Evolving Efficient Theme Park Tours

Leonard Testa[1], Albert C. Esterline[1] and Gerry V. Dozier[2]

[1] North Carolina A &T State University, Greensboro, North Carolina, USA
[2] Department of Computer Science and Engineering, Auburn University, Alabama, USA

In this paper, we describe the use of an evolutionary algorithm (EA) for the problem of visiting rides in a theme park. Generating a tour of the rides in a theme park is an instance of the time dependent traveling salesman problem (TDTSP) in which the cost of visiting any two rides depends not only on the distance between the rides, but also the time to wait in line and ride each ride. We describe the implementation of the evolutionary algorithm, including EA's ability to satisfy the constraint of scheduling lunch in the theme park. We present several test problems and compare both the running times and costs of tours generated by the EA to those of a previously described dynamic programming heuristic.

1. Introduction

Millions of people annually include visits to theme parks such as Walt Disney World in Orlando, Florida as part of their vacation plans. These theme parks regularly attract so many customers that an entire industry exists whose sole purpose is to write guidebooks that help plan such visits [6][7][8]. Those guidebooks typically cover every aspect of the visit, including where to stay, where to eat, and how to see the more popular rides while avoiding long waits in line. This last problem, of seeing the most popular rides in the shortest amount of time, is an instance of the time dependent traveling salesman problem (TDTSP) [9] and is thus amenable to computer-generated solutions. This paper describes a Java-based system available on the World Wide Web that uses an evolutionary algorithm (EA) to help plan efficient visits to Walt Disney World's Magic Kingdom theme park.

The TDTSP is similar to the time dependent vehicle routing problem (TDVRP), which is a variant of traveling salesman problem (TSP) where the amount of time it takes to travel from one city to another varies depending on the time of day. This varying travel time enables modeling of real world conditions such as heavy traffic, road construction or repair, and accidents [9]. In our theme park problem the salesman is replaced with a customer, cities are replaced with rides, and heavy traffic is replaced with large crowds. The solution, an ordered list of rides, is called a touring plan (TP) [6]. Also, most TDTSP and TDVRP solutions omit one essential fact: the vehicle drivers and salesmen have to eat [4][10][11]. Theme park customers have to eat, too, so our solution gives its users the ability to specify at which theme park restaurants they would consider eating lunch. Further, users can specify whether the time to eat lunch is a rigid (e.g., "Meet me at the Liberty Tree Tavern at noon.") or flexible (e.g., "Let's eat around twelve-thirty.") constraint. The system then chooses a restaurant from the list of possible restaurants such that the overall touring plan cost is minimized.

If the lunchtime constraint specification can be based on fuzzy membership [15] then the TDTSP becomes a fuzzy constrained optimization problem [17]. These types of constrained problems have been successfully solved by EAs [17].

EAs are robust, adaptive, search procedures based on natural selection [20]. Unlike genetic algorithms (GAs), which traditionally have represented individuals as binary strings, EAs make use of a variety of data structures to represent individuals [20]. In contrast to most search algorithms, which operate on a single candidate solution (CS), EAs operate on a population of individuals where each individual represents a CS. After an initial population of randomly generated individuals has been produced, each in-

dividual in the population is assigned a fitness by an evaluation function. Individuals within the population are selected to be parents based on their fitness. The parents create offspring (which are also individuals that represent CSs) by either mutating themselves slightly (asexual procreation), mating with other parents (sexual procreation) or both. The offspring are then evaluated to determine their fitness and are possibly added to the population, usually replacing lesser-fit individuals in order to keep the population size constant. This process of selecting parents and allowing them to procreate based on their fitness is continued until an individual representing an optimal or near optimal CS has been evolved.

Evolutionary search has been successfully used to solve a wide variety of problems in such areas as machine learning, pattern recognition [1], communication networks [21], constrained optimization and robotics [20], to name a few. EAs are well-suited to the TDTSP and our theme park problem because they are adaptive systems that have proven effective at solving complex optimization problems [1]. Like simulated annealing and tabu search, EAs can be queried at any time for the best solution found so far [20].

