefully inherit good feature combinations from the parents), and feasibility handling (the degree to which feasibility is guaranteed in the encoding rather than imposed by repair operations). In most cases, encodings so far investigated have low locality, require ad hoc or expensive additional operations to ensure feasibility, and face problems in incorporating problem specific knowledge into the approach. In this paper, we describe a new encoding strategy for tree-based problems which appears to strike a good balance among these issues, and turns out to perform well.

The main idea is to encode a tree as a list of nodes, which is then interpreted by a tree construction routine (TCR), which builds a tree from this list. Meanwhile, operators can be freely designed to work at the genotype (list of nodes) level. A general semiformal description of the strategy, which we call the edge-window decoder (EWD) strategy is as follows. An EWD strategy is a triplet (Gt, TCR, Ops), in which:

- Gt specifies the genotype space (e.g., what are valid and invalid lists of nodes, required length of the list, and other such issues);
- TCR specifies the tree construction routine, which must build a valid tree given an input genotype fitting the Gt specification;
- Op is simply a set of mutation and crossover operators designed to work on genotypes, producing children which must fit the Gt specification.

Clearly, the design of Gt and TCR must take into account their close interaction, and different kinds of TCR will impose different requirements on the required inputs. This will become more clear as we discuss TCRs later. Meanwhile, Op only depends on Gt, and is otherwise independent of the TCR. This effectively provides two distinct ways in which we can impose domain-specific tailoring in a given application. That is, we can design specialized operators which appropriately bias the contents of the genotype, and we can design specialized TCRs which appropriately influence the structure of the trees. We refer to an application of this strategy, with a specific Gt, TCR, and Op, as an *instantiation* of the strategy.

As we will see, the instantiations of EWD investigated here all lead to redundant encodings, as is also the case with (for example) LNB representation and NetKey representation, in the sense that the number of genotypes exceeds the number of phenotypes [42], [43]; i.e., several genotypes will encode the same phenotype. It has been demonstrated in some cases that redundant representations appear to increase the evolvability of EAs and increase the performance of evolutionary search [9], [48]. However, recently Knowles and Watson [22] showed that redundant representations do not necessarily increase an EA's performance; moreover, in most of the tested problems (NK-landscapes, H-IFF, and MAX-SAT) redundant representations appeared to reduce performance. Also, Rothlauf [43] analyzed a property of a redundant representation (the redundant trivial voting mapping), concluding that *uniformly* redundant representations generally do not increase an EA's performance. "Uniformly redundant" indicates a redundant encoding that has no fitness bias. As we will see, we find that certain instantiations of EWD improve over the performance of previous approaches, providing evidence on the positive side for redundancy in this debate, but we will also see that this performance improvement also correlates with the presence of appreciable fitness bias (i.e., the EWD encoding is not uniform).

In the paper, we prove certain properties of the simplest EWD instantiation which are not immediately obvious, e.g., as long as certain properties are satisfied by a basic TCR, the length of genotype string necessary for reliably generating a tree over an N-node graph is $2(N-1)$, and every spanning tree over N nodes can be generated by such a string.

The rest of this paper is organized as follows. The various spanning tree problems which are going to be dealt with in this paper are introduced briefly in Section II. The prior representation methods are described in Section III. The new strategy, and TCRs and operators which we use in instantiations of the strategy, are described in Sections IV and V. The experimental results are described in Section VI. Section VII then offers an analysis of the relative performance of EWD instantiations in terms of locality, heritability, and fitness bias characteristics. Finally, the paper concludes with a summary of the results in Section VIII.

II. Constrained Spanning Tree Problems

In the MST problem, the edges (or links) have associated weights that could be based on their distance, capacity, quality of line, etc. Various constraints such as the number of nodes in a sub tree, degree constraints on nodes, flow, and capacity constraints on any edge or node, and types of services available on an edge or node, can be imposed on the network design. Therefore, there are many variants of the MST in relation to different classes of constraints [8], [14]. Of these variants, we are going to deal with the DCMST problem, OCST problem, and q-MST problem. These are now briefly described.

A. Degree Constrained Minimum Spanning Tree (DCMST) Problem

Research on the DCMST problem seems to have begun with Narula and Ho [27]. It is an extended formulation of the MST problem. Consider an undirected complete graph $G = (V, E)$, where $V = \{1, 2, \ldots, N\}$ is the set of N nodes and $E = \{1, 2, \ldots, M\}$ is the set of M edges with given distance (or weight). The DCMST on a graph is the problem of generating a spanning tree T with minimum cost satisfying with degree constraints on the number of edges that can be incident to nodes of the graph.

More formally, the DCMST can be described as follows:

$$\text{Min} f(x) = \left[\sum_{i=1}^{M} x_i \cdot C_i \,|\, d_j \leq b_j, x \in T \right] \tag{1}$$

where d_j is the degree of node j and b_j is the degree constraint on node j and C_i is the weight of an edge i. x_i indicates the decision variable and if an edge e_i is included in a tree T, it is 1, otherwise, 0.

In this paper, we consider only problems where the degree constraints on all nodes in the complete graph are the same, i.e., the *uniform degree constraint problem*. This has generally been the case in the research literature so far on these problems, although our encoding approach (and many of the others cited) can easily handle different individual degree constraints for each node.

B. Optimum Communication Spanning Tree (OCST) Problem

The OCST problem is a variant of the MST problem, first introduced by Hu [17]. The problem—proved as NP-hard [12], and further shown to be MAX SNP-hard [31],[1] is to find a minimal cost spanning tree which satisfies given communication requirements between pairs of nodes. Some researchers have proposed approximation algorithms for solving this problem [32], [54], [55]. Recently, Wu *et al.* [54], [55] investigated the OCST problem in more detail, and introduced two variants of OCST problems and new approximation algorithms which obtain a solution in $O(|N^3|)$ time.

As with the DCMST problem, consider an undirected complete graph $G = (V, E)$. In the case of the OCST problem, instead of degree constraints, there are communication requirements associated with each pair of nodes, specified by $R(i, j)$, e.g., these may represent the number of telephone calls between the two cities. For any spanning tree T of G, the communication cost between two nodes is defined to be the communication requirement multiplied by the distance between the two nodes on T; the communication cost of T is the total communication cost summed over all pairs of nodes.

The goal is to construct a spanning tree with minimum communication cost. It can be formulated as in (5), where $C_T(i, j)$ sums the weight along edges between i and j on T. Therefore, the pair i and j may be a single edge, or a path along edges, from i to j

$$\text{Min} f(x) = \left[\sum_{i,j \in V} R(i, j) C_T(i, j) \right]. \tag{2}$$

[1]Arora *et al.* [2] proved that no MAX SNP-hard problem has a polynomial time approximation scheme, unless NP = P.

C. Quadratic Minimum Spanning Tree (q-MST) Problem

The q-MST problem, which is also a new formulation of the conventional MST problem, was described by Xu [3], [56]. It involves searching for a spanning tree of minimum cost in circumstances where this cost is a quadratic function of weights of certain pairs and groups of edges. The q-MST problem had been proven NP-hard by Xu [3], [56]. The q-MST problem is formulated as follows:

$$\text{Min} f(x) = \left[\sum_{i=1}^{M} \sum_{j=1, j \neq i}^{M} a_{ij} x_i x_j + \sum_{i=1}^{M} b_i x_i \,|\, x \in T \right] \tag{3}$$

where a_{ij} is the "intercost" between edge e_i and edge e_j and b_i is cost (or distance). x_i is a binary decision variable. In contrast to the DCMST and OCST problems, only a small number of articles concerning the q-MST appear in the research literature.

However, the q-MST is arguably as important and relevant as are the DCMST and OCST in real-world applications. The problem arises, for example, when transmitting oil from one pipe to another or transmitting information from one line to another; in such cases the cost may depend on the nature of the interface between two pipes or that between two transmitting lines. The same pairwise interaction effect arises in the connection of aboveground and underground cables in a transportation or road network with turn penalties. In all these cases, the presence of intercosts results in a minimum spanning tree with quadratic (rather than linear) costs [58].

Assad and Xu [3], [56] proposed three heuristic methods for solving the q-MST and they applied them to small instances ($N = 6$ to $N = 15$). Zhou and Gen [57], [58] showed that the EA using the Prüfer number representation is superior to these heuristics (on sizes ranging from $N = 6$ to $N = 50$). But, since the Prüfer number representation is known to have poor locality [15], [28], [29], it has generally shown the worst results when compared with other encoding methods we have mentioned (and further describe later). We therefore expect improvements in the best-so-far results on the q-MST by applying current tree-based encodings, while the q-MST also serves as an additional test bed for the comparative evaluation of our new encoding.

III. Spanning Tree Representations

We later compare our new encoding (described in the next section) with the best-performing previously published encodings for tree-based problems. In this section, we give the key details of the comparative encodings.

The Prüfer Number Representation: By Cayley's theorem, there are N^{N-2} distinct labeled trees on a complete graph with N nodes. Each such tree can be represented by a unique string of $L = N - 2$ digits where each digit is an integer between 1 and N [6], [14], [23], [58]. This string is called the Prüfer number; encoding a spanning tree by its Prüfer number provides a nonredundant mapping from genotype to phenotype, covering all possible trees. However, this technique turns out to have poor locality characteristics, which is reflected in comparatively poor performance in spanning tree optimization when compared to the other methods described later in this section [15], [38]. We nevertheless use the Prüfer encoding as a comparative technique

in our experiments, and consider it as providing a baseline measure. Other encodings have been developed that share the Prüfer encoding's nonredundancy but have better locality [33]. However, we choose Prüfer here since, prior to this article, it provides the best previously published results for the q-MST problem.

The Prüfer number representation scheme involves processes for generating a spanning tree from a Prüfer number, and *vice versa*. We refer the reader to [58] for full details of these routines. We will also mention, however, that Prüfer numbers can be encoded and decoded in $O(N \log N)$ time. Also, since every Prüfer number represents a unique spanning tree, specialized crossover and mutation operations need not be designed. But as pointed out in much recent literature [15], [38] this encoding has poor performance despite its lack of redundancy.

A. Link and Node Biased (LNB) Representation

The link and node biased (LNB) representation was proposed by Palmer and Kershenbaum [28], [29] and a version of it was applied to the OCST problem. This method represents the structure of a tree network using a weighted vector and allows EAs to distinguish between the importance of the nodes and links in the network. In this representation, the individual holds a bias value for each node and each link. Each node bias and each link bias is in the range $[0, 1]$; in our experiments, we use only 8-bit precision, so there are 256 possible bias values. The spanning tree corresponding to the LNB code is found by running Prim's algorithm on the following modified cost function C':

$$C'_{(i,j)} = C_{(i,j)} + P_1 b_{(i,j)} C_{\max} + P_2 (b_i + b_j) C_{\max} \qquad (4)$$

where $C_{\max}$ is the maximum link cost in the graph, $b_{(i,j)}$ is the link bias associated with the edge from node i to node j, and b_i is the node bias associated with node i. The node and link biases are normalized to map to the range $[0, 1]$ before using them in the equation above. P_1 and P_2 are control parameters. Actually, although the authors describe the encoding in the above generic way to account for both link and node biases, their experiments generally found that only node biases were necessary to achieve good performance. That is, Palmer and Kershenbaum used $P_1 = 0$ and $P_2 = 1$ in their reported OCST experiments. Also, Palmer's dissertation [30] reports on experiments with a "meta-GA" that attempted to find good values in the range $[0, 1]$ for P_1 and P_2 (among other parameters), and these led to a negligible value for P_1 (0.044), compared to 0.95 for P_2. We, therefore, use $P_1 = 0$ and $P_2 = 1$ and, hence, effectively use a "node biased"' encoding. Palmer and Kershenbaum, along with others (e.g., [1]) tend to refer to this encoding as LNB, despite only using the "NB" version. From here onwards, we will generally refer to the encoding as NB, to emphasize that all experiments with LNB in this paper refer to the version with $P_1 = 0$, hence involving no manipulation of link biases. It is worth noting that this version has more favorable time complexity [41], however, it is also worth noting the finding that the version with $P_1 = 0$ is biased toward star topologies, and that use of the link-bias (i.e., setting P_1 above 0) will reduce this bias [13]. This is likely to improve performance on problems where the better solutions are far from a star topology, and the relative performance of LNB in such cases may or may not make up for the increased time complexity involved when dealing with link biases.

B. Network Random Keys (NetKeys) Representation

The NetKeys representation [40], [45] is an efficient encoding method which is generally applicable to problems involving orderings and permutations. NetKeys are based on random keys, which in turn were introduced by Bean [4] for solving combinatorial optimization problems such as machine scheduling, resource allocation, and quadratic assignment. Rothlauf *et al.* [40], then extended the approach to solve a real-world OCST problem. The basic idea is to encode an ordering of m items by using a string of m arbitrary numbers, $x_1, x_2, \ldots, x_m$. The string simply encodes an ordering as follows. If x_i moves to position j in the string when it is sorted in ascending order, then item i is in position j of the decoded ordering.

The NetKey representation, as applied by Rothlauf to spanning tree problems, is similar in essence to Kruskal's well-known MST algorithm [24]. Kruskal's algorithm involves first sorting all edges based on their weight, and then gradually building the tree edge by edge based on this ordering. In the Netkeys approach, the only essential difference is that the ordering of edges is encoded by a candidate solution, and in the way described above, rather than based on a deterministic sort. The NetKeys method shares a drawback with Kruskal's algorithm, which is that computational cost becomes unacceptable in large instances with highly connected graphs (this is also true of the LNB encoding, when both link and node biases are involved). For example, in the case of a complete network with 100 nodes, a string of length $4950 = 100(100-1)/2$ is required to encode each solution (one gene per edge). Meanwhile, $O(L \log N)$ time is needed to decode the candidate solution. On the other hand, as with the Prüfer encoding, no special genetic operators are required. However, when designing operators, the NetKeys method cannot easily incorporate network information or constraints. It also has a high level of redundancy.

C. Edge Set (ES) Representation

Raidl *et al.* [36], [38] proposed the ESs representation, and have applied this representation to the DCMST and shown it to be superior to other representations on this problem. This approach represents trees in a direct way (as a set of edges) and makes use of Prim's algorithm [35] and Kruskal's algorithm [24] in initialization and operators. Specialized operators such as a heuristic initialization operator, edge crossover, and edge insertion mutation are used, to ensure the generation of feasible new solutions (i.e., sets of edges which contain all nodes, are connected, and involve no cycles). An improved version of the method involves *heuristic edge crossover* and *heuristic edge insertion mutation*. We later refer to ES with these heuristic operators as ES with heuristics (ESWH), and we refer to the version with "straightforward" edge crossover and edge insertion mutation as ES without heuristics (ESWOH).

Rothlauf *et al.* [41] analyzed the properties of the OCST problem and showed that the optimal solution of the OCST problem is biased toward the MST. That is, there is generally a good correlation between the similarity of an arbitrary tree to the (unconstrained) MST and its distance from the OCST. Hence, performance on the OCST can be usefully aided by incorporating methods developed with the MST in mind. This

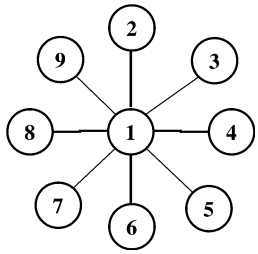

Fig. 1. Star type spanning tree.

in turn suggests that the ES representation should perform well on the OCST problem, owing to its use (in initialization and operators) of Prim's and Kruskal's algorithms.

However, Tzoppe *et al.* [52] showed that ESWH is not ideal for solving OCST problems because the heuristic crossover method is too strongly biased toward the simple MST. But, as they indicated, if the optimal solution of an OCST problem happens to coincide with the simple MST, then ESWH can easily solve that problem. If not, ESWOH turns out to be more effective. Of course, in a realistic case, users are not aware of how close the solution to their OCST is to the MST, and so it is difficult to predict in advance which of the two to use.

IV. EWD Encoding Strategy

Our new encoding hybridizes a straightforward and direct encoding of a subgraph with a tree-construction algorithm. Consider the tree in Fig. 1. This consists of the "edge set": $\{\{1,2\},\{1,3\},\{1,4\},\{1,5\},\{1,6\},\{1,7\},\{1,8\},\{1,9\}\}$. Now, considering Fig. 1, imagine tracing a line from node 1 to node 9 by following edges, visiting every node, without leaving the paper. This will result in a string of nodes visited in order such as (1, 2, 1, 3, 1, 4, 1, 5, 1, 6, 1, 7, 1, 8, 1, 9).