The remainder of this paper is organized as follows. Section 2 provides a brief overview of TDTSPs and methods previously used to solve instances of this class of TSPs. In Section 3, an Evolutionary Tour Planner that is better suited to theme park tour planning than the systems described in [12] and [22] is described in detail, including it's implementation on the World Wide Web. In Section 4, a test suite of its problems is described and in Section 5 the performance of the Evolutionary Tour Planner is compared and contrasted with the approaches proposed in [22] and [12]. A number of conclusions are stated in Section 6 and some directions for future research are discussed in Section 7.

2. Overview of the Time-Dependent Traveling Salesman Problem

The typical TSP starts with a list of N cities labeled $(1, 2, \ldots, N)$ and an $N \times N$ cost matrix D in which the value D_{ij} is the distance from city i to city j [11]. The goal is usually stated as finding the minimal cost tour of the cities in which each city is visited exactly once and the salesman returns to the starting city at the end of the tour [4][5]. The TDTSP is a special instance of the TSP where the salesman must still visit each city, but the cost of traveling from city i to city j depends on both the distance matrix D and the time of day the travel takes place [10]. The additional cost associated with the time of day can be computed by first dividing the day into t discrete timeslices of fixed duration, then constructing a cost matrix W where W_{ijk} is the cost of traveling from city i to city j at timeslice k [9].

Although the TSP and TDTSP are both NP-hard [11], published results for TDTSP problems seem to indicate that it is a much more difficult problem than the TSP. While exact solutions to TSPs involving several thousand cities have been reported [26], exact solutions to TDTSPs have been reported for only a few dozen cities [24]. Typically, these exact solutions also involve restrictions on the problems solved, such as requiring the travel time from city to city to be completed in a single time period [11][24]. Because the TDTSP is NP-hard, heuristics are usually combined with other methods to generate near-optimal solutions in reasonable amounts of time [11].

Malandraki and Daskin [9] used a mixed integer linear programming (MILP) model combined with several variations of the nearest-neighbor heuristic [13] on randomly generated TDTSP problems of up to 25 cities. Vander Wiel and Sahinidis [11] use a different MILP formulation coupled with a time-dependent Lin-Kernighan heuristic which produced solutions within 4.4% of the optimum known solution, on average, for TDTSP problems of up to 100 cities.

Malandraki and Dial [22] implemented a restricted dynamic programming (DP) algorithm, based on the nearest neighbor heuristic, that outperformed the MILP model described in [9] for TDTSPs of up to 55 cities. The restricted DP heuristic retained at each stage only a user-defined subset of all possible partial tours, which reduced the amount of memory and CPU resources required by exact DP solutions. The DP heuristic performed better than the MILP model when as few as 100 partial tours were retained at each stage.

Ahmadi [12] used a hybrid approach to generate optimal TPs for theme park customers. First, a

neural network model called the Ride Capacity Model (RCM) was built that analyzed historical data on customer arrivals and departures at each ride to determine that ride's observed capacity to handle customers. The output of the RCM was used as input into a MILP-based Capacity Management Model (CMM), along with the data on which rides customers tended to go to immediately after riding some other ride (the customer's transition patterns). With this data the CMM was able to determine the optimal passenger capacity of each ride at different times of the day, based on each ride's operating expenses, the perceived value of the ride to the customer, and the maximum wait in line at each ride. Output of the RCM was also used to feed a MILP-based Flow Pattern Model (FPM). The FPM was created to help the theme park management modify the customers' observed transition patterns so that large variations in each ride's wait time in line throughout the day was reduced. Like the CMM, the FPM also helped management set the number of customers each ride could service. Finally, the output of the CMM and FPM was fed into a MILP-based Tour Design Model (TDM), whose goal was to maximize the number of rides each customer visited. The TDM produced optimal TPs in two stages: initially the day was divided into time intervals (e.g., morning, afternoon, and evening). The task of choosing which rides to visit during each time interval, called the Ride Selection Problem (RSP), was solved first, using a variation on the multiple-choice knapsack problem. Once rides were grouped into time intervals, the TDM had to find the order of rides in each interval to minimize the customer travel time. This problem, known as the Ride Visiting Order problem (RVP), was solved using space filling curves. The author stated that this hybrid approach to generating TPs took 18 minutes on average with a maximum time of 37 minutes, running on a 66-Mhz HP Vectra 486 machine.

3. The Evolutionary Tour Planner

In this section, we describe the construction of the EA in detail. Section 3.1 is an overview of the entire algorithm. Section 3.2 details the cost (i.e. fitness) function used by the EA. Section 3.3 describes the various operators we evaluated for the EA, including the one we selected. Section 3.4 gives the algorithm for scheduling lunch with both a rigid or flexible time constraint, and section 3.5 describes the optimal touring plan generated by the algorithm.

3.1. Overview

Previous research [5] has shown that EAs alone do not perform as well as those incorporating some sort of local search heuristic. Therefore, our EA starts by creating an initial population of TPs using a simple time-dependent nearest

```
Function TPGEN(R, s, t_a, t_d, t_w, t_y, t_l, L, b_l, b_p)
 Begin
    Initialize timer to s seconds
    Initialize pool P with time-dependent Nearest-Neighbor heuristic using R
    Calculate cost c for each TP in P:  c_q = Cost( q ) for all q in P.
    While timer has not expired do
          Select a TP p from P using binary tournament selection
          Let o = Mutate( p )
          Let c_o = the cost of o
          Let j = the TP in P with the highest cost c_j
          If c_o < c_j and o is not already in P then
               Delete j from P
               Insert o into P
    Done

    If customer wants to schedule lunch then
      Foreach TP i in P do
          Insert lunch information into i
          Recalculate c_i with new lunch information
      Done
    Endif

    Return the TP i_min in P having lowest cost c_i min
 End
```

Fig. 1. The TPGEN Function.

neighbor heuristic [11]. The complete algorithm for creating TPs is shown in Figure 1.

The algorithm accepts the following input parameters:

R: the list of rides the customer wants to visit.

s: the number of seconds the customer wants the algorithm to run.

t_a: the time of day the customer will arrive at the park.

t_d: the time of day the customer will depart the park.

t_w: the weekday the customer will visit the park.

t_y: the time of year the customer will visit the park.

t_l: the time the customer indicated s/he wants to eat lunch.

L: the list of restaurants the customer will consider eating at.

b_l: a boolean flag indicating whether the customer indicated t_l was the exact time to eat lunch or an approximation.

b_p: a boolean flag indicating whether the customer is eligible for early entry into the park.

The elements of R are integers from 0 to 36, corresponding to the 36 rides in the park plus the end of Main Street. The elements of L are integers from 37 to 49, corresponding to the 13 restaurants in the park. Through our web site, the customer can choose a specific restaurant to eat lunch at, or can ask the EA to consider a list of restaurants. If L contains only one element, then the customer has indicated he will eat lunch at that restaurant. If L contains more than one element, the EA will schedule lunch at the restaurant in L that results in the lowest cost TP. If L is empty, then it is assumed the customer does not want to schedule lunch in the park. Also, the customer can use b_l to specify whether the time to eat lunch is an absolute or approximate time. Indicating an exact time, for example, might be useful if the customer is part of a large group that is touring the park independently but has agreed to meet for lunch at a certain time and place. Exact lunch times impose an additional constraint on the EA, since enough time must be allocated to allow the customer to walk from the last ride visited to the restaurant before the lunch time indicated. Customers with more relaxed schedules can tell the EA to fit lunch into the TP at approximately the time indicated, freeing the EA from having to satisfy the exact-time constraint. Finally, b_p indicates whether the customer is eligible for early entry into the park. Early entry is usually available only to theme park customers who are also staying at one of the Disney-owned hotels, and it is only permitted on certain days of the week. Customers eligible for early entry are usually allowed to enter the theme park one and one-half hours before the general public. This allows the early entry customers to tour the park more efficiently, since a smaller number of other customers are competing with them to visit the same rides.

The algorithm begins by initializing a timer to the number of seconds the customer specified the EA should run. When the timer expires the algorithm's main loop will stop. Next, a pool P of initial touring plans is created. The main data structure in each TP is an array of size R that holds a permutation of the elements of R [4]. For example, if a customer indicated s/he wanted to visit rides 1, 2, 4 and 32, a sample array might be $\{4, 2, 32, 1\}$. The number of individual TPs in P is fixed at 36, corresponding to the number of rides in the park plus the end of Main Street. When creating P, the EA ensures that at least one TP in P begins with each ride in R. A simple time-dependent nearest neighbor heuristic [5][11] is used to order the remaining rides in R for each TP in P.