The encoding (which was inspired by this tracing process) goes in the other direction, by generating a tree from such a string. We illustrate it with this same string, (1, 2, 1, 3, 1, 4, 1, 5, 1, 6, 1, 7, 1, 8, 1, 9). We start by initializing a subgraph (a set of edges) as empty. Now, move a window of length two along the string, one step at a time. The window first sees "1, 2," so we add this edge to our developing subgraph. The window then sees "2, 1," then "1, 3," and so on. We successively consider the edges indicated by each window, and include them or not according to the particular TCR in place (see later this section). In this paper, we only consider undirected graphs, so (1, 2) is equivalent to (2, 1). Also, any window containing copies of the same node identifier is ignored. Hence, in the previous example, when "2, 1" is seen, it will not be included in the growing subgraph, since it is already present. Also, if "3, 3" were in the string, it would be passed over without effect, and the next window (starting with "3") would be considered.

In more general and formal terms, our encoding works with a string of L node identifiers $(i_1, i_2, \ldots, i_L)$, where all $i_j \in 1, \ldots, N$. Note that in general we may have $i_j = i_k$ for distinct j and k. Abstractly, this string is interpreted in two steps. First, we extract an ordered set of unique edges from this string, where every edge is of the form $\{a, b\}$ with a distinct from b. Second, we use a TCR to generate a spanning tree from these edges. A TCR may or may not work with the given ordering of edges, and the two steps may be interleaved for convenience. Herein, the first step always proceeds in the same way, as illustrated by the example above, retaining only the unique edges. Note that this set of edges (i.e., before we apply a TCR) is in general a subgraph (it may contain cycles).

We call our encoding EWD, to stand for "edge-window-decoder." In the following, when discussing our encoding, we will often call it EWD, whichever choice of TCR is being used (we start to further describe TCRs below). If it is necessary or desirable in context, we sometimes refer to the new encoding by the name of the TCR in use (e.g., CB-TCR).

We now need to clarify issues concerning the instantiation of Gt in an EWD strategy. As indicated above, Gt gives specifications for the list of node identifiers allowed as input to the TCR. An example of an instantiation of Gt would be to specify that each list of node identifiers be a permutation of the N nodes (i.e., all strings which contain every node precisely once). Another might be simply to specify that the any list of numbers (up to some specified maximum length) between 1 and N is acceptable. The former specification would clearly lead to an impoverished set of trees, while the latter could be the basis for an encoding of arbitrary multigraphs.

In this paper, we are specifically interested in specifying Gt in such a way that, in combination with a simple "default" TCR, we can guarantee that all spanning trees over N nodes are encoded, and that every input string encodes a spanning tree. It turns out that this can be done with certain straightforward constraints on Gt. It will be useful to first briefly discuss TCRs, and present the most straightforward of our TCRs, called the *cycle free TCR* (CF-TCR), and then we will return to this question.

A. TCRs and CF-TCR

The role of the TCR is to consider the set of edges obtained from the string of node identifiers, and produce a spanning tree. Many TCRs are possible, and it is useful in what follows to consider various properties that TCRs may or may not have. First, a TCR may or may not treat the set of obtained edges as an ordered list (i.e., the same ordering of edges as obtained from the individual). We will call a TCR *order preserving* if this applies. TCRs will typically develop a tree by making a number of successive distinct ESs, $E_1, \ldots, E_m$, where each $E_k, 2 \leq k \leq m$ contains precisely 0 or 1 more edges than E_{m-1}, and E_m will contain $N-1$ edges (i.e., it will be a spanning tree). It is safe to say that in all usable TCRs each of these edge sets will not contain a cycle, but in one TCR we consider they will not necessarily all be trees (although of course E_{N-1} should always be a tree). We will call a TCR *node-preserving* if the set of *nodes* contained in E_k is always a (nonstrict) subset of those contained in E_{k+1}, for all $1 \leq k < m$. Finally, we call an order-preserving TCR *greedy* if it will *always* include the next edge when it is feasible to do so without violating any constraints.

We now describe CF-TCR, a specific and simple example of a TCR. Derived from Prim's algorithm [35], CF-TCR builds a tree by continually adding edges which link a new node to an existing selected node in a growing sub tree, ensuring there are no cycles, and continuing until every node is connected. To illustrate, consider the individual (1, 3, 2, 1, 4, 3, 5, 5). First, set $\mathrm{SelNode}\{\phi\}$ and $\mathrm{EdgeSet}\{\phi\}$, where these are the

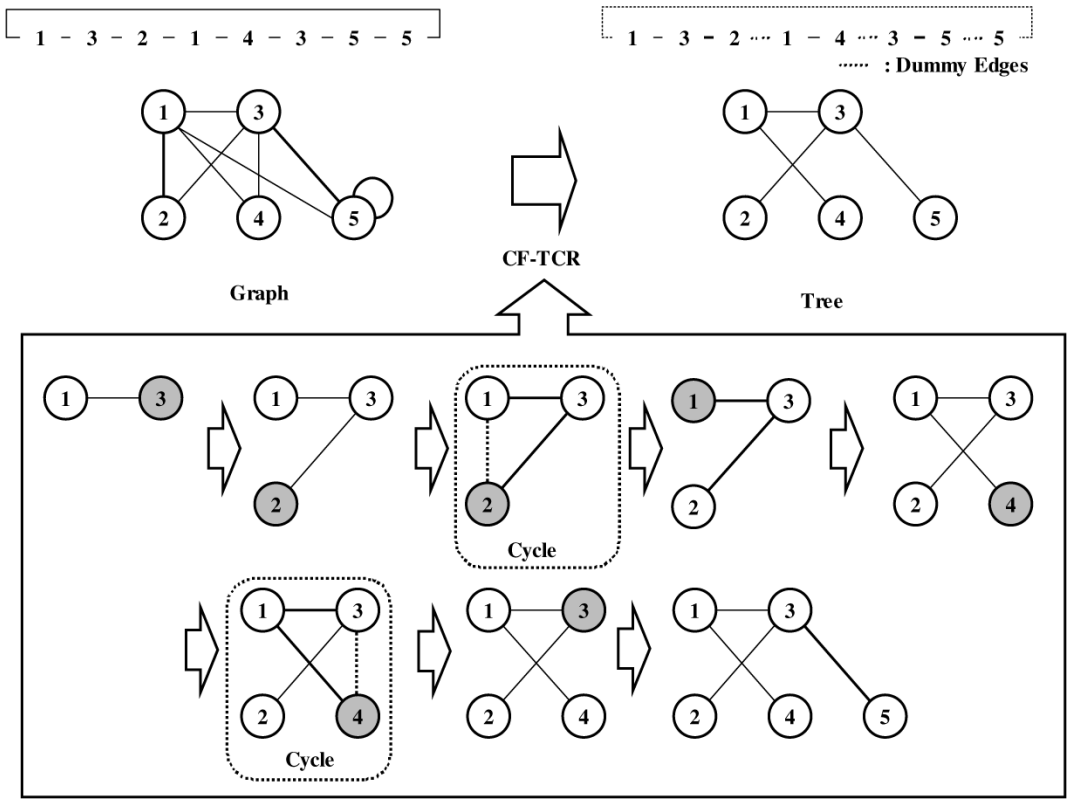

Fig. 2. A process generating from a graph to a spanning tree using CF-TCR.

sets of selected nodes and selected edges, respectively. We start with the leftmost two genes, $\{1,3\}$, which populate the node and edge sets to leave us with: $\mathrm{EdgeSet}\{\{1,3\}\}$ and $\mathrm{SelNode}\{1,3\}$. Moving our two-node window left to right, the next *new* edge to consider is $\{3,2\}$. Node 2 has not been selected, so use this edge, giving $\mathrm{EdgeSet}\{\{1,3\},\{3,2\}\}$ and $\mathrm{SelNode}\{1,3,2\}$. Next, we consider $\{2,1\}$. Node 1 is selected at the previous step. So, this edge is skipped. Next, we consider $\{1,4\}$. Node 4 is not yet selected, so we use this edge, giving us: $\mathrm{EdgeSet}\{\{1,3\},\{3,2\},\{1,4\}\}$ and $\mathrm{SelNode}\{1,3,2,4\}$. Next, we consider other 2 nodes, node 4 and node 3. Node 3 was already selected. So, this edge is also skipped. Finally, we check edge $\{3,5\}$. Node 5 has not been selected, so we connect nodes 3 and 5, and the final result is $\mathrm{EdgeSet}\{\{1,3\},\{3,2\},\{1,4\},\{3,5\}\}$ and $\mathrm{SelNode}\{1,3,2,4,5\}$. CF-TCR terminates now since the number of selected edges is $N-1$. Note that CF-TCR is both order-preserving, node-preserving, and greedy.

Fig. 2 shows by example how CF-TCR generates a spanning tree from a subgraph.

It turns out that, when the TCR in use has certain properties, in order to encode all possible spanning trees of an N node graph a string of node identifiers needs to be of length $2(N-1)$. We prove this in the following.

Theorem 1: When an order-preserving, node-preserving, and greedy TCR is used, any spanning tree on $N \geq 2$ nodes can be encoded by a sequence of at most $2(N-1)$ node identifiers.

Proof: We prove this by induction on N. First, consider the case $N=2$. There is only one possible string which satisfies the given constraints, and that is: (1, 2). This single edge is also the only possible spanning tree on a two-node graph. We now need to show the inductive case; that is, to prove that if the theorem is true for $N=k$ then it is also true for $N=k+1$.

Assuming the theorem is true for $N=k$, we first show that any spanning tree on $k+1$ nodes can be encoded by a string of at most $2((k+1)-1)$ node identifiers. Consider any such tree. We can remove an arbitrary leaf, leaving a spanning tree on k nodes, which we will call T_k, and an unconnected node u, which had previously been connected to T_k at c. By assumption, we know that T_k can be encoded by a string of maximal length $2(k-1)$. Assume "$S_k=(n_1,\ldots,n_L)$" is such a string, encoding T_k, where $L \leq 2(k-1)$. Now, consider the string S_{k+1} formed by adding (c,u) to the end of S_k. When we interpret this using an order-preserving TCR, then (by assumption) after we have considered the node-pair "(n_{L-1},n_L)," we have already built the tree T_k. Next, we consider the node-pair "(n_L,c)"; both of these nodes are already in the partial tree already built by the TCR; being node-preserving by assumption, it cannot add this edge. The TCR will, therefore, ignore this and move on. Finally, the (order-preserving) TCR will consider the node-pair (c,u); since u is not in SelNode at this point, the inclusion of this edge is feasible, and this (greedy) TCR will, therefore, incorporate (c,u), producing our original "arbitrary" tree on $k+1$ nodes. Also, the length of S_{k+1} is $L+2$, and clearly $(L+2) <= (2(k-1)+2) <= 2((k+1)-1)$. By induction, it follows that the statement of Theorem 1 is true for all $N \geq 2$.

We have yet to prove that every string that satisfies this length constraint actually encodes a valid spanning tree. The next theorem covers this.

Theorem 2: When an order-preserving, node-preserving, and greedy TCR is used, every sequence which contains every node identifier at least once, and is no longer than $2(N-1)$, encodes a valid spanning tree on $N \geq 2$ nodes.

Proof: We prove this by appeal to the assumed TCR properties. As a greedy TCR proceeds, it considers edges $\{n_1, n_2\}$, $\{n_2, n_3\}$, and so on, up to $\{n_{L-1}, n_L\}$, where $N \leq L \leq 2(N-1)$, building SelNode, which contains all nodes contained so far in the developing tree. Consider the first time a given node n appears in the string. If we have $n = n_1$ or $n = n_2$, then n will be seen in the first iteration and immediately become part of the developing tree. Otherwise, and in general, we will first see n in a window (n_i, n) for some i, with $1 \leq i < L$. There are two cases: either n_i is already in SelNode, or it is *not*. In the first case, a greedy TCR will clearly add $\{n_i, n\}$ to the developing tree. The second case can only happen in the first iteration of an order-preserving, greedy TCR, which we have already covered. So, node n will always be added to the tree. It follows that every node in the string must also appear in the tree. Given that every TCR guarantees to maintain a tree with no cycles, it is clear that any string satisfying the given constraints will encode a valid spanning tree on N nodes.

Since CF-TCR is order-preserving, node-preserving, and greedy, it follows that when CF-TCR is used, the encoding (given the $2(N-1)$ length constraint) covers all possible spanning trees over N nodes, and always generates a valid spanning tree.

B. Natural Handling of Degree Constraints With Suitable TCRs

Also, it turns out that we can naturally and easily handle degree-constraints by simply ensuring that, if a node has a maximum allowed degree of d, then that node appears at most $d-1$ times in the sequence. It is straightforward to see that this works by appealing to the operation of a node-preserving, order-preserving, and greedy TCR. We will show this informally as follows. As already shown, every node n in the linear string will appear at least once in the spanning tree; it first appears in the tree, with degree 1, the first time it is encountered in the string. In each subsequent appearance of n in a new window, there will clearly be either 0 or 1 edges added to the tree which link to n. However, we must note that where the n appears in the context "$\ldots, p, n, q, \ldots$," it will appear in two subsequent windows and, hence, this one appearance may lead to two edges. However, beyond the first appearance of n, and since the TCR is node-preserving, p in this context will already be in the tree, and the "p, n" window will therefore not introduce a new edge, since this would introduce a cycle. Of the $d-1$ appearances of n, $d-2$ of them may *each* lead to a maximum of 1 new edge (i.e., these could lead to a total of $d-2$ new edges), while the first appearance only may lead to two edges—hence, the maximum degree of n will be d.

The EWD encoding (with Gt instantiated in harmony with the above proofs, and when an order-preserving, node-preserving, and greedy TCR is in use) therefore has a very convenient property when it comes to incorporating degree constraints, simply by placing a maximum on the number of times a node can appear in the string; note that this is straightforwardly achieved even when different nodes have individual degree constraints.

C. Degree Constrained Kruskal Tree Construction Routine (dK-TCR)

Kruskal's algorithm [24] works by first sorting all edges of a graph in ascending order of weight, and then building a spanning tree by including edges (following this ordering), maintaining a set of partial trees as it proceeds, discarding any edges which introduce a cycle. Kruskal's algorithm is a common inspiration in recent encodings for tree problems [36], [38], [40], [45].

In dK-TCR, following step 1 (extraction of the edges encoded in the string of node identifiers), we apply a version of Kruskal's algorithm to the resulting subgraph as follows. First, dK-TCR enumerates all edges in the encoded subgraph, and then sorts these in ascending order of weight. It then repeatedly adds the shortest edge in the subgraph to a set of growing partial trees, so long as this shortest edge does not result in a cycle or a degree constraint violation. When $N-1$ edges are gathered, the process is terminated. Fig. 3 represents the pseudocode of dK-TCR.

With dK-TCR, as with CF-TCR, every individual is a string of $2(N-1)$ node identifiers, with each node appearing at least once, and at most $d-1$ times. However, dK-TCR is not order preserving. This affects the properties of the encoding when it is used. Although the encoded edges "contain" at least one valid tree (as interpretable, for example, by CF-TCR), a "clean" version of dK-TCR (which would terminate after the first `while` loop in Fig. 3) does not guarantee to obtain one. There are a variety of candidates for ways to address this, ranging from introducing a backtracking process (which would be unreasonably expensive) through to, for example, defaulting to use of CF-TCR if the first pass fails to complete a spanning tree. In this paper, we simply choose a random valid edge when necessary (the second `while` loop in the pseudocode), and ensure that the choice is recorded if necessary to recover a best-so-far solution. In preliminary experiments, we found that this random choice occurs less than 0.1 times per interpretation of an individual in the population. We are exploring the alternative approaches in ongoing work, but meanwhile the low incidence seems to us to be acceptable.

D. Cycle-Breaking Tree Construction Routine (CB-TCR)

Finally, we consider a CB-TCR. This is very similar to CF-TCR, but introduces a bias toward the lower weight edges in the encoded subgraph. CB-TCR works in every way like CF-TCR, except that when the next edge under consideration would introduce a cycle, it may include that edge (and remove an existing one, hence breaking that cycle).