3.2. Cost (Fitness) Function

After the initial pool P is created, each of the TPs in P must be evaluated to determine how long it would take a customer to complete the tour. This cost is measured in minutes, and consists of three parts: the time it takes to walk to the ride, plus the time the customer can expect to wait in line at each ride, plus the number of minutes the ride lasts. The cost function is shown in Figure 2.

The cost function accepts the following input parameters:

p: a TP in P

W: the wait time matrix

```
Function Cost( p, W, D, R, t_a, t_d, t_o, w_w, w_a)
  Begin
    Initialize tour_cost to 0
    Initialize clock to t_a, the customer's arrival time at the park

    If t_a = park opening time t_o then
      tour_cost = tour_cost + (walk time from end of Main Street to first ride * w_w)
      clock = clock + (walk time from end of Main Street to first ride * w_w)
    Else
      tour_cost = tour_cost + (walk time from park entrance to first ride * w_w)
      clock = clock + (walk time from park entrance to first ride * w_w)
    Endif

    For each ride r in p do
      tour_cost = tour_cost + (wait time at r * w_a)
      clock = clock + (wait time at r * w_a)

      tour_cost = tour_cost + ride time at r
      clock = clock + ride time at r

      If r is not the last ride in p then
        tour_cost = tour_cost + walk time from r to next ride in p
        clock = clock + walk time from r to next ride in p
      Endif
    Done

    Return tour_cost
  End
```

Fig. 2. The Cost Function.

D: the distance matrix

R: the ride time matrix

t_a: the time of day the customer will arrive at the park.

t_d: the time of day the customer will depart the park.

t_o: the time of day the park will open.

w_w: the relative preference the customer has for walking versus waiting in line.

w_a: the relative preference the customer has for waiting in line versus walking.

Appendix A lists the 36 rides and 13 restaurants in the Magic Kingdom, plus an entry for the park entrance and the end of Main Street. These additions are necessary because the touring plan will start at the end of Main Street if the customer arrives at the Magic Kingdom at the park's opening time, but at the entrance to the Magic Kingdom if the customer arrives later [7]. From this list a (symmetric) distance matrix D (Appendix B) was created containing the fixed cost in minutes of walking from one place in the list to any other[1]. For example, entry $D_{1,2}$ is 2, illustrating the assumption that the walking time from the Swiss Family Treehouse ride to the Jungle Cruise ride is two minutes.

The ride time array R (Appendix C) is a 50 row by 1 column matrix that contains the duration in minutes of each ride. For example, the Space Mountain ride lasts approximately 3 minutes, so its entry, $R_{28,1}$ is 3. The ride time matrix is similar to the service time in the TDVRP [9] and was obtained from [7]. The duration of each ride is assumed fixed.

The wait time matrix W (Appendix D) holds the amount of time a customer can expect to wait in line at each ride at different times of the day. W was obtained by first dividing the day into discrete intervals of fifteen minutes starting with the time the Magic Kingdom opened each day. Each row in W represents a different ride, restaurant or place in the Magic Kingdom. Each column represents a time interval. Next, the wait time in minutes at each time interval, rounded to the nearest minute, was estimated using measurements obtained during two visits to the Magic Kingdom and [7]. For example, $W_{29,2}$ holds the amount of time a customer could expect to wait in line if he arrived at the

[1] The distance and wait time measurements in this paper are approximate and were obtained during two visits to the Magic Kingdom in 1996 and 1997.

Space Mountain ride at 9:15 AM, $W_{29,3}$ holds the amount of time a customer could expect to wait in line if he arrived at 9:30 AM, and so on. If an attraction is not open at certain times of the day (e.g., rides that close early) the entries in W for that ride at those times are set to large integer values.

3.3. Operators

Once the initial pool P is generated and the cost of each tour in P is calculated, a TP p, in P is chosen for mutation. A binary tournament selection function [18] is used to choose p. Next, p is copied to an offspring o. A swap mutation operator is applied twice to o in which the positions of two different, randomly chosen rides in o are switched. For example, if o contains $\{1, 12, 32, 4, 15, 9\}$ and rides 12 and 15 are chosen for mutation, then o would contain the TP $\{1, 15, 32, 4, 12, 9\}$ after mutation. The swap mutation operator is applied twice to o. Mutation was chosen as the only operator, although several traditional crossover functions were also tested. Specifically, the authors implemented versions of the partially matched crossover, order crossover and cycle crossover functions from [1], the sub-inversion and sub-rotation operators from [3], and a new crossover function the authors named "modified Lebby crossover (MLX)[2]", described fully in Appendix E. Using the swap mutation operator described above consistently yielded superior results over each of these crossover functions, so it was retained. The superiority of mutation over traditional crossover operators for certain applications has been explored by others, including Fogel and Atmar [27], and Eshelman [16].