CB-TCR successively includes edges from the ordered list until an edge is included which introduces a cycle in the graph. At this point, the cycle-inducing edge is temporarily included, and we then consider the complete set of edges involved in this cycle, C. The highest weight edge in C is then removed. If there is a tie for the highest weight edge, then this tie is broken randomly. (In the case of OCST, if the highest weight edge is a tie, CB-TCR considers the communications requirements; if this is also a tie, then it is broken randomly.) We then carry on to consider the next edge in the encoded sequence, and the process continues until $N-1$ edges are gathered.

Fig. 4 illustrates CB-TCR in a case in which two cycles are successively introduced and broken: $\{1, 3\}, \{3, 2\}, \{2, 1\}$ and

Begin
Let E be the set of edges extracted from the string of node identifiers.
Sort E into ascending order of weight
Set $T = \phi$ (this will grow to be the set of edges in the resulting tree)
Set $C(k) = 0, \forall k \in N$; ($C(k)$ keeps track of the degree of node k in T)
Set $i = 0$ (this counts the number of edges so far added to T)
while E is non-empty **do**
  choose the shortest weight edge $(j,\ t) \in E$ (the leftmost edge in the sorted list);
  if adding $(j,\ t)$ to T wouldn't introduce a cycle, and both $C(j) and C(t) < d$ **then**
    $T \leftarrow T \cup (j,\ t)$; (add this edge to T)
    $C(j) \leftarrow C(j) + 1\ and\ C(t) \leftarrow C(t) + 1$; (update the degrees of j and t)
    $i \leftarrow i + 1$; (update the number of edges in T)
  $E \leftarrow E - (j,\ t)$; (finally, remove this edge from E)
  (At this point, all edges have been removed from E,
  but some may not have been included in T.)
  (the following loop complete T with randomly chosen edges.)
while $i \neq N - 1$ **do** (when i reaches $N - 1$, we know that T will be a spanning tree)
    Randomly select edges until one is found, $(u,\ v)$, which wouldn't introduce a cycle
      and where $C(u)\ and\ C(v) < d$;
    $T \leftarrow T \cup (u,\ v)$; (include this edge in T)
    $C(u) \leftarrow C(u) + 1\ and\ C(v) \leftarrow C(v) + 1$; (update the node degrees in T)
    $i \leftarrow i + 1$; (update the number of edges in T)
end

Fig. 3. Pseudocode for the d-Kruskal TCR.

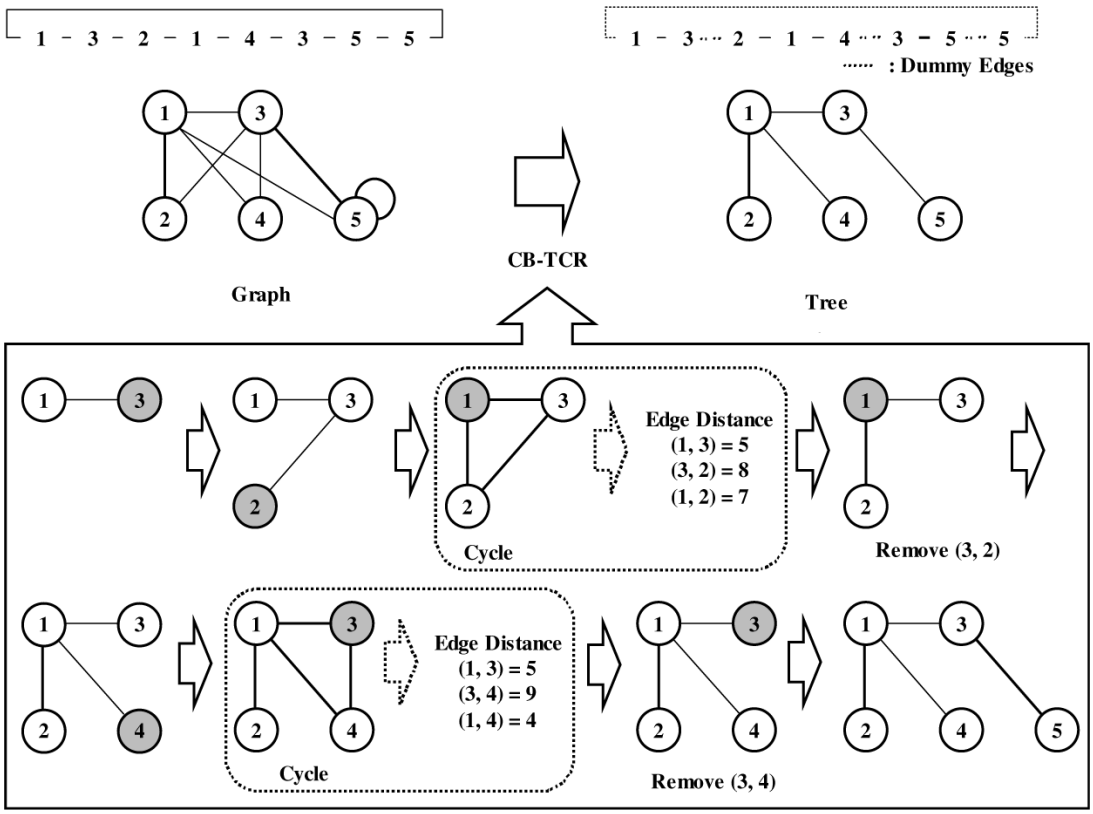

Fig. 4. Generating a spanning tree from a sequence of edges using CB-TCR.

$\{1,3\}, \{1,4\}, \{4,3\}$. In the first cycle, the edge $\{3,2\}$ is removed through the weight comparison process and in the second cycle, the edge $\{4,3\}$ is removed.

Pseudocode for CB-TCR is given in Fig. 5. In this figure, and in all of the following we will simplify the presentation by taking advantage of the fact that we consider only undirected and loop-free graphs in this paper. Further, in the context of Fig. 5, if $\{a,a\} \in E$, then this edge will be ignored. We now explain CB-TCR in detail. In the Fig. 5, lines 1 and 2 initialize the process, where E is the set of edges initially decoded from the string, k is a counter (which indexes the edge from E currently under consideration), T is the set of edges in the (gradually developing) tree, and U is the set of nodes so far *unvisited*. The algorithm proceeds by removing edges one by one from E, and then executing precisely one of lines 5–8 in each case. Line 5 is only performed when $k = 1$ (it is only then that both of the nodes in the edge will be unvisited). The effect of line 5 is to initialize the tree with the first edge from E, and to remove the

1. Let E be the ordered set of edges extracted from the list of node identifiers.
2. $k \leftarrow 1, T \leftarrow \phi, U \leftarrow \{1, 2, \ldots, N\}$
   (k indexes edges $(e_1, e_2, \ldots)$ in E from left to right, T will contain edges in the developing tree, and U is the set of nodes which currently do not appear in T).
3. **If** $|T| = N - 1$, terminate; return the tree T.
4. Remove edge e_k from E, and let $e_k = (u, v)$; (get the next edge)
5. **If** both $u \in U$ and $v \in U$, **then**; (neither node is currently in T)
   add edge e_k to the growing tree T,
   and update U by removing both u and v from it.
   $k \leftarrow k + 1$ (update the index into E), and go to step 3
6. **If** precisely one of u and v is in U, **then**;
   add edge e_k to the growing tree T,
   and update U by removing whichever of u or v it contains.
   $k \leftarrow k + 1$ (update the index into E), and go to step 3
7. **If** edge $(u, v) \in T$ (this edge is already included in T), **then**;
   $k \leftarrow k + 1$ (update the index into E), and go to step 3
8. **If** neither u nor v are in U (both nodes are in T), but (u, v) is *not* in T, **then**;
   (this is the case in which inclusion of (u, v) in T induces a cycle in T)
   add edge (u, v) to T, temporarily inducing a cycle
   8.1. Let C be the set of edges in this cycle
   8.2. Let (p, q) be the highest weight edge in C (breaking ties randomly)
   (in cases where there are distinct cost elements – such as the OCST – we consider these in order, and break ties randomly as a last resort – refer to the text for details on the OCST)
   8.3. Remove edge (p, q) from T, set $k \leftarrow k + 1$, go to step 3.

Fig. 5. Pseudocode for CB-TCR.

two corresponding nodes from U. Line 6 is executed when one of the nodes in the edge has already been visited and the other has not, with the effect of adding this edge to the tree and updating T and u appropriately. Lines 7 and 8 cover the two possible cases in which both nodes in the edge have already been visited; in the first case (line 7), this is when the edge itself is already in the tree, and the effect of line 7 is simply to move on by incrementing k, and ignoring the duplicate edge. Line 8 covers the other case, in which the candidate new edge introduces a cycle in T. The effect of line 8 (incorporating 8.1–8.3) is to include this edge, and then break the cycle by removing a high-weight edge from the cycle.

We note that CB-TCR is clearly order-preserving, greedy, and node-preserving. It follows from Theorems 1 and 2 that, when CB-TCR is in use, every string of length $2(N - 1)$ containing each node at least once, encodes a valid spanning tree, and every spanning tree is encoded by some string.

V. Genetic Operators

In this section, we describe genetic operators associated with EWD, and we also describe a selection scheme we use which seems to lead to good performance.

A. Real-World Tournament Selection (RWTS)

RWTS is a selection and replacement strategy which is used to produce an intermediate population. It was introduced elsewhere [49], and found to lead to slightly superior results overall, which is why we use it here. It exhibits rather high selection pressure (much higher, for example, than standard tournament selection with typically low tournament sizes).

In some of our result tables we give, for completeness, results using RWTS as well as results using standard tournament selection (denoted GTS) with a tournament size of 2 (determined from preliminary work), e.g., the GTS results are in parentheses in Table III. However, we save space in this article by omitting an analysis of RWTS versus GTS in this context, which has already been done for a subset of the DCMST problems, finding that RWTS provides generally better results in [49].

We briefly describe RWTS as follows. RWTS is a strategy for generating an intermediate population I, which is then subject to genetic operators. That is, in describing RWTS, we indicate how to generate the intermediate population I from the current population P. First, all individuals in population P are partitioned into pairs according to index number, and the fittest in each pair (breaking ties randomly) is placed directly into I. (In the case of an odd population size (say, $2k+1$), k such pairings are done, and at this step population I contains k individuals.) These "winning" individuals are called "level 1" winners. In the next and each subsequent step, a subset of the "level l" winners become the "level $l + 1$ winners", and a further copy of each such level $l + 1$ winner is added to I. Level $l + 1$ winners are obtained from level l winners as follows. The level l winners are also paired according to index number (if there is an odd number, $2k+1$, of level l winners, then just k pairs are formed), and the fittest in each of these pairs (breaking ties randomly) become the level $l+1$ winners. When the point is reached at which there is only a single level l winner, the intermediate population is completed by filling its remaining slots with copies of that individual. It is worth noting that the resulting intermediate population I will contain at least l copies of the best individual.

B. Common Gene Preservation Crossover (CGPX) Operator

One specific crossover operator designed for the EWD strategy, which we call "common gene preservation crossover" (CGPX) is tailored to focus on the inheritability of common genetic information [11], as well as to exploit edge-weight information. That is, in case of offspring 1 (offspring 2), CGPX gives priority to genes of parent 1 (parent 2).

CGPX produces two children, O^1 and O^2, from two parents, P^1 and P^2, as follows: We use P_i^k (or O_i^k) to denote the node at position i in individual k. To produce O^1, we first set $O_1^1 = P_1^1$. Then, working from left to right (i.e., from $i = 2$ to $i = 2(N - 1)$), we check the appearance counter $C(i)$ of P_i^1. If $C(i) = 0$, then $O_i^1 = P_i^1$, otherwise, we compare the weights of the edges $\{O_{i-1}^1, P_i^1\}$ and $\{O_{i-1}^1, P_i^2\}$, and make $O_i^1 = P_i^l$, where $l = 1$ or $l = 2$, whichever corresponds to the lower weight of these two edges. However, if one of $l = 1$ or $l = 2$ violates the degree constraint, then we use the other; if both would violate the degree constraint, then we make a random valid choice for O_i^1. We use precisely the same process to generate O^2, except that we begin with $O_1^2 = P_1^2$.

C. Adjacent Node Crossover (ANX) Operator

We introduce another new crossover operator, ANX for the OCST problem. ANX focuses on the inheritability of genetic information from parents, and attempts maximal use of the information contained in the parents. The basic procedure is given in Fig. 6, and we offer the following explanatory text to aid understanding. First (in a similar way to [53]), a node-adjacency map is built for each parent. For example, the set A_4^1 lists all nodes which were adjacent to node 4 in parent A^1. So, if $A^1 = (1, 4, 2, 1, 5, 2, 3, 1)$, then $A_4^1 = \{1, 2\}$. A random node is then chosen to be the start node of the offspring, and becomes the first "current" node. ANX then iterates the following process until the child contains $2(N - 1)$ nodes: where c is the current node, we first find $A_c^1 \cap A_c^2$. If this is nonempty, then

1. Build adjacency maps $A_t^1\{\cdots\}$ and $A_t^2\{\cdots\}$;
2. Select an initial node n randomly. $O_1 \leftarrow n$;
3. **for** i = 2 to L
   $S \leftarrow A_n^1 \cap A_n^2$; $V \leftarrow A_n^1 \cup A_n^2$;
   3.1 **if** S is non-empty **then**
   Let p be the value of j such that, for $j \in S$,
   $D(n,j)$ is minimal (break ties randomly)
   $A_n^{1\ and\ 2} \leftarrow A_n^{1\ and\ 2} - p$ and $A_p^{1\ and\ 2} \leftarrow A_p^{1\ and\ 2} - n$;
   3.2 **else**
   3.2.1 **if** V is non-empty
   Let p be the value of j such that, for $j \in V$,
   $D(n,j)$ is minimal (break ties randomly)
   $A_n^{1\ or\ 2} \leftarrow A_n^{1\ or\ 2} - p$ and $A_p^{1\ or\ 2} \leftarrow A_p^{1\ or\ 2} - n$;
   3.2.2 **else**
   $p \leftarrow r$ (r is a randomly chosen valid node)
   $n \leftarrow p$ and $O_i \leftarrow n$;

$A_t^k\{\cdots\}$: The set of nodes being adjacent to node t in parent k.
O_i : The ith element of the offspring. $D(a,b)$: The weight of edge (a,b).

Fig. 6. Pseudocode for adjacent node crossover.

(breaking ties randomly) the node in this intersection which has the least-weight edge with c is selected; this becomes the next node in the child, and now becomes the current node. If the intersection was empty, then the next gene in the child (which becomes the next current node) is simply that from $A_c^1 \cup A_c^2$ which forms the least-weight edge with c. Finally, if both A_c^1 and A_c^2 are empty, then a random valid node is chosen to be the next node in the child. In each case, as soon as we have established the next node d in the child, d is removed from the adjacency maps A_c^1 and A_c^2 and c is removed from the adjacency maps A_d^1 and A_d^2. The generated offspring replaces the worst of the two parents. Fig. 6 shows the pseudocode of ANX.

D. Repair Processes for CGPX and ANX

When CGPX or ANX are applied as described, the child may not be a valid EWD-encoded tree, since it may violate the need to have at least one occurrence of each node in the offspring. In these cases, the offspring will need to be repaired, and we adopt a simple repair process as follows, which takes advantage of maintaining a check on the number of occurrences $C(k)$ of each node k as the child is developed. If there is some node k with $C(k) = 0$, we first select a gene r randomly, but such that $2 \leq C(r) \leq d-1$, and replace the node r in the child with k. This is repeated until every node has at least one occurrence.

We also note that the ANX operator does not have an inbuilt way to ensure that the child maintains degree constraints (i.e., that each node occurs at most $d-1$ times in the string). In the current work, this issue does not matter since we do not apply ANX on the DCMST. However, we note that it would be straightforward to make appropriate modifications to ANX to be applied to problems with degree constraints.

E. Mutation Operator

In preliminary experiments on small problems [49], we had tested several mutation operators, inversion, reciprocal exchange and random perturbation. Among them, reciprocal exchange mutation gave better results. So, we use only reciprocal exchange mutation in this paper. This mutation operator simply selects two random positions, and then exchanges the genes at these positions.

TABLE I
TEST ENVIRONMENT : CROSSOVER AND MUTATION OPERATORS USED IN THE DCMST EXPERIMENTS

	Selection	Crossover	Mutation
Prüfer	RWTS & GT	Two-Point (60%)	Reciprocal Exchange (60%)
NetKey	RWTS & GT	Uniform (60%)	Reciprocal Exchange (60%)
EdgeSet	RWTS & GT	Heuristic Crossover (80%)	Heuristic Mutation (80%)
EWD	RWTS & GT	CGPX (60%)	Reciprocal Exchange (60%)

VI. EXPERIMENTAL RESULTS

We performed experiments on three classes of spanning tree problems, the DCMST, the OCST, and the q-MST, in order to evaluate all instantiations of EWD that arise from the combination of the three described TCRs and the two described crossover operators. All algorithms were implemented in Visual C++ and every trial of a specific algorithm on a specific problem was repeated 20 times with different random seeds.

A. Experiments on the DCMST Problem

We test our algorithms on the two well-known sets of test instances for the DCMST. One is the set of structured hard instances (SHRD) and the other is Knowles & Corne's instances (K&C) [21]. We generated the SHRD instances using Krishnamoorthy's method [23], producing problems with 30, 40, 50, 60, 70, and 100 nodes, respectively, each leading to three DCMSTs, one for each of the degree constraints 3, 4, and 5. Note that we test sizes of SHRD instances which are commonly used, as well as much larger instances, to thoroughly test the relative scale-up properties of EWD.

In the case of the K&C instances, we use the available existing test instances (rather than generate new ones of the appropriate size) for more exact comparison with the edge set representation, which showed the best results on these instances [36], [38].

The varying genetic operators, the crossover rates, and the mutation rates for Prüfer, NetKey, and EWD have all been selected following preliminary tuning experiments [49]. The genetic operators used for each algorithm are shown in Table I. In all cases, the percentage in parentheses refers to invocations of the operator (e.g., 60% of selected individuals are subject to reciprocal exchange mutation in the NetKey experiments).

Here, in the case of the edge set encoding, we used the specialized crossover, the specialized mutation operator, the crossover rate (80%) and the mutation rate (80%) which were proposed by Raidl *et al.* [36], [38] unless otherwise indicated. The only difference is that we did not use Raidl *et al.*'s duplication elimination scheme; this was a pragmatic choice, enabling fairer comparison with the other algorithms (which also did not employ such a scheme). And, in the cases of Prüfer, NetKey, and the EWD instantiations, we selected the best combination and rates obtained through the preliminary research [49]. Because the Prüfer number encoding is known to have low locality and inheritance [15], [23], [28], many researchers have used low mutation rates (below 1%) with this encoding [14], [23], [57]–[59]. Rothlauf *et al.* [40] did not use a mutation operator using the NetKeys representation, while Schindler *et al.* [45] also did not give an exact description of their mutation method. Anyway, in preliminary testing, we found the Prüfer scheme to

TABLE II
EXPERIMENT RESULTS (GAINS) ON SHRD INSTANCES

Instances		Prüfer	NetKey	ESWH			EWD+CGPX									EWD+ANX								
							CF-TCR			*d*K-TCR			CB-TCR			CF-TCR			*d*K-TCR			CB-TCR		
N	*d*	Avg.	Avg.	Avg.	Best	STD.	Avg.	Best	STD.	Avg.	Best	STD.	Avg.	Best	STD.	Avg.	Best	STD.	Avg.	Best	STD.	Avg.	Best	STD.
SHRD50	3	18.92	18.41	19.65	19.75	0.06	19.40	19.65	0.19	19.80	19.85	0.02	19.58	19.65	0.04	19.41	19.64	0.20	**19.83**	19.85	0.01	19.61	19.73	0.08
		(9.80)	(18.37)	(19.71)	(19.80)	(0.06)	(14.59)	(16.06)	(1.30)	(18.50)	(18.78)	(0.16)	(19.56)	(19.64)	(0.07)									
	4	13.03	12.80	14.42	14.61	0.11	14.26	14.50	0.24	14.67	14.72	0.02	14.58	14.65	0.04	13.30	14.44	1.43	**14.73**	14.74	0.01	14.63	14.70	0.06
		(4.37)	(11.74)	(14.53)	(14.66)	(0.09)	(3.30)	(7.17)	(2.41)	(11.59)	(13.42)	(0.90)	(14.55)	(14.65)	(0.05)									
	5	10.31	8.98	12.13	12.23	0.09	11.90	12.08	0.17	12.28	12.32	0.03	12.16	12.34	0.17	9.58	12.10	3.44	**12.35**	12.36	< 0.00	12.16	12.26	0.08
		(3.03)	(8.29)	(12.11)	(12.19)	(0.05)	(-4.47)	(9.36)	(3.13)	(6.42)	(7.98)	(1.18)	(12.22)	(12.36)	(0.08)									
SHRD60	3	15.43	14.80	15.88	16.08	0.22	15.68	15.98	0.22	16.23	16.25	0.01	16.04	16.13	0.04	15.89	15.99	0.09	**16.26**	16.27	0.01	16.05	16.14	0.07
		(13.58)	(14.18)	(15.52)	(16.03)	(0.38)	(15.43)	(15.88)	(0.33)	(16.11)	(16.17)	(0.03)	(16.02)	(16.10)	(0.04)									
	4	6.92	5.64	8.54	8.69	0.17	8.29	8.48	0.24	**8.86**	8.91	0.01	8.72	8.82	0.04	7.61	8.65	0.83	**8.86**	8.89	0.01	8.73	8.81	0.05
		(5.39)	(5.11)	(8.20)	(8.67)	(0.39)	(7.94)	(8.31)	(0.24)	(8.54)	(8.71)	(0.11)	(8.76)	(8.81)	(0.04)									
	5	3.52	0.92	4.86	4.99	0.07	4.32	4.79	0.44	5.05	5.08	0.02	4.95	5.03	0.04	3.23	4.87	1.54	**5.06**	5.10	0.03	4.94	5.01	0.04
		(1.01)	(0.12)	(4.89)	(4.97)	(0.06)	(3.93)	(4.49)	(0.44)	(4.69)	(4.83)	(0.10)	(5.00)	(5.07)	(0.04)									
SHRD70	3	11.51	11.28	12.31	12.48	0.17	12.00	12.30	0.21	12.55	12.57	0.01	12.40	12.44	0.03	12.32	12.44	0.07	**12.56**	12.58	0.01	12.41	12.46	0.02
		(10.21)	(11.08)	(12.39)	(12.53)	(0.10)	(11.87)	(12.23)	(0.25)	(12.44)	(12.48)	(0.03)	(12.37)	(12.44)	(0.06)									
	4	7.38	6.44	8.62	8.72	0.07	8.29	8.66	0.23	**8.85**	8.86	0.01	8.76	8.79	0.03	6.30	8.30	1.99	**8.85**	8.88	0.01	8.77	8.80	0.03
		(5.39)	(6.62)	(8.67)	(8.76)	(0.04)	(8.14)	(8.65)	(0.29)	(8.58)	(8.72)	(0.09)	(8.78)	(8.82)	(0.02)									
	5	7.00	5.46	8.00	8.19	0.27	7.99	8.15	0.31	**8.43**	8.44	0.01	8.33	8.39	0.03	6.66	8.33	1.45	**8.43**	8.45	0.02	8.34	8.37	0.02
		(4.78)	(4.93)	(8.12)	(8.24)	(0.05)	(7.35)	(8.07)	(0.54)	(7.92)	(8.01)	(0.05)	(8.34)	(8.37)	(0.01)									
SHRD100	3	23.64	22.37	23.82	24.05	0.23	23.86	24.15	0.27	24.29	24.30	< 0.00	24.17	24.22	0.03	24.11	24.22	0.05	**24.30**	24.31	< 0.00	24.18	24.21	0.01
		(22.51)	(21.85)	(24.02)	(24.13)	(0.10)	(23.67)	(23.96)	(0.24)	(24.14)	(24.22)	(0.05)	(24.19)	(24.22)	(0.02)									
	4	12.40	9.87	12.97	13.48	0.37	13.38	13.54	0.19	**13.78**	13.79	< 0.00	13.69	13.71	0.02	10.55	13.31	1.53	**13.78**	13.80	0.01	13.69	13.73	0.02
		(10.73)	(10.40)	(13.40)	(13.60)	(0.17)	(12.90)	(13.35)	(0.33)	(13.27)	(13.39)	(0.06)	(13.69)	(13.73)	(0.02)									
	5	6.21	2.45	7.56	7.75	0.16	7.39	7.72	0.23	7.90	7.92	0.01	7.83	7.86	0.01	4.07	6.98	1.17	**7.91**	7.92	0.01	7.82	7.86	0.02
		(4.48)	(2.25)	(7.75)	(7.82)	(0.04)	(6.02)	(6.83)	(0.50)	(7.16)	(7.29)	(0.09)	(7.83)	(7.86)	(0.02)									

Note: "()" indicates results using standard tournament selection (GTS); N is the number of nodes; d is the degree constraint; STD is the standard deviation.

benefit most from a mutation rate of 60%, while the NetKeys encoding appeared relatively insensitive to the mutation rate.

The population size is set as 100 in all experiments, except in the case of the EWD strategy when using CF-TCR, where it is set to 500 for the K&C instances. In the case of the SHRD instances, the termination condition is set as a 5000 generation limit for problems with 50 to 60 nodes, and 10 000 generations for node sizes 70 to 100. The limit is 20 000 generations in the case of the K&C instances.

Table II summarizes all our experimental results on the test SHRD instances of size 50 nodes or more. In it, a table entry gives the relative difference between the mean (over the 20 trials) C value for the algorithm, and the mean solution for 20 trial runs of d-Prim $(C_{d-\mathrm{Prim}})$[2] as follows:

$$\mathrm{Gains} = (C_{d-\mathrm{Prim}} - C)/C_{d-\mathrm{Prim}} \times 100(\%). \qquad (5)$$

Bold entries indicate the best result over all algorithms for a particular problem. EWD using dK-TCR, ANX and RWTS found the best solutions on all SHRD instances and showed better overall performance than ESWH. The standard deviation (STD) of Gap was also low. On the 100-node instances, the differences in average gains between EWDs best results and the others' best results were 0.66% for the Prüfer, 1.93% for NetKeys, 0.28% for ES, and 0.44% for CF-TCR in the case of degree constraint 3; 1.38% for Prüfer, 3.38% for NetKeys, 0.39% for ES, and 0.41% for CF-TCR in the case of degree constraint 4, and 1.70% for Prüfer, 5.46% for NetKeys, 0.16% for ES, and 0.52% for CF-TCR in the case of degree constraint 5.

[2]d-Prim is a version of Prim's algorithm commonly used to quickly generate low-cost DC-MSTs; it is simply Prim's, modified to maintain degree constraints. In our work, one trial of d-Prim's consists of running it 20 times with randomly chosen starting node each time, and returning the best of those 20 results.

TABLE III
STATISTICAL ANALYSIS : t-TEST RESULTS—EWD (dK-TCR, ANX, RWTS) VERSUS ESWH (GT)

N	*d*	*t* value	*P*	N	*d*	*t* value	*P*
30	3	2.325	0.032	60	3	6.155	<0.001
	4	2.939	0.009		4	5.334	<0.001
	5	0.976	0.342		5	7.289	<0.001
40	3	6.039	<0.001	70	3	4.981	<0.001
	4	8.845	<0.001		4	10.68	<0.001
	5	6.342	<0.001		5	16.34	<0.001
50	3	5.596	<0.001	100	3	8.305	<0.001
	4	6.363	<0.001		4	7.063	<0.001
	5	12.73	<0.001		5	11.46	<0.001

Especially, EWD using dK-TCR and CB-TCR showed better performance except in the case of $N = 50, d = 3$.

Comparing the selection strategies, RWTS showed better results than GTS overall, although specifically in the case of ESWH, the RWTS strategy was less effective than GTS. We omit a more detailed analysis of RWTS versus GTS since that is not the focus of this work.

The results of t-tests on the solution quality measures are shown in Table III. First, we note that, when the new encoding is used, EWD using dK-TCR, ANX, and RWTS always statistically outperformed the other variants of EWD except EWD using dK-TCR, CGPX, and RWTS. Therefore, a comparison between EWD using dK-TCR, ANX, and RWTS, and ESWH using GT is provided in Table III.

ESWH was never statistically better than the new encoding in these 18 cases and, hence, the table indicates either that no statistical difference was found (e.g., the 30-node problems), or, when a p-value is given, it indicates that EWD was better than ESWH

TABLE IV
EXPERIMENTAL RESULTS (GAPS) ON THE K&C DCMST INSTANCES WITH MAXIMUM DEGREE 5

Instances	Opt*	ESWH*	ESWOH*	CF-TCR				dK-TCR				CB-TCR			
				Worst	Avg.	Best	STD	Worst	Avg.	Best	STD	Worst	Avg.	Best	STD
50n1	6.60	0.00	1.19	9.39	4.11	1.67	1.87	6.21	3.71	1.21	1.51	0.00	0.00	0.00	0.00
50n2	5.78	< 0.01	1.45	6.75	2.77	1.38	1.62	5.19	2.83	1.21	1.11	0.00	0.00	0.00	0.00
50n3	5.50	0.00	0.92	5.64	3.03	1.09	1.29	5.09	3.97	2.91	0.55	0.00	0.00	0.00	0.00
100n1	11.08	< 0.01	6.58	11.28	5.58	2.98	2.33	12.09	7.55	5.14	1.52	0.27	0.07	0.00	0.06
100n2	11.33	< 0.01	10.78	14.21	6.98	1.85	3.50	22.15	8.76	4.24	3.99	0.44	0.12	0.00	0.12
100n3	10.19	0.01	14.70	17.17	10.63	4.61	3.58	17.76	11.24	6.77	3.00	1.28	0.32	0.00	0.44
200n1	18.33	0.02	17.64	15.66	11.13	9.33	1.65	37.42	25.64	18.82	3.88	4.26	1.71	0.60	0.91
200n2	19.16	0.02	26.30	18.06	12.07	7.36	2.81	28.71	22.15	15.61	3.76	2.61	1.42	0.37	0.67
200n3	16.13	0.01	9.78	19.90	14.41	10.17	2.82	35.28	31.30	26.35	2.26	4.40	2.25	0.68	0.96

" * " indicates results from [38]. ESWH and ESWOH represent edge set with heuristics and edge set without heuristics respectively.

and all the results are significant at $p < 0.001$. As the node size increases (over $N = 40$), t test results are significant at $p < 0.0001$.

We conclude from this that the new encoding performs generally as well as the previous best method for the DCMST, at least on the SHRD problems, and most significantly outperforms the previous best method on the larger-sized instances.

We mention some additional observations on the raw data from all trials. To simplify, we define best(n/d) to be the best result from all trials of every algorithm on the problem with n nodes and degree-constraint d. (Note that we can typically expect best(n/d) to be found several times among the trials of the better performing algorithms). Also, let best $(n/d, M)$ be the best result among the trials of method M, with worst $(n/d, M)$ defined analogously. In *every* case, best$(n/d, \mathrm{EWD})$ was the same as best(n/d) for some instantiation of EWD. Further, it was always the case that worst$(n/d, \mathrm{EWD})$ was better than worst$(n/d, \mathrm{ES})$. Finally, best$(n/d, \mathrm{EWD})$ was better than best$(n/d, \mathrm{ES})$ for all instances. The Prüfer results show perhaps surprisingly good performance; this partly arises from our preliminary work to identify the best parameter settings to use. It is also interesting to note that the Prüfer results benefit particularly well from the RWTS strategy.

Table IV shows experimental results which show the comparison between ES [38] and the new encoding on the K&C benchmark instances, which were designed to be especially deceptive (rather than to reflect real-world problems) [21].