After swap mutation has been applied twice to o, the cost of the new TP in o is calculated. The decision of whether to retain o in pool P is made using a $(P + 1)$ reproduction [19] approach. Specifically, let q be the TP in P with the highest cost. The costs of o and q are compared, and if o has a lower cost than q, then q is deleted from P and o is retained.

3.4. Scheduling Lunch

The algorithm performs the selection, mutating, and (possible) insertion of offspring repeatedly until the timer has expired. Once the timer expires, the algorithm begins an optional second phase of scheduling lunch for the customer. The decision to defer the scheduling of lunch to outside of the main loop was based on the observation that inserting the lunch information into a TP takes a significant amount of computation compared to the rest of the main loop. Scheduling lunch inside the main loop would reduce the number of alternate TPs that could be generated before the timer expires, thus limiting the number of TPs considered by the EA. On a 133 Mhz Pentium computer with Netscape Navigator 3.01 the EA was able to execute about 245 iterations of the main loop in two minutes of real time, on average, when scheduling lunch inside the main loop for a 21 city problem. The TPs produced after 245 iterations had an average cost of 849 minutes. The EA was able to execute about 2,500 iterations in two minutes of real time, on average, of the main loop when the code to schedule lunch was removed from the main loop. The TPs produced, when the main loop was able to execute more, had an average cost of 736 minutes.

To schedule lunch effectively there are two constraints that must be satisfied. First, the customer must indicate a time to eat lunch. The customer may also indicate that that time is a rigid constraint (e.g., "I must eat lunch at noon at the Liberty Tree Tavern."). Alternatively, the customer may indicate that the time to eat lunch is more flexible, if she is less concerned with eating at a specific time (e.g., "Let's stop to eat around twelve-thirty, wherever we're at."). We believe that most of the guests at theme parks have relatively flexible scheduling needs for lunch.

The customer must also indicate a list of restaurants to consider eating lunch at, and it is the EA's responsibility to produce a TP that routes the customer to one of those restaurants at the lunch time indicated. On our web site, the customer has two mutually exclusive options for selecting restaurants: she can indicate a single,

[2] The Lebby crossover function, a composition of permutations, was described by Dr. Gerald Lebby of North Carolina A&T State University in a class taught there in 1996. To our knowledge, no other published results of the MLX operator's effectiveness exist.

specific restaurant to each lunch at, or she can create a non-empty list of restaurants to consider eating lunch at. The restaurant information is passed to the EA as a list L of restaurants. If L contains no elements, it is assumed the customer does not want to schedule lunch. If L contains a single restaurant, that restaurant is considered the customer's "definite" choice. If L contains more than one restaurant, the EA will try scheduling lunch at each restaurant l in L for each TP p in P to see which restaurant/TP combination yields the lowest cost TP. That lowest cost TP is retained.

The process of finding the correct position in a TP for lunch at some restaurant l is straightforward. Recall that a TP p contains an ordered list of rides, for example $\{1, 7, 4, 5, 9\}$. For any ride r_i , $i > 1$, the time the customer arrives at r_i can be calculated by the formula:

arrival time at r_i = arrival time at r_{i-1} +
wait time at r_{i-1} + ride time at r_{i-1}+
walk time to r_i

Using this formula, we first calculate the customer's arrival time at each ride in p. Next, we traverse p from the beginning to find the ride r_i in p with an arrival time greater than or equal to the time the customer wishes to eat lunch. Once r_i is found, the algorithm checks whether the customer has indicated that the lunch time is rigid. If the lunch time is not rigid, l is inserted in p between r_{i-1} and r_i. If the lunch time is rigid, the algorithm verifies the customer has enough time to travel from r_{i-1} to l in the time allotted. If enough time exists, the lunch restaurant is inserted into p between r_{i-1} and r_i. If there is not enough time to travel from r_{i-1} to l the algorithm will try in successive attempts to schedule lunch after r_{i-2}, r_{i-3}, and so on. To recalculate the cost of p with the lunch restaurant inserted in the path, the cost function treats the lunch restaurant as it would any ride: the amount of time it takes to be served at the restaurant corresponds to the ride wait time, and the time it takes to eat lunch corresponds to the ride's ride time.