We directly report Raidl and Julstrom's results, where they used the following simple replacement strategy to avoid duplicates in the population; if an offspring does not duplicate an existing solution, it replaces the worst solution in the population. This differs from the replacement strategy we use, which is a generational strategy (the offspring replace the previous generation). We did not implement Raidl and Julstrom's strategy, preferring to avoid the computational cost of searching for duplicates, especially on large problem sizes. However, we invite the reader to agree that, with this slight difference in strategies, the comparison between the algorithms is more likely to be biased in favor of the ES approach, since the replacement method is a diversity preservation mechanism which tends to correlate highly with better performance.

Further differences between our experimental setup and that of Raidl and Julstrom are the population size and the termination condition. Raidl and Julstrom use a population size of 500, and we use a population size of 100. For the same number of evaluations, preliminary EWD tests showed no significant difference for these population sizes, and we prefer the smaller population size. As regards the termination criterion, Raidl and Julstrom's strategy was to terminate when there had been no improvement in best-so-far after 1 000 000 consecutive iterations. Given the nontrivial computation cost of CB-TCR, we preferred to use the far less computationally intensive strategy of simply terminating after 20 000 generations (2 000 000 evaluations). We argue that the balance of probabilities suggests that these choices are biased against our approach in terms of performance, which in turn underlines the significance of any evidence we find that shows a performance advantage for EWD.

In Table IV, all entries represent the relative Gap between the mean (over the twenty trials) solution C for the algorithms, and the optimum solution C_{opt} [38] as follows:

$$\mathrm{Gap} = (C - C_{\mathrm{opt}})/C_{\mathrm{opt}} \times 100\%. \tag{6}$$

EWD with CB-TCR found the optimum solution in all trials on the 50-node instances. CB-TCR also found the optimum solutions on the 100-node instances, but the mean Gap ranged from 0.07 to 0.32. On the 200-node instances, CB-TCR did not find the optimum solutions and the mean Gap ranged between 1.71 and 2.25. ESWH was clearly better on the larger instances of these problems, however, it is also clear that CB-TCR was clearly always better than ESWOH at all sizes of problem. To save space, we omit a full statistical report, but we note that *t*-tests show with confidence above 97% that CB-TCR outperforms ESWOH on all sizes of problem, and that ESWH outperforms CB-TCR on the two larger sizes of problem, while there is no statistically conclusive difference between ESWH and CB-TCR on the smaller problems.

B. Experiments on the OCST Problem

We used the existing test benchmark instances [41], as well as some randomly generated instances (since we wanted to experiment with larger problems than have previously been addressed) of the OCST problem. In the case of the preexisting benchmark instances, we have already tested our algorithm using CB-TCR and ANX on some of these benchmarks, and have shown that it can outperform previous results [50]. In that work, we found a better solution than the previously best known for the Berry35U

benchmark. In this paper, we provide further results on existing OCST benchmarks and analyze certain properties of the ANX operator, and also investigate the scale-up properties of our and other methods on larger OCSTs.

Rothlauf *et al.* [42] analyzed properties of OCST instances in terms of whether the weights were randomly generated or were Euclidean. In their experiments, they showed that when random weights are used, the optimal solutions are more similar to the simple MST than when Euclidean distances are used for weights. It is known that the better performing methods on the OCST are those that exhibit a bias toward the MST [42]. We also suspected that the encoding and operators involved in most of the instantiations of EWD would exhibit such a bias (see Section VII.D). When we generated OCST instances, we therefore generated ones with Euclidean distances in order to pose a greater challenge to the biased methods.

1) Benchmark Results and Analysis of ANX's Properties: As mentioned earlier, work submitted elsewhere [50] has shown EWD $(\mathrm{CB} - \mathrm{TCR} + \mathrm{ANX})$ to outperform other approaches on the standard benchmark instances of the OCST. We briefly summarize those results here. EWD was compared with the NB, ESWH, ESWOH, Netkey, and Prüfer encodings on all of the well-known OCST benchmarks [28], [29], [42], which ranged in size from 6 to 50 nodes, and are almost all real-world or real-world example inspired instances. EWD was found statistically superior to all other approaches on these cases except in the case of "Palmer 24," where it was outperformed by ESWOH, but even in that case EWD found the best known solution in 17 out of 20 trials. EWD also found better than previously published solutions on two of the other benchmarks (a 35- and a 50-node instance), and generally required much less computational time than the other methods. NB was generally good and fast on small benchmark instances (up to 12 nodes), but performed poorly on larger ones. As we will see later, the situation is not quite the same on randomly generated instances.

Before proceeding to results on new and larger OCST instances, we report on some further work on the standard OCST benchmarks concerning the ANX operator. As mentioned earlier, the ANX operator may need a repair process for the offspring generated after it is applied to parent, and also it may make a random selection of a node during the process of generating offspring (see 3.3 of Fig. 6). In the process of designing ANX, these processes seemed necessary (given that we did not want to instill undue computational complexity into ANX), although this appeared to introduce potentially undesirable elements of randomness, which we felt may limit performance, especially with tight degree constraints. However, results so far with ANX (on the benchmark OCSTs, as just discussed) have shown excellent performance, hence, we were motivated to better understand this operator. In particular, we want to understand how many times the random selection typically occurs and how much it may affect overall performance.

We, therefore, performed experiments to answer the following questions.

- *What percentage of individuals in a population typically need a random selection when involved in ANX crossover?*—If this proportion was high, this would bias ANX toward poor locality and heritability characteristics.
- *In an individual where random selection occurs, typically how many edges are added via random selection?*—If many edges are selected this way, ANX will find it difficult to find good solutions since relatively significant information from the parents is lost.
- *Does the random selection occur more often for particular nodes?*—If nodes with *high degree* in the tree structure of good parent solutions are involved in the random selection step, the resulting tree is likely to diverge considerably from the parents.

TABLE V
MEAN NUMBER OF EDGES INVOLVED IN RANDOM SELECTION STEPS WHEN USING ANX. COLUMN A GIVES THE MEAN PER GENERATION, AND COLUMN B GIVES THE MEAN PER INDIVIDUAL, CALCULATED ONLY AMONG INDIVIDUALS WHICH ARE SUBJECT TO THE RANDOM SELECTION STEP AT ALL. "B/A" SHOWS THAT THE AVERAGE NUMBER OF EDGES SELECTED RANDOMLY PER INDIVIDUAL PER GENERATION IS BELOW 2 IN ALL INSTANCES

	Per Population(A)	Per Individual(B)	B/A
Palmer6	2.7	2.7	1
Palmer12	2.14	2.26	1.06
Palmer24	11.13	15.05	1.35
Berry6	0.61	0.61	1
Berry35NU	9.53	18.27	1.92
Berry35U	14.51	21.39	1.47
Raidl10	2.12	2.13	1
Raidl20	5.79	6.7	1.16
Raidl50	9.14	14.74	1.56

Table V indicates the average proportion of individuals in a population which undergo random edge selection during ANX per generation, averaged over all generations, and also indicates the frequency with which this process occurs within those individuals. These are the mean figures for ten trial runs of $\mathrm{CBCTR} + \mathrm{ANX}$ with other algorithm design parameters as previously described. "B/A" indicates the average number of edges selected randomly over the whole population, and we note that it never exceeds 2 during an entire run.

Fig. 7 indicates the results after analysis of ANX's properties. On the left are the tree structures of the best known solutions and on the right we see, for each node, the frequency per generation with which that node is involved in the random selection process. In all cases, nodes which indicate high frequency (nodes selected frequently during the random selection process) are almost always *leaf nodes* in the tree structures of the best known solutions. On the contrary, the low frequency nodes turned out to be only relatively high-degree nodes. For example, on Palmer's 24-node benchmark, nodes 1, 4, 8, and 21 were high-frequency and are leaf nodes in the optimum solution. Meanwhile, nodes 2, 14, 18, and 22 were all low-frequency, and all have high degree in the best-known structure. These results suggest that the random selection process involved in ANX should not detrimentally affect progress toward near-optimal trees, since those nodes most critical in the overall structure of good trees are least affected by random selection, and the frequencies in any case are very low per generation.

2) Test Results: As mentioned above, in the interest of providing challenging test instances, we generated random OCST instances using Euclidean distances as follows. Requirements

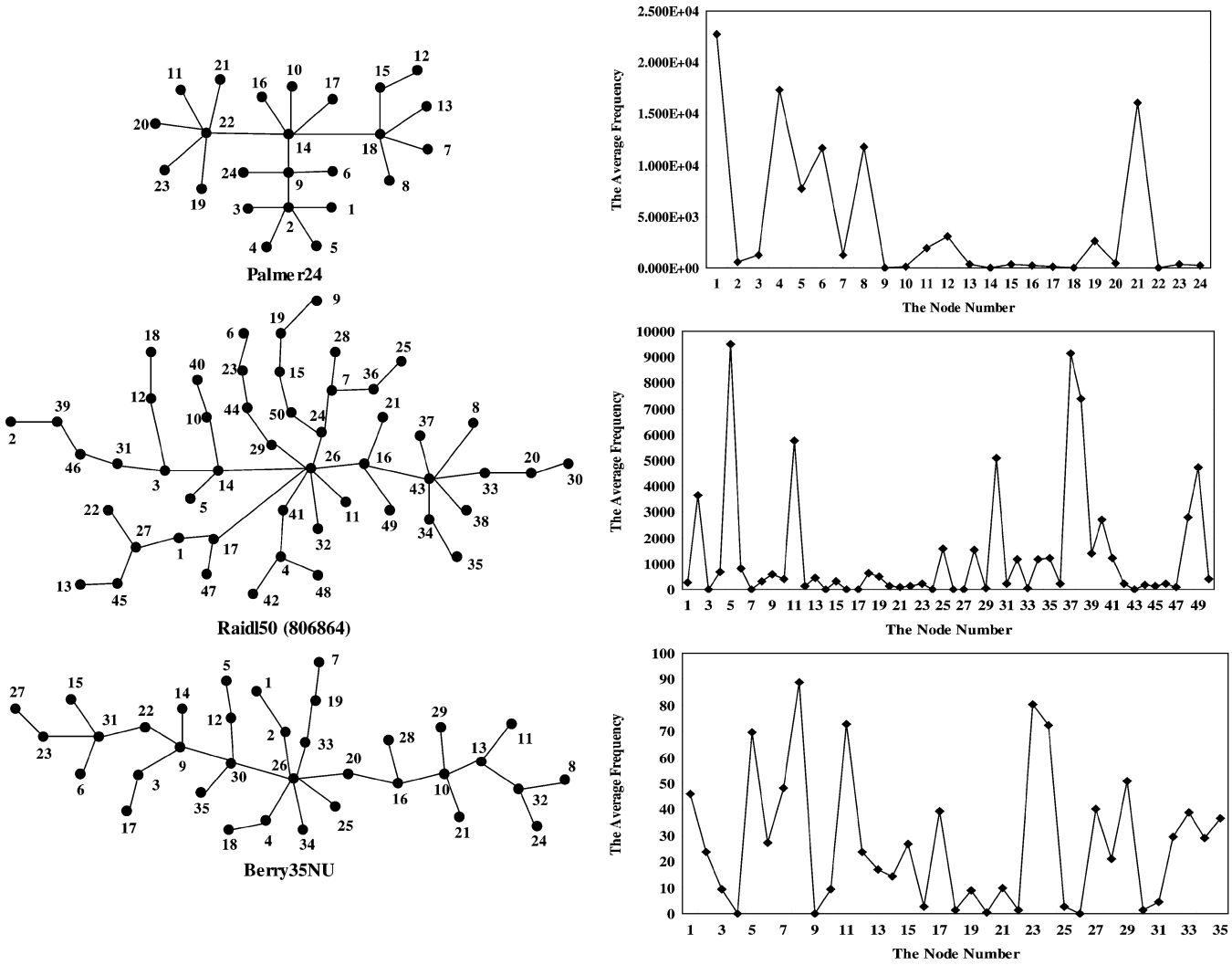

Fig. 7. Tree structure of the best known solutions and the frequency of each node involved in random selection in ANX on the Palmer24, Raidl50, and Berry35NU instances.

TABLE VI
TEST ENVIRONMENT: GENETIC OPERATORS USED FOR EACH ALGORITHM IN THE OCSTP EXPERIMENTS

	Selection	Crossover	Mutation
NetKey	RWTS	Uniform (60%)	Reciprocal Exchange (60%)
NB	RWTS	One-Point (60%)	Random Perturbation (1%)
EdgeSet	GTS	Heuristic Crossover (80%)	Heuristic Mutation (80%)
EWD	RWTS	ANX (60%)	Reciprocal Exchange(60%)

$(R(i,j))$ between two nodes i and j were generated randomly and were uniformly distributed in the interval $[0, 100]$, and nodes were placed randomly on a 500×500 two-dimensional (2-D) plane. Population size was 100 and the termination condition of each algorithm was set at 10 000 generations for $N = 20$ to 40 and 5000 generations for $N = 60$.

Table VI shows the experimental environment used for each tested representation for the OCST trials. In nearly all cases, the percentage in parentheses refers to invocations of the operator (e.g., 60% of selected individuals are subject to reciprocal exchange mutation in the NetKey experiments). In just one case (the random perturbation operator), the 1% refers to the amount of the individual that is disturbed, while all (100%) selected individuals undergo the operation.

The NB and ES were implemented as suggested in [28], [29], and [38], respectively, but NB used our RWTS selection strategy because this returned better results than the alternative in preliminary experiments. All experiments were performed on a Pentium IV 1.6 GHz CPU.

Table VII shows the results in terms of (Gap) for the randomly generated test instances. Bold entries indicate the best results over all algorithms for each particular problem. The EWD using CB-TCR+ANX and NB using RWTS as the selection strategy returned the best solutions. EWD managed to find the best solution for each problem among its 20 trials, while NB failed to find the best solution on the 60-node instance.

In particular, on the 40-node instance, the proposed algorithm was superior to NB, but on the 30- and 60-node instances, NB was superior to the proposed algorithm.

The average Gap ranged from 0.00% (NB, $N = 30$) to 15.45% (ESWOH, $N = 60$). But in the case of CB-TCR+ANX, the average Gap ranged from 0.00% to 13.51% (EWD, ANX,

TABLE VII
EXPERIMENT RESULTS (GAPS) ON RANDOMLY GENERATED OCST INSTANCES

N	Best	NB				NetKey				ESWH				ESWOH				EWD+ANX CB-TCR			
		Min.	Avg.	STD	CPU	Min.	Avg.	STD	CPU	Min.	Avg.	STD	CPU	Min.	Avg.	STD	CPU	Min.	Avg.	STD	CPU
20	3,232,359	0.00	**0.00**	0.00	610.2	0.00	0.77	1.03	1140.7	2.04	2.33	0.13	492.3	1.25	1.84	0.43	516.3	0.00	**0.00**	0.00	785
30	6,363,993	0.00	**0.00**	0.00	2158.7	0.03	0.85	1.05	2599	2.65	2.93	0.15	1606.2	4.40	5.48	0.30	1688.1	0.00	0.13	0.12	2192.1
40	14,301,220	0.00	0.16	0.20	5094.7	0.30	1.59	1.43	5645.7	0.33	1.14	0.47	3899.9	4.74	6.65	1.39	4193.1	0.00	**0.12**	0.15	4843.6
60	29,679,402	0.04	**0.05**	0.18	9080.4	3.78	11.78	6.66	8999.6	12.34	14.28	1.39	6719.2	12.02	15.45	3.15	7157.1	0.00	1.89	2.45	7975.9
Total Avg.			**0.05**				3.75				5.17				7.35				0.53		

N	Best	EWD+ANX CF-TCR				EWD+ANX *d*K-TCR				EWD+CGPX CB-TCR				EWD+CGPX CF-TCR				EWD+CGPX *d*K-TCR			
		Min.	Avg.	STD	CPU	Min.	Avg.	STD	CPU	Min.	Avg.	STD	CPU	Min.	Avg.	STD	CPU	Min.	Avg.	STD	CPU
20	3,232,359	3.36	10.42	5.20	584.7	0.00	0.63	0.66	696.2	0.00	0.25	0.52	388	0.86	2.34	0.89	299.7	0.00	0.50	0.64	333.9
30	6,363,993	1.55	5.88	5.17	1773.4	0.00	0.05	0.08	1953.4	0.00	0.03	0.06	1277.5	0.32	1.18	0.55	1411.4	0.00	0.08	0.12	1059.1
40	14,301,220	2.89	10.53	4.92	4140.9	0.03	0.17	0.11	4222.9	0.03	0.23	0.18	2554.8	2.65	3.97	1.40	2340.4	0.00	0.27	0.23	2546.8
60	29,679,402	5.57	13.51	5.70	7025.5	0.00	0.95	1.88	7313.3	0.10	2.24	2.85	4233.9	2.65	8.40	4.11	3899.3	0.19	2.24	3.09	4100.4
Total Avg.			10.09				0.45				0.69				3.97				0.77		

"Best" : the found best solution. "CPU" : computation time (sec.).