It is worth noting that delaying the scheduling of lunch until after the main loop has completed led to a significant decrease in the average cost of the TPs generated. By deferring the scheduling of lunch to outside the main loop, the amount of computation inside the main loop was reduced. This allowed the main loop to execute faster, which generated more offspring. While the exact reasons for the improvement are the subject of further research, the authors believe that generating more offspring leads to a wider search of the solution space, allowing the EA to identify TPs of much lower cost. Also, the bulk of the work in scheduling lunch lies in finding the correct position between two rides in the TP to insert the lunch restaurant. The additional overhead to satisfy a rigid lunch time versus a flexible one is relatively small. Thus, it makes sense to delay the scheduling of both the rigid and flexible lunch constraints to outside the main loop. The authors also note that restaurants are well distributed throughout the Magic Kingdom, so customers never have to walk far to eat. This helps the EA by not imposing a large cost penalty for generating TPs that put the customer far away from a restaurant at lunch time. Whether the EA can produce TPs of similar quality when such penalties exist is the subject of further study. However, if more constraints are added to the EA, such as for multiple, user-defined breaks in the touring plan, it is doubtful that all of the constraints could be effectively postponed until after the main loop has completed. However, existing EA-based optimization problem solvers such as GENECOP II [20], which defer some constraints at each iteration, show that not every constraint need be satisfied with every generation.

3.5. The Generated Touring Plan

After the timer has expired and any lunch scheduling is completed, the web site prints the complete touring plan. Should the customer specify more rides than can be visited before the customer's indicated departure time, a warning message is displayed in the touring plan after the last ride visited before the departure time. This warning message notifies the user that their stated departure time has been reached, and suggests either leaving the park or adjusting their departure time appropriately. The rest of the touring plan is then displayed, so that the customer can plan accordingly. A sample touring plan with a 35 minute lunch scheduled for 12:30 PM at the Plaza Pavilion is shown in Figure 3.

Here is the Touring Plan:

The Magic Kingdom opens at 07:30 a.m.
You arrive at the Magic Kingdom at 07:30 a.m.
Your scheduled departure time is 24:00 p.m.

Your touring plan will start at the end of Main Street when the park opens.
It takes about 3 minutes to walk to the first attraction.

Time of Day	Scheduled Attraction	Wait Time (est)	Ride Time (est)	Walk to Next Attraction
07:33 a.m.	Alien Encounter	0m	12m	8m
07:53 a.m.	Peter Pan's Flight	5m	4m	11m
08:13 a.m.	Tomorrowland Transit Authority	0m	10m	4m
08:27 a.m.	Space Mountain	15m	3m	16m
09:01 a.m.	Big Thunder Mountain Railroad	0m	4m	2m
09:07 a.m.	Splash Mountain	0m	10m	5m
09:22 a.m.	Pirates of the Caribbean	5m	8m	3m
09:38 a.m.	Country Bear Jamboree	20m	15m	8m
10:21 a.m.	It's a Small World	10m	11m	2m
10:44 a.m.	Haunted Mansion	25m	9m	10m
11:28 a.m.	Donald's Boat	0m	5m	1m
11:34 a.m.	Mickey's Country House	25m	22m	13m
12:30 p.m.	The Plaza Pavilion	0m	30m	5m
13:13 p.m.	The Jungle Cruise	45m	9m	12m
14:19 p.m.	Carousel of Progress	23m	18m	11m
15:11 p.m.	Diamond Horseshoe Saloon Revue	30m	40m	1m
16:22 p.m.	Hall of Presidents	22m	23m	4m
17:11 p.m.	Swiss Family Treehouse	5m	13m	7m
17:36 p.m.	The Timekeeper	12m	20m	11m
18:19 p.m.	Legend of the Lion King	30m	16m	6m
19:11 p.m.	Barnstormer@Goofy's Wiseacres Farm	15m	10m	1m
19:37 p.m.	Minnie's Country House	15m	1m	0m

Approximate Plan completion at 19:53 p.m.
Total minutes touring : 739
Total minutes on attractions : 293
Total minutes of walking : 144
Total minutes of waiting : 302
--End--

Fig. 3. A Sample Touring Plan.