TABLE VIII
STATISTICAL ANALYSIS OF SOLUTION QUALITY: t-TEST RESULTS FOR $\mathrm{CB}-\mathrm{TCR}+\mathrm{ANX}$ VERSUS OTHER METHODS ON THE RANDOMLY GENERATED OCST INSTANCES

(a)

N	NB		NetKey		ESWH		ESWOH	
	t value	P	*t* value	P	*t* value	P	*t* value	P
20	NA	NA	29.75	< 0.0001	282.4	< 0.0001	81.44	< 0.0001
30	-4.25	< 0.0001	3.06	0.0041	62.71	< 0.0001	72.92	< 0.0001
40	0.822	0.416	4.584	< 0.0001	9.226	< 0.0001	20.84	< 0.0001
60	-3.355	0.0018	6.232	< 0.0001	19.63	< 0.0001	15.17	< 0.0001

(b)

N	NB		NetKey		ESWH		ESWOH	
	t value	P	*t* value	P	*t* value	P	*t* value	P
20	-91.89	< 0.0001	9.269	< 0.0001	-139.7	< 0.0001	-84.51	< 0.0001
30	-0.6518	0.5185	8.57	< 0.0001	-48.9	< 0.0001	-39.65	< 0.0001
40	22.89	< 0.0001	9.174	< 0.0001	-72.05	< 0.0001	-21.04	< 0.0001
60	23.44	< 0.0001	13.77	< 0.0001	-43.25	< 0.0001	-24.12	< 0.0001

CF-TCR, $N = 60$). Taking into account the total average Gap except for EWD using CF-TCR, the difference in performance between the variants of EWD and other algorithms is obviously large, with the exception of NB. In this case, the average Gap was not over 0.8%. Anyway, in case of OCST problem EWD using CF-TCR, especially EWD using ANX and CF-TCR, showed relatively bad performance.

The relative rankings of the encodings, based on Gap, are NB, EWD, NetKey, ESWH, and ESWOH in descending order of overall performance.

Table VIII indicates the t test results on (a) solution quality and (b) computation time between the EWD representation and each other representation. The results indicate that there are significant differences in solution quality between EWD and the other methods (except the case of NB) and similarly significant differences in computation time. In (a), The EWD representation is statistically superior to NetKey, ESWH and ESWOH. All the results except the case of NetKey at $N = 30$ are significant at $p < 0.0001$. In (b), all the results except the case of NB at $N = 30$ are significant at $p < 0.0001$. While the proposed method is better than NB and NetKey in terms of computation time, it is worse than ESWH and ESWOH.

C. Experiments on the q-MST Problem

For the q-MST problem, we used six randomly generated test instances with sizes ranging from 50 to 100 nodes and all graphs were complete graphs as in the previous experiments. The intercost matrix for the q-MST problem contains integer values which are randomly generated and uniformly distributed over [1], [20]. The coordinates of each node are also randomly generated on a 500×500 2-D grid. We used integer Euclidean distance values, as with the OCST instances.

The experimental setup used for each algorithm is the same as in the previous two experiments. Here, $\mathrm{pop\_size}$ is set as 100 in all cases and the termination condition is set to 10 000 generations at all instances.

Table IX gives the results obtained from each representation. Here, solution quality is again measured by the relative difference (Gap) between the average solution of each algorithm and the best found solution over all relevant experiments.

The best solutions were returned by EWD using dK-TCR and ANX, with Gap ranging from minimum 0% (in all instances except $\mathrm{N} = 60$) to maximum average 0.70% (in the 70-node case). The comparative ranges for the other algorithms were 3.71–4.78% (NB), 1.55–3.51% (ESHW), 2.76–5.28% (NetKey), and 7.78–10.56% (Prüfer). The differences in total average Gap between EWD and others were 3.70% (EWD versus NB), 2.24% (EWD versus ESWH), 3.63% (EWD versus NetKey), and 8.67% (EWD versus Prüfer), respectively. Considering the total average Gap, the ranking in performance is EWD, ESWH, NetKey, NB, and Prüfer in descending order of performance.

Table X shows the statistical analysis results. As the t-test results show, EWD is statistically superior to others in performance comparison (a). All the results are significant at $p < 0.0001$. Considering the computation time, EWD is statistically inferior to ESWH in all instances and Prüfer in $\mathrm{N} = 50 \sim 80$, but it is superior to NB and NetKey in all instances and to Prüfer in the largest instance.

TABLE IX
EXPERIMENTAL RESULTS ON RANDOMLY GENERATED q-MST INSTANCES

N	Best	NB				ESWH				NetKey				Prüfer				EWD+CGPX CB-TCR			
		Min.	Avg.	STD.	CPU	Min.	Avg.	STD.	CPU	Min.	Avg.	STD.	CPU	Min.	Avg.	STD.	CPU	Min.	Avg.	STD.	CPU
50	25339	2.92	3.71	0.52	1075.8	1.08	1.55	0.20	155.9	1.87	2.76	0.47	474.3	8.00	10.56	1.31	233.8	0.12	0.56	0.28	591.1
60	36014	4.31	4.78	0.40	1883.4	2.23	3.06	0.39	225.6	1.98	3.43	0.70	788.5	7.71	9.33	1.06	419.7	1.15	1.94	0.38	854.2
70	48538	3.64	4.43	0.50	2947.6	2.83	2.93	0.05	286.7	3.17	4.08	0.45	1127.3	7.66	9.40	0.87	630.5	0.21	2.16	0.51	1138.4
80	63546	3.31	3.95	0.30	4458.2	3.23	3.51	0.14	350.0	3.57	4.76	0.67	1584.5	8.43	9.49	0.68	1010.5	0.82	2.48	0.68	1533.5
90	79627	3.69	4.45	0.41	6455.1	2.86	3.09	0.11	463.5	3.85	5.28	0.70	2112.4	7.92	9.18	0.70	1391.6	0.78	2.61	1.01	1972.4
100	98342	4.08	4.64	0.37	8939.7	2.71	3.05	0.15	578.3	3.92	5.21	0.64	3000.8	6.21	7.78	0.72	1969.2	0.50	1.03	0.28	2607
Total Avg. Gap			4.43				2.86				4.26				9.29				1.80		

N	Best	EWD+CGPX CF-TCR				EWD+CGPX *d*K-TCR				EWD+ANX CB-TCR				EWD+ANX CF-TCR				EWD+ANX *d*K-TCR			
		Min.	Avg.	STD.	CPU	Min.	Avg.	STD.	CPU	Min.	Avg.	STD.	CPU	Min.	Avg.	STD.	CPU	Min.	Avg.	STD.	CPU
50	25339	1.53	2.58	0.60	158.5	0.00	0.59	0.31	302.95	0.00	0.48	0.33	1015.1	1.15	2.04	0.49	217.5	0.00	**0.46**	0.28	343.0
60	36014	1.61	2.55	0.58	291.1	0.00	**0.62**	0.41	448.4	0.04	0.71	1491.5	806.3	1.12	2.15	0.73	375.3	0.20	0.69	0.35	495.7
70	48538	1.92	2.78	0.52	459.7	0.13	**0.62**	0.36	699.7	0.45	1.04	1951.5	1239.0	1.70	2.48	0.38	555.1	0.00	0.70	0.32	716.6
80	63546	2.03	2.74	0.48	771.4	0.45	0.97	0.34	1044.45	0.00	0.70	0.31	2255.9	1.68	2.48	0.46	828.5	0.00	**0.61**	0.37	1086.7
90	79627	2.62	3.34	0.47	1171.3	0.51	0.98	0.25	1323.75	0.37	0.76	0.20	2323.7	1.80	2.61	0.32	1206.7	0.00	**0.68**	0.28	1337.2
100	98342	2.72	3.20	0.35	1681.1	0.44	1.00	0.31	1773.45	0.23	0.72	0.31	4106.3	1.67	2.30	0.28	1739.8	0.00	**0.60**	0.23	1828.9
Total Avg. Gap			2.87				0.80				0.74				2.34				**0.62**		

"Best" : the found best solution. "CPU" : computation time (sec.). All best solution were found by EWD.

TABLE X
STATISTICAL ANALYSIS OF SOLUTION QUALITY: t-TEST RESULTS FOR $d\mathrm{K}-\mathrm{TCR}+\mathrm{ANX}$ VERSUS OTHER ALGORITHMS ON THE RANDOMLY GENERATED q-MST INSTANCES.
(a) SOLUTION QUALITY. (b) COMPUTATION TIME

(a)

N	ESWH		NetKey		Prüfer		NB	
	t value	P	t value	P	t value	P	t value	P
50	13.99	< 0.0001	18.76	< 0.0001	33.73	< 0.0001	24.55	< 0.0001
60	20.16	< 0.0001	15.62	< 0.0001	34.51	< 0.0001	34	< 0.0001
70	29.88	< 0.0001	26.97	< 0.0001	41.66	< 0.0001	27.78	< 0.0001
80	32.19	< 0.0001	23.93	< 0.0001	50.82	< 0.0001	30.89	< 0.0001
90	35.18	< 0.0001	27.17	< 0.0001	50.24	< 0.0001	33.6	< 0.0001
100	39.64	< 0.0001	29.97	< 0.0001	42	< 0.0001	41.18	< 0.0001

(b)

N	ESWH		NetKey		Prüfer		NB	
	t value	P	t value	P	t value	P	t value	P
50	-167.7	< 0.001	72.58	< 0.001	-29.71	< 0.001	78.3	< 0.0001
60	-62.6	< 0.001	53.47	< 0.001	-11.23	< 0.001	39.13	< 0.0001
70	-70.6	< 0.001	59.74	< 0.001	-7.781	< 0.001	189.9	< 0.0001
80	-14.43	< 0.001	9.566	< 0.001	-1.466	0.150	63.57	< 0.0001
90	-35.99	< 0.001	27.44	< 0.001	1.783	0.082	152.2	< 0.0001
100	-72.19	< 0.001	30.29	< 0.001	4.129	< 0.001	171.1	< 0.0001

VII. ANALYSIS

In recent years, many researchers have proposed various analytical methods designed to help reveal the basic principles of encodings in EAs (and used of course in black box search in general). Manderick *et al.* [25] used correlation coefficients for the fitness values of solutions before and after operators were applied. Sendhoff *et al.* [47] proposed the concept of "causality" to analyze the locality of EAs. Gottlieb *et al.* [10], [16], [37] proposed "mutation innovation," "crossover innovation," and "crossover loss" to emphasize the importance of locality and heritability. Merz *et al.* [26] and Reeves *et al.* [39] used fitness landscape analysis. Besides these, much literature has dealt with methods specifically for analyzing the properties of encodings [18], [40], [43].

Here, we present some analysis of the encodings used in this work, in attempt to identify what may be behind the generally strong performance of the EWD strategy. In order, we provide the results of some investigations into locality, inheritability, and bias. We avoid an overlong analysis by mainly restricting this presentation to selected cases. For analysis in terms of locality and heritability, we focus on the OCST, and consider (as well as the non-EWD encodings) only the instantiation of EWD that performed best overall on OCST problems. However, when we analyze bias (the tendency of the encoding and/or operator to lean toward a nonuniform distribution in phenotype space), we recognize that this has been found to be a particularly useful explanatory factor in studies of tree-based problems [52], and we report results for each of the EWD instantiations, as well as ES, NB, and Prüfer.

We begin with preliminaries and definitions, and then proceed to consider locality, heritability, and diversity (in that order) with discussion of fitness bias interleaved.

A. Definitions and Other Preliminaries

In order to analyze the properties of an encoding, suitable metrics need to be defined. The search spaces of interest include the genotype space (e.g., the space of data structures that encode spanning trees), and the phenotype space (i.e., the space of spanning trees). Genetic operators (normally) work with genotypes, and movement in genotype space (e.g., from a parent genotype to a mutant genotype) corresponds to a related movement in phenotype space, from the parent phenotype to the mutant phenotype. Movements in phenotype space also correspond to changes in a third space of interest, the fitness space, which in this paper is simply the subset of the real numbers which correspond to valid objective cost function values.

The first definition we need to clarify is that of *genotypic distance*, i.e., the distance between two genotypes in genotype space. Since this is dependent on the encoding used, it must be

TABLE XI
COMPARING OF LOCALITY ON SIX OCST INSTANCES: Palmer24, Berry35U, AND FOUR RANDOMLY GENERATED INSTANCES OF SIZE 70, 80, 90, AND 100 NODES, RESPECTIVELY, BASED ON RANDOMLY GENERATING 1000 GENOTYPES AND APPLYING MUTATION ONCE TO EACH

Palmer24	Prüfer	NB	NetKey	ESWH	ESWOH	EWD	Berry35U	Prüfer	NB	NetKey	ESWH	ESWOH	EWD
$P(MI=0)(\%)$	0.00	80.5	80.3	0.00	0.00	4.10		0.00	93.5	87.3	0.00	0.00	0.00
$E(MI\|MI>0)$	4.49	7.87	1.51	1.00	1.00	2.42		5.08	33.0	1.49	1.00	1.00	11.70
$Max(MI)$	12	22	2	1	1	7		17	33	3	1	1	18
$\sigma(MI\|MI>0)$	2.01	5.67	0.50	0.00	0.00	1.02		2.64	0.00	0.51	0.00	0.00	2.24
Rand70							Rand80						
$P(MI=0)(\%)$	0.00	68.1	91.0	0.00	0.00	2.40		0.00	70.1	90.1	0.00	0.00	2.80
$E(MI\|MI>0)$	7.38	5.53	1.58	1.00	1.00	2.63		8.02	5.92	1.48	1.00	1.00	2.65
$Max(MI)$	31	22	2	1	1	7		29	23	3	1	1	9
$\sigma(MI\|MI>0)$	5.56	4.22	0.49	0.00	0.00	1.01		5.94	4.52	0.54	0.00	0.00	1.01
Rand90							Rand100						
$P(MI=0)(\%)$	0.00	71.1	92.2	0.00	0.00	1.70		0.00	69.4	90.5	0.00	0.00	1.50
$E(MI\|MI>0)$	8.59	5.62	1.46	1.00	1.00	2.71		9.49	6.03	1.45	1.00	1.00	2.69
$Max(MI)$	36	23	3	1	1	6		38	23	3	1	1	5
$\sigma(MI\|MI>0)$	6.88	4.41	0.55	0.00	0.00	1.01		7.99	4.84	0.57	0.00	0.00	0.98

defined in a way that has suitable generality and universality. Following concepts in evolutionary biology [46], we simply define genotypic distance as the smallest number of individual mutations required for the interconversion of two genotypes. This is straightforwardly measured for each of the encodings used in this paper.

Meanwhile, phenotypic distance and fitness distance are both independent of the encoding used, but are both dependent on the problem domain. However, since in all cases our phenotypes are spanning trees, we can define phenotypic distance (d_p) as follows.

- The *phenotypic distance* $d_p(T_1, T_2)$ between two trees T_1 and T_2 is the number of edges in $T_1 \cup T_2$ which do not appear in $T_1 \cap T_2$. This is equivalent to the *Hamming distance* in the case where T_1 and T_2 are represented in a binary edge-list encoding.

Meanwhile, fitness distance (d_f) is simply measured as the difference in fitness value between two phenotypes.

When we investigate bias, we follow [52] in considering the tendencies of the various encodings and operators to favor phenotypes in the region of the MST. The metric we use is $d_{\text{mst-pop}} = (\sum_{i=1}^{n} d_p(T_i, \text{MST}))/N$, which gives the mean distance from the MST over all individuals in a population of size N.

B. Locality

We adopt the notions of locality used in [10], [16], [43], and [44] in which it measures how well neighboring genotypes correspond to neighboring phenotypes. Hence, high locality means that small changes in genotype correspond to small changes in phenotype, while low locality indicates the situation where small changes in genotype may often correspond to large changes in phenotype. Gottlieb and Eckert [10], [16] introduced *the mutation innovation* (MI) as a way of measuring locality. Given a parent and a mutant genotype, MI is equal to the phenotypic distance (d_p as defined above) between their corresponding phenotypes

$$\text{MI}(x, x^m) = d_p(T_x, T_{x^m}) \tag{7}$$

where x and x^m indicate parent and mutant genotypes, and T_x and T_{x^m} indicate their corresponding phenotypes.

To measure locality for each encoding, we generated 1000 random initial solutions in compliance with the used encoding, applied only a mutation operator (in each case, that used in the experiments reported earlier in the paper) to each of these solutions, and performed the experiment on selected OCST instances: In the case of the OCST, the selected instances were Palmer24, Berry35U, and several randomly generated instances $(N = 10 \sim 100)$.

Table XI shows comparisons of locality among all encodings for selected OCST problems. Part of this table already appears in [51], along with similar for several other randomly generated OCST problems (for which the numbers generally echo the findings in the selection used here). $P(\text{MI} = 0)$, $E(\text{MI} \,|\, \text{MI} > 0)$ and $\sigma(\text{MI} \,|\, \text{MI} > 0)$ echo their definition in [37]. Respectively, they represent the percentage of cases for which $\text{MI} = 0$, the expected value of MI in cases where $\text{MI} > 0$, and the standard deviation in the latter figure, each of these being estimated by the empirical results.

NetKey and NB show rather higher $P(\text{MI} = 0)$ than the other encodings. In the NetKey case, the mutation operator results in a change of sorting order of exactly two genes. But if these genes were not "used" in parent phenotype, the child phenotype will be no different (this may also be the case if one or more of these genes *was* used in the parent. Note that we can expect $P(\text{MI} = 0)$ to increase for NetKeys with network size (which is observed), since the proportion of "unused" genes in the NetKeys genotype $(1-2/n)$ rises with n. The NB encoding displays even higher redundancy; this is partly for the reasons already discussed for NetKeys. Additionally, owing to its reliance on Prim's algorithm it displays a strong heuristic bias (manifest here as an increase in the tendency for a phenotype to "resist" change in response to mutation of the genotype) toward a specific spanning tree [13]. Therefore (especially in the Berry35U instance), NB shows a higher $P(\text{MI} = 0)$ value (93.5%) than the other encodings.

Meanwhile, even though the EWD strategy (in all instantiations used in this article) is a redundant encoding, its value of $P(\mathrm{MI} = 0)$ is negligible (when compared with NetKeys and NB). In the Prüfer and ES encodings, $P(\mathrm{MI} = 0) = 0$ by definition, and verified here empirically as a test of our implementations; but with EWD, whose design does not rule out the possibility of some "noninnovative" mutation steps, $P(\mathrm{MI} = 0)$ is low in one case and zero in another. This suggests that the inherent redundancy in EWD is not likely to inhibit search progress, perhaps owing to the way redundant phenotypes are distributed in the genotype space (which is under exploration in ongoing work).

Moving on to discuss $E(\mathrm{MI} \,|\, \mathrm{MI} > 0), \sigma(\mathrm{MI} \,|\, \mathrm{MI} > 0)$ and $\mathrm{Max}(\mathrm{MI})$, we can arguably expect "good" locality to be indicated by low values for these, while noting that a moderate or high standard deviation or Max value may not necessarily be a bad sign, since it indicates a potentially beneficial exploratory bias. The ES encodings have idealized values, since every mutation causes precisely a unit change in the phenotype. EWD has rather weaker locality than the ESs and NetKeys, but roughly similar locality to Prüfer and NB (though this varies a great deal, see below). But, when considering the $P(\mathrm{MI} = 0)$ values in conjunction with this, it is very difficult to distinguish which of EWD and NetKey (for example) appears to have better locality properties. When we look at $\sigma(\mathrm{MI} \,|\, \mathrm{MI} > 0)$, it appears that NetKeys, the ESs, and EWD are much more stable than Prüfer and NB. It is worth noting that, in the case of Berry35U, all distance weights are the same. In this situation, the NB encoding can only encode stars [40]; the value $E(\mathrm{MI} \,|\, \mathrm{MI} > 0) = 33$ in this case arises because the distance between each pair of possible stars in a 35-node network is 33.

Overall, given that we are seeking potential explanations for the strong performance of EWD, we can find tentative clues in the fact that it maintains a good level of $P(\mathrm{MI} = 0)$ (in common with the ESs), while also achieving more exploration than the ESs by having higher, but arguably not too high, values of $E(\mathrm{MI}|\mathrm{MI} > 0)$. Two of the other encodings which also have similarly high values for the latter (NB and NetKeys) let themselves down by many wasted mutations (high values of $P(\mathrm{MI} = 0)$); the third, Prüfer, generally presents worse locality than EWD.

C. Heritability

Heritability is a measure of the interaction between an encoding and a crossover operator. Julstrom [18] defined it, in the spanning trees context, as the number of edges in the offspring's spanning tree that appeared in either parent's tree. We define the heritability as a similar way.

- *Heritability* is the number of edges in the offspring's spanning tree that appeared in either parent's tree

$$d_h(P_i, P_j, O) = |(P_i \cup P_j) \cap O| \tag{8}$$

where P and O represent parent and offspring.

In Table XII, (again part of a table that appears in [51]), we compare the heritability values based on 1 000 crossover operations for the same choice of encodings and problem instances as used in the locality experiments reported here. In each case, the encoding/crossover pairing is the best-performing such pairing for the encoding concerned based on preliminary experiments (Prüfer uses two-point crossover, NB one-point crossover, NetKey uniform crossover, ESs their specialized crossover operators, and EWD uses adjacent node crossover.) The table gives the mean heritability for each encoding, where a high d_h indicates high or "good" heritability.

TABLE XII
HERITABILITY VALUES BASED ON RANDOMLY GENERATING 1000 GENOTYPES AND APPLYING CROSSOVER ONCE TO EACH

Heritability	Prüfer	NB	NetKey	ESWH	ESWOH	EWD
Palmer24	15.72	19.08	18.79	23.00	23.00	16.77
Berry35U	22.27	26.63	27.90	34.00	34.00	17.77
Rand70	43.94	57.01	56.17	69.00	69.00	60.04
Rand80	49.96	64.78	64.32	79.00	79.00	67.93
Rand90	55.57	73.46	72.73	89.00	89.00	77.51
Rand100	62.49	81.14	80.81	99.00	99.00	86.17

The ESs again show "ideal" results, with offspring only ever having a single edge included in neither parent. EWD generally exhibits higher heritability than the other (non-ES) encodings, while Prüfer exhibits the poorest heritability overall.

D. Bias Toward the MST

1) Bias in the Encoding: We first examine the bias of each encoding, without reference to the effect of operators. For each of eight randomly generated sets of Euclidean distance weights (30, 40, 50, 60, 70, 80, 90, and 100 nodes, respectively), we generated 1000 random solutions using each encoding.

Table XIII summarizes the results, showing the means and standard deviations of $d_{\mathrm{mst-pop}}$ for each encoding and each problem size. We know *a priori* that Prüfer and Netkeys are unbiased encodings (as is also ESWOH, which we omit to avoid clutter, and whose results were very similar to those of Prüfer and Netkeys). Of these two, we arbitrarily choose the $d_{\mathrm{mst-pop}}$ values of the Prüfer encoding as our benchmark against which we estimate the level of bias of the other encodings. It is clear from the table that both NB and ESWH are biased toward the MST, and to a similar mean extent. In the 70-node case, for example, both NB and ESWH (although with rather high variance in the ESWH case) tend to favor trees that share half of their edges with the MST, while the unbiased encodings tend to visit trees that share only a third of their edges with the MST. What is most interesting to note, however, is that the EWD-based encodings exhibit at most a very slight bias. We would not expect any bias from CF-TCR, and the statistics support this statement. However, both dK-TCR and CB-TCR involve consideration of weights in the interpretation of an individual. In the case of dK-TCR, edges in the subgraph encoded by an individual are considered in order of ascending weight; in the case of CB-TCR, consideration of edge-weights comes into play whenever a cycle is encountered during interpretation of an individual. From these points alone, we would expect some level of bias toward the MST, and perhaps more bias in the case of dK-TCR than with CB-TCR. The results hint in this direction, however, it is clear that such bias is negligible, certainly much less than in the cases of NB and ESWH. We can understand this in the case of dK-TCR by noting that although the edges are considered in order of weight, the set of edges considered in the first place is an unbiased and usually small subset of all

TABLE XIII
BIAS OF ENCODINGS TOWARDS THE MST: THE MEAN AND THE STANDARD DEVIATION OF THE METRIC $d_{\mathrm{mst-pop}}$ FOR A POPULATION OF 1000 RANDOMLY GENERATED INDIVIDUALS. A HIGHER VALUE INDICATES A LOWER BIAS

N	ES		Prüfer		NetKey		NB		EWD(CF-TCR)		EWD(dK-TCR)		EWD(CB-TCR)	
	std.	avg.	std.	avg.	std.	avg.	std.	avg.	std.	avg.	std.	avg.	std.	avg.
30	7.310	21.485	1.309	27.085	1.344	27.059	1.893	19.524	1.335	27.060	1.701	25.435	1.627	25.754
40	9.952	29.003	1.385	37.045	1.279	37.102	2.612	27.388	1.379	37.077	1.724	35.437	1.713	35.581
50	12.251	36.532	1.312	46.971	1.320	47.061	2.874	34.523	1.357	47.038	1.812	45.292	1.761	45.367
60	15.509	43.380	1.378	56.978	1.327	57.036	3.131	43.739	1.366	57.033	1.895	55.225	1.793	55.420
70	17.717	51.198	1.336	67.029	1.393	67.024	3.288	50.360	1.391	67.071	1.815	65.208	1.788	65.358
80	20.853	58.374	1.413	77.052	1.398	77.017	3.438	58.782	1.313	77.043	1.856	75.213	1.844	75.280
90	22.246	66.506	1.406	86.978	1.397	87.017	3.483	67.899	1.435	87.038	1.852	85.258	1.839	85.334
100	26.172	73.320	1.360	96.974	1.325	97.047	3.659	76.273	1.396	96.982	1.862	95.184	1.975	95.138

possible edges. In the case of CB-TCR, the likely explanation would be the relative infrequency with which cycles are introduced during an individual's interpretation, however, we have yet to collect the information to confirm that.

The absence (or weakness) of bias in the EWD instantiations suggests a valuable flexibility in the EWD strategy; that is, with no *a priori* bias toward problems whose solutions are close to the MST, the encoding is applicable in the first instance to any tree-based problem (unless we already know properties of the problem that suggest a specific different approach would be better). Further reasons for the EWD instantiations' good performance on the problems studied may instead be found in the bias of the EWD-specific operators; we address this question next.

2) Bias in the Operators: To examine the effect of operator biases, we used two of the larger of the test problems used in the "encoding bias" experiment (with $N = 70$ and $N = 80$). In most cases, we begin from the same initial populations (of 1000 individuals) used in the corresponding experiments in the previous section; however, in the case of ES (where we test the "with heuristics" variant), this time we do not use heuristic initialization—this enables us to get a better feel for the operator bias of ES with heuristics in comparison with the operator bias for the other methods. In each case, we ran a straightforward EA *without* selection (equivalently, using random selection) and using the crossover operator only. That is, to produce population $k+1$ from population k, the following was repeated 1000 times: two individuals are chosen at random (with replacement) from population k, they are subjected to the crossover operator, and the resulting child is placed into population $k + 1$.

In Fig. 8, we see the results, with a similar pattern for each of the two problem sizes. The plots show, for each of several methods, the value of $d_{\mathrm{mst-pop}}$ over time from generation 1 to generation 100. Prüfer numbers again provide our baseline, establishing the $d_{\mathrm{mst-pop}}$ level for an unbiased encoding. $\mathrm{EWD(CF-TCR+CGPX)}$ consistently shows the same minor level of bias; we already know that the TCRs show little or no bias (almost certainly no bias in the case of CF-TCR), however, it is mildly surprising to learn that CGPX introduces little or negligible bias. Although edge weights are occasionally considered in the CGPX procedure, we can infer that this is too infrequent to lead to a measurable effect. Meanwhile, both CB-TCR + CGPX and dK-TCR + CGPX show a stronger, but still relatively gentle bias. We suspect this is explained by the rather weak source of bias in CGPX working together with

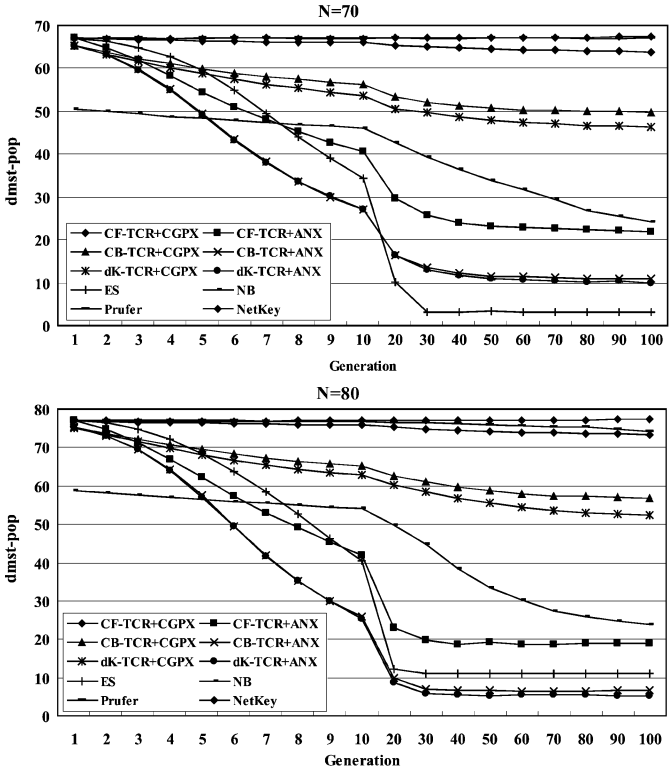

Fig. 8. Plots show the mean of $d_{\mathrm{mst-opt}}$ over 100 generations of an EA running with random selection and crossover only, where each encoding has its associated (or specifically indicated) crossover operator. Results are presented for 70- and 80-node problem instances.

the indications of slight bias seen (in the "bias in the encoding" experiments) in both CB-TCR and dk-TCR.

Toward the other extreme, we find that each of ES, NB, and EWD strategies using ANX show a strong bias. NB has the weakest bias of these five methods, while the relative degree of bias between CF-TCR + ANX, CB-TCR + ANX, and dK − TCR + ANX increases in that order, reflecting the contribution of bias from the TCRs (see previous section), although the difference between dK − TCR + ANX and CB-TCR + ANX is small. Meanwhile, the level of bias shown by ES with heuristics is similar to that of CB-TCR+ANX and dK−TCR+ANX, but ES seems to lose "momentum" after many generations. Toward the middle of the evolution period in these experiments, in both cases ES begins to induce a sharper decrease in $d_{\mathrm{mst-pop}}$ than any of the other methods, but then $d_{\mathrm{mst-pop}}$ quickly stabilizes

at a nonzero value, unable to proceed further toward the MST. The latter is also the case for the other strong-bias encodings, but these experiments suggest that the ES heuristic crossover operator is less able than the EWD instantiations in exploiting diversity in the population. ANX clearly presents more opportunity than CGPX for edge-weights to come into consideration; meanwhile, we have already seen the minor hints of bias in CB-TCR and (more so) in dK-TCR. These observations would seem to underpin the fact that CF-$\mathrm{TCR}+\mathrm{ANX}$, CB-$\mathrm{TCR}+\mathrm{ANX}$, and $d\mathrm{K}-\mathrm{TCR}+\mathrm{ANX}$ show, respectively, increasing degrees of bias toward the MST. Given that a bias toward the MST is helpful for DCMST and OCST problems (and we infer so for the q-MST, but cannot yet confirm), this would help explain the fact that the generally better-performing EWD instantiations on these problems were CB-TCR+ANX and $d\mathrm{K}-\mathrm{TCR}+\mathrm{ANX}$.