4. Test Problems

We chose the dynamic programming heuristic described in [22] as the algorithm to compare against our EA for two reasons. First, the DP heuristic has been shown to perform better than MILP models for TDTSPs of up to 55 cities, and so appears to be the best known approach to solving TDTSPs. Second, the MILP model described in [9] had many tens of thousands of temporal and capacity constraints, which appeared to be solvable on only large, commercial linear programming software. In contrast, the DP heuristic was coded and tested over the course of a few days by one of the authors.

Both the DP heuristic and EA were coded as Java applets [23], for distribution on the World Wide Web. Since Java applets run in the context

Dynamic Programming Heuristic				**Evolutionary Algorithm (100 Trials)**					
Retained Partial Tours	Elapsed Time (sec)	Best Path Cost	Best Path / Best Known	Elapsed Time (Sec)	Average Path Cost	Median Path Cost	Standard Deviation	Best Path	Average Path/Best Known
10	< 1	179	1.072	60	167	167	0.0	167	1.000
100	< 1	167	1.000						

Fig. 4. Results for 5 Ride Tour.

Dynamic Programming Heuristic				Evolutionary Algorithm (100 Trials)					
Retained Partial Tours	Elapsed Time (sec)	Best Path Cost	Best Path / Best Known	Elapsed Time (Sec)	Average Path Cost	Median Path Cost	Standard Deviation	Best Path	Average Path/Best Known
10	<1	313	1.098	60	285.89	286	0.31	285	1.003
100	<1	286	1.004	120	285.82	286	0.12	285	1.003
1,000	2	285	1.000						
10,000	51	285	1.000						
15,000	92	285	1.000						

Fig. 5. Results for 10 Ride Tour.

Dynamic Programming Heuristic				Evolutionary Algorithm (100 Trials)					
Retained Partial Tours	Elapsed Time (sec)	Best Path Cost	Best Path / Best Known	Elapsed Time (Sec)	Average Path Cost	Median Path Cost	Standard Deviation	Best Path	Average Path/Best Known
10	<1	559	1.075	60	524.44	523	1.26	520	1.009
100	<1	540	1.038	120	522.51	522	1.26	520	1.005
1,000	15	521	1.002	180	522.86	521	4.84	520	1.006
10,000	739	520	1.000	300	521.13	521	0.69	520	1.002
15,000	3,841	520	1.000						

Fig. 6. Results for 15 Ride Tour.

Dynamic Programming Heuristic				Evolutionary Algorithm (100 Trials)					
Retained Partial Tours	Elapsed Time (sec)	Best Path Cost	Best Path / Best Known	Elapsed Time (Sec)	Average Path Cost	Median Path Cost	Standard Deviation	Best Path	Average Path/Best Known
10	<1	819	1.119	60	749.68	749	1.84	736	1.024
100	1	789	1.078	120	745.18	746	1.59	734	1.018
1,000	33	769	1.051	180	743.13	744	2.50	733	1.015
1,430	60	769	1.051	300	742.31	743	2.28	732	1.014
2,100	120	756	1.033						
2,655	180	756	1.033						
10,000	2,602	754	1.030						
15,000	25,862	748	1.022						

Fig. 7. Results for 20 Ride Tour.

Dynamic Programming Heuristic				Evolutionary Algorithm (100 Trials)					
Retained Partial Tours	Elapsed Time (sec)	Best Path Cost	Best Path / Best Known	Elapsed Time (Sec)	Average Path Cost	Median Path Cost	Standard Deviation	Best Path	Average Path/Best Known
10	<1	1,873	2.386	60	835.44	837	7.39	801	1.064
100	1	857	1.092	120	824.87	821	2.63	791	1.051
1,000	51	849	1.082	180	813.44	809	3.77	786	1.036
1,100	62	849	1.082	300	803.91	801	4.73	785	1.024
1,575	120	841	1.071						
2,000	189	834	1.062						
10,000	4,275	823	1.048						
15,000	13,994	822	1.047						

Fig. 8. Results for 25 Ride Tour.

of a Web browser, measuring the CPU utilization of the applet directly is not feasible. Thus, we measured the running time of each algorithm in seconds of real time. Both algorithms were tested on a 233-megahertz Pentium computer, running Microsoft Windows NT 4.0. The browser was Microsoft Internet Explorer version 4.01. The Java compiler was Symantec Café version 1.0.