VIII. Conclusion

We have presented an encoding strategy for EAs applied to constrained spanning tree-based combinatorial optimization problems, and have tested it on the DCMST, OCST and q-MST. The EWD strategy encodes a tree as a linear string of node identifiers, which is then interpreted as a set of edges, which is further interpreted as a tree by a specified TCR. EWD is highly amenable to the design of new, specialized and domain-informed operators (since the underlying encoding is close to a directly represented set of edges), it can handle degree constraints in a straightforward and computationally simple way (although this is not the case for all TCRs), and it is modular in the sense that different TCRs can be used for different classes of problem, which represents another way to inform the approach with specialist problem and domain knowledge.

The performance of EWD was generally found to be strong in comparison with the best of previous approaches so far. We provided some empirical analyses aimed at understanding the relative performance of the encodings used. This was done by providing some results from [51] concerning locality and heritability, combined with new results on the degree of bias toward the MST that is exhibited by each encoding, and also each of the interesting encoding/operator combinations.

It is generally accepted that high locality and high heritability will potentially lead to good performance. However, certain EWD instantiations tend to show less high locality and heritability than ES, but higher locality and heritability than the other encodings studied. The particular instantiations which work most well on these problems are also those which exhibit a strong level of bias toward the MST. We, therefore, conjecture that the generally strong performance of these EWD instantiations can be explained in terms of the combination of two things.

- A need (in the context of typically complex combinatorial landscapes) to maintain a good level of exploration. We infer this need from the fact that good performance among the encodings generally correlated with *moderate* levels of locality and heritability.
- An appropriate bias in the encoding and operators toward regions of the landscape which share properties in common with good local optima. We infer this need from the bias analyses, confirming that the better performing methods showed a strong bias toward the MST.

EWD's locality, heritability, and (in certain instantiatons) bias are enough to ensure efficient exploitation and general progress, but are also at a level that maintains a reasonable degree of exploration (as measured by mutation innovation). We conclude that the EWD strategy may be a preferred approach for real-world oriented instances of tree-based combinatorial problems (especially larger ones, where exact methods could never be appropriate). Further, where it is known that good solutions are likely to be similar to the MST, the CB-TCR, and dK-TCR strategies are recommended, especially with operator ANX, while CGPX may be more helpful when it is known that optimal solutions do not share much in common with the MST. Further work is due to investigate more closely the relationship between the levels of bias in an encoding and associated operators, and the details of the distribution of MST-like trees in optimal regions.

Acknowledgment

The authors would like to thank F. Rothlauf for many very helpful discussions and comments. They also thank the anonymous reviewers for their constructive and helpful remarks.

References

[1] F. N. Abuali, R. L. Wainwright, and D. A. Schoenefeld, "Determinant factorization: A new encoding scheme for spanning trees applied to the probabilistic minimum spanning tree problem," in *Proc. 6th Int. Conf. Genetic Algorithms*, L. J. Eshelman, Ed., 1995, pp. 470–477.

[2] S. Arora, C. Lund, R. Motwani, M. Sundan, and M. Szegedy, "Proof verification and the hardness of approximation problems," in *Colloquium on Computational Complexity Report TR98-008*. Trier, Germany: Univ. Trier, 1998.

[3] A. Assad and W. Xu, "The quadratic minimum spanning tree problem," *Naval Research Logistics*, vol. 39, pp. 399–417, 1992.

[4] J. C. Bean, "Genetic algorithms and random keys for sequencing and optimization," *ORSA J. Comput.*, vol. 6, no. 2, pp. 154–160, 1994.

[5] B. Boldon, N. Deo, and N. Kumar, "Minimum-weight degree-constrained spanning tree problem: Heuristics and implementation on an SIMD parallel machine," *Parallel Comput.*, vol. 22, pp. 369–382, 1996.

[6] H. Chou, G. Premkumar, and C. H. Chu, "Genetic algorithms for communications network design-an empirical study of the factors that influence performance," *IEEE Trans. Evol. Comput.*, vol. 5, no. 3, pp. 236–249, 2001.

[7] B. Dengiz, F. Altiparmak, and A. E. Smith, "Local search genetic algorithm for optimal design of reliable networks," *IEEE Trans. Evol. Comput.*, vol. 1, no. 3, pp. 179–188, 1997.

[8] N. Deo and N. Kumar, "Computation of constrained spanning trees: A unified approach," *Lecture Notes in Economics and Mathematical Systems*, vol. 450, pp. 194–220, 1996.

[9] M. Ebner, P. Langguth, J. Albert, M. Shackleton, and P. Shipman, "On neutral networks and evolvability," in *Proc. 2001 CEC*, 2001, pp. 1–8.

[10] C. Eckert and J. Gottlieb, "Direct representation and variation operators for the fixed charge transportation problem," in *Lecture Notes in Computer Science*, vol. 2439, Proc. PPSN VII, 2002, pp. 77–87.

[11] B. Freisleben and P. Merz, "A genetic local search algorithm for solving symmetric and asymmetric travelling salesmen problems," in *Proc. of IEEE Int. Conf. Evol. Comput.*, 1996, pp. 616–621.

[12] M. R. Garey and D. S. Johnson, *Computers and Intractability, A Guide to the Theory of NP-Completeness*. San Francisco, CA: Freeman, 1979, pp. 73–80.

[13] T. Gaube and F. Rothlauf, "The link and node biased encoding revisited: Bias and adjustment of parameters," in *Proc. EvoWorkshop*, vol. 2037, Lecture Notes in Computer Science, 2001, pp. 1–10.

[14] M. Gen and R. Chen, *Genetic Algorithms and Engineering Design*. New York: Wiley, 1997.

[15] J. Gottlieb, B. A. Julstrom, G. R. Raidl, and F. Rothlauf, "Prüfer numbers: A poor representation of spanning trees of evolutionary search," in *Working Papers in Information Systems*. Bayreuth, Germany: Univ. Bayreuth, 2000.

[16] J. Gottlieb and C. Eckert, "A comparison of two representations for the fixed charge transportation problem," in *Lecture Notes in Computer Science*, vol. 1917, Proc. PPSN VI, 2000, pp. 345–354.

[17] T. C. Hu, "Optimum communication spanning trees," *SIAM J. Comput.*, vol. 3, no. 3, pp. 188–195, 1974.
[18] B. A. Julstrom, "The blob code: A better string coding of spanning trees for evolutionary search," in *Proc. Genetic and Evol. Comput. Conf.*, 2001, pp. 256–261.
[19] B. Julstrom and G. Raidl, "Initialization is robust in evolutionary algorithms that encode spanning trees as sets of edges," in *Proc. ACM Symp. Appl. Comput.*, 2002, pp. 547–552.
[20] C. Palmer and A. Kershenbaum, "An approach to a problem in network design using genetic algorithms," *Networks*, vol. 26, pp. 151–163, 1995.
[21] J. Knowles and D. Corne, "A new evolutionary approach to the degree-constrained minimum spanning tree problem," *IEEE Trans. Evol. Comput.*, vol. 4, no. 2, pp. 25–134, 2000.
[22] J. Knowles and R. A. Watson, "On the utility of redundant encodings in mutation-based evolutionary search," in *Lecture Notes in Computer Science*, vol. 2439, Proc. PPSN VII, 2002, pp. 88–98.
[23] M. Krishnamoorthy, A. Ernst, and Y. Sharaiha, "Comparision of algorithms for the DC-MST," *J. Heuristics*, vol. 7, pp. 587–611, 2001.
[24] J. B. Kruskal, "On the shortest spanning tree of a graph and the travelling salesman problem," in *Proc. Amer. Math. Soc.*, vol. 7, 1956, pp. 48–50.
[25] B. Manderick, M. de Weger, and P. Spiessens, "The genetic algorithm and the structure of the fitness landscape," in *Proc. 4th Int. Conf. Genetic Algorithms*, 1991, pp. 143–150.
[26] P. Merz and B. Freisleben, "Fitness landscapes, memetic algorithms, and greedy operators for graph bipartitioning," *Evol. Comput.*, vol. 8, no. 1, pp. 61–91, 2000.
[27] S. C. Narula and C. A. Ho, "Degree-constrained minimum spanning tree," *Comput. Oper. Res.*, vol. 7, pp. 239–249, 1980.
[28] C. C. Palmer and A. Kershenbaum, "Representing trees in genetic algorithms," in *Proc. IEEE Int. Conf. Evol. Comput.*, 1994, pp. 379–384.
[29] ——, "An approach to a problem in network design using genetic algorithms," *Networks*, vol. 26, pp. 151–163, 1995.
[30] C. C. Palmer, "An approach to a problem in network design using genetic algorithms," Ph.D. dissertation, Polytechnic Univ., Ann Arbor, MI, 1994.
[31] C. H. Papadimitriou and M. Yannakakis, "Optimization, approximation, and complexity classes," *J. Comput. Syst. Sci.*, vol. 43, pp. 425–440, 1991.
[32] D. Peleg and E. Reshef, "Deterministic polylog approximation for minimum communication spanning trees," in *Proc. ICALP*, vol. 1443, Lecture Notes in Computer Science, 1998, pp. 670–681.
[33] S. Picciotto, "How to encode a tree," Ph.D. dissertation, Univ. California, San Diego, CA, 1999.
[34] P. Piggott and F. Suraweera, "Encoding graphs for genetic algorithms: An investigation using the minimum spanning tree problem," in *Progress in Evolutionary Computation*. Berlin, Germany: Springer-Verlag, 1995, pp. 305–314. LNAI 956.
[35] R. Prim, "Shortest connection networks and some generalizations," *Bell Syst. Tech. J.*, vol. 36, pp. 1389–1401, 1957.
[36] G. R. Raidl, "An efficient evolutionary algorithm for the degree-constrained minimum spanning tree problem," in *Proc. IEEE Int. Conf. Evol. Comput.*, 2000, pp. 104–111.
[37] ——, "Empirical analysis of locality, heritability and heuristic bias in evolutionary algorithms: A case study for the multidimensional knapsack problem," Vienna Univ. Technol., Vienna, Austria, Tech. Rep. TR-186-1-04-05, 2004.
[38] G. R. Raidl and B. Julstrom, "Edge-sets: An effective evolutionary coding of spanning trees," *IEEE Trans. Evol. Comput.*, vol. 7, no. 3, pp. 225–239, 2003.
[39] C. R. Reeves and T. Yamada, "Genetic algorithms, path relinking, and the flowshop sequencing problem," *Evol. Comput.*, vol. 6, pp. 45–60.
[40] F. Rothlauf, D. Goldberg, and A. Heinzl, "Network random keys—A tree network representation scheme for genetic and evolutionary algorithms," *Evol. Comput.*, vol. 10, no. 1, pp. 75–97, 2002.
[41] ——, *Representations for Genetic and Evolutionary Algorithms*. Berlin, Germany: Springer-Verlag, 2002. Number 104 in Studies on Fuzziness and Soft Computing.
[42] F. Rothlauf, J. Gerstacker, and A. Heinzl, "On the optimal communication spanning tree problem," in *Working Papers in Information Systems*: Univ. Mannheim, 2003.
[43] F. Rothlauf, "Population Sizing for the redundant trivial voting mapping," in *Working Papers in Information Systems*. Mannheim, Germany: Univ. Mannheim, 2003.
[44] F. Rothlauf and D. Goldberg, "Tree network design with genetic algorithms—An investigation in the locality of the prüfernumber encoding," in *Proc. Late Breaking Papers at the Genetic and Evol. Comput. Conf.*, 1999, pp. 238–244.
[45] B. Schindler, F. Rothlauf, and H. Pesch, "Evolution strategies, network random keys, and the one-max tree problem," in *Evoworkshops*. Berlin, Germany: Springer-Verlag, 2002, pp. 143–52.
[46] P. Schuter, "Artificial life and molecular evolutionary biology," in *Advances in Artificial Life*, F. Moran, Ed. Berlin, Germany: Springer-Verlag, 1995, pp. 3–19.
[47] B. Sendhoff, M. Kreutz, and W. V. Seelen, "A condition for the genotype-phenotype mapping: Casualty," in *Proc. 7th Int. Conf. Genetic Algorithms*, 1997, pp. 73–80.
[48] R. Shipman, M. Shackleton, and L. Harvey, "The use of neutral genotype-phenotype mappings for improved evolutionary search," *British Telecom Technol. J.*, vol. 18, no. 4, pp. 103–111, 2000.
[49] S.-M. Soak, D. Corne, and B.-H. Ahn, "A new encoding for the degree constrained minimum spanning tree problem," in *Lecture Notes in Computer Science*. Berlin, Germany: Springer-Verlag, 2004, vol. 3213, Proc. KES, pp. 952–958.
[50] ——, "A recent encoding achieves progress on optimum communication spanning tree benchmark instances," *Operations Research Letter*, 2004. In review.
[51] ——, "On a property analysis of representations for constrained spanning tree problems," in *Proc. 7th Int. Conf. Artif. Evol.*, 2005. In press.
[52] C. Tzoppe, F. Rothlauf, and H. J. Pesch, "The edge-set encoding revisited: On the bias of a direct representation for trees," in *Working Paper in Information Systems*. Mannheim, Germany: Univ. Mannheim, Jan. 2004.
[53] L. D. Whitley, T. Starkweather, and D. Fuquay, "Scheduling problems and traveling salesman: The genetic edge recombination," in *Proc. 3rd Int. Conf. Genetic Algorithms*, 1989, pp. 133–140.
[54] B. Y. Wu, K. M. Chao, and C. Y. Tang, "Approximation algorithms for some optimum communication spanning tree problem," in *Lecture Notes in Computer Science*. Berlin, Germany: Springer-Verlag, 1998, vol. 1533, Proc. ISAAC, pp. 407–416.
[55] ——, "A polynomial time approximation scheme for optimal product-requirement communication spanning trees," *J. Algorithms*, vol. 36, pp. 182–204, 2000.
[56] W. Xu, "On the quadratic minimum spanning tree problem," in *Proc. 3rd Int. Conf. Genetic Algorithms*, J. Schaffer, Ed., 1995, pp. 141–148.
[57] G. Zhou and M. Gen, "A note on genetic algorithms for degree-constrained spanning tree problems," *Networks*, vol. 30, pp. 91–95, 1997.
[58] G. Zhou and M. Gen, "An effective GA approach to the quadratic minimum spanning tree problem," *Comput. Oper. Res.*, vol. 25, no. 3, pp. 229–237, 1998.
[59] G. Zhou and M. Gen, "Genetic algorithm approach on multi-criteria minimum spanning tree problem," *Eur. J. Oper. Res.*, vol. 114, pp. 141–152, 1999.

Sang-Moon Soak was born in Gyeong-San, Kyung-Buk Province, Korea, in 1973. He received the B.S. and M.S. degrees in logistics engineering from Korea Maritime University, Busan, Korea, in 1998 and 2000, respectively. He is currently working towards the Ph.D. degree in the Department of Mechatronics, Gwangju Institute of Science and Technology (GIST), Gwangju, Korea.

From January to December 2004, he was Visiting Scholar in the Department of Computer Science, University of Exeter, Exeter, U.K. His research interests are in evolutionary computation, solution representation for evolutionary algorithms, tree-based network optimization problem.

David. W. Corne received degrees in mathematics and artificial intelligence from the University of Edinburgh, Edinburgh, U.K.

He was a Research Associate at the Department of Artificial Intelligence, University of Edinburgh for six years, before obtaining a Lectureship at the University of Reading in 1995. He moved to the University of Exeter, Exeter, U.K., in 2003, where he took up a Chair in Computer Science, and is now Professor of Computer Science at Heriot Watt University, Edinburgh. He maintains research interests in evolutionary computation, multiobjective optimization, bioinformatics, telecommunications, and general aspects and applications of nature-inspired computation.

Byung-Ha Ahn received the B.S. degree from the Korea Air Force Academy, Chongju, Korea, in 1965 and the M.S. and Ph.D. degrees in industrial engineering from Korea Advanced Institute of Science and Technology, Taejon, in 1977 and 1980, respectively.

Since 1995, he has been a Professor with the Department of Mechatronics, Gwangju Institute of Science and Technology, Gwangju, Korea. His research interests include system development and optimization, reliability analysis of large-scale systems, man-machine interfaces, and control for intelligent transportation systems.