Both algorithms were tested on 5, 10, 15, 20 and 25 ride tours. The tours were programmed to start at 9:00 A.M. on an "early-entry" day in the Magic Kingdom park. The performance and running time of the DP heuristic was measured when it was allowed to retain 10, 100, 1,000, 10,000 and 15,000 partial tours at each stage, except the 5 and 10 ride tours. For the 20 and 25 city tours, additional tests were run

in which the number of partial tours retained by the DP heuristic was adjusted so that the running time of both algorithms would be equal. This allowed us to compare the performance of both algorithms with the same amount of running time. The EA was measured by allowing it to run for 60, 120, 180 and 300 seconds. The results of the tests are shown below.

In addition, the authors had previously tested TPs containing 21 rides with different lunch scheduling needs. All tests were performed on a 166 Mhz Pentium computer with 64 MB of RAM and Netscape Navigator 3. The tours were programmed to begin at 7:30 A.M. on an "early-entry" day in the Magic Kingdom park. When the EA did not have to schedule lunch, the average TP produced had a cost of 709 minutes. When the EA was asked to schedule a thirty-minute lunch around noon at any of the restaurants in the park, the average TP produced had a cost of 746 minutes. When the EA scheduled a thirty-minute lunch for exactly noon at any of the restaurants in the park the average TP was 751 minutes. The difference between the TPs with no lunch and those with flexible lunch scheduling was only 37 minutes. Since 30 of those 37 minutes were for the lunch itself, only 7 minutes of slack time were introduced by adding a flexible lunch constraint to the TP. Adding a rigid lunch constraint introduced only twelve minutes of slack time, still within an acceptable margin of the best-case scenario of zero minutes of slack time. While the EA did not produce any TP exhibiting the worst-case lunch scheduling scenario, that worst-case scenario is easy to describe. A TP would exhibit worst-case characteristics if the last ride before lunch was scheduled so that the customer did not have enough time to walk from that ride to the restaurant and still arrive at the restaurant on time. For example, a TP with a rigid lunch time of noon at the Liberty Tree Tavern could not have the customer departing the Space Mountain ride at the opposite end of the park at 11:59 A.M.; the walking time would be more than one minute, violating the rigid lunch constraint. In this case, the EA would schedule lunch between the ride scheduled before Space Mountain and Space Mountain, and build slack time into the schedule to satisfy the rigid lunch time constraint. This has the potential to greatly increase the overall cost of the TP, since the slack time could not be used to tour the rest of the park.

5. Comparison with Other Approaches

The authors implemented the dynamic programming heuristic described in [22] to compare against the tours produced by the EA. The tours produced by the dynamic programming heuristic have been shown superior to those produced by MILP models [22] for TDTSPs of up to 55 cities.

For the 5 and 10 ride tours the dynamic programming heuristic provides equal or superior results in less time than the evolutionary algorithm. This conclusion is not surprising; due to the relatively small number of tour permutations, brute force methods can be used to solve TDTSPs of this size as well, in a similar amount of time.

For the 15 ride tour, the performance of the DP heuristic and the EA is roughly equivalent, with the DP heuristic producing slightly better results. The DP heuristic produces a tour whose cost is within 1 minute of the best known tour, in about fifteen seconds. The median tour produced by the EA with 60 seconds of run time is within 3 minutes of the best known tour. However, we see with this problem that the running time of the DP heuristic increases sharply when 15K partial tours are retained. This is probably due to the relatively large number of tour permutations, combined with the large number of partial tours that have to be processed at each stage.

The EA exhibits superior performance on the 20 and 25 ride tour problems, both in terms of running time and tour quality. For the 20 ride tour problem, the average path generated by the EA with sixty seconds of running time is superior to the best path generated by the DP heuristic in forty-three minutes of running time. Also, the EA produces a better tour with 120 seconds of running time than the DP heuristic can with over seven hours of running time. For the 25 ride tour problem, the results are similar. The DP heuristic requires over an hour of processing time to produce a tour of comparable quality to the average tour produced by the EA in two minutes.

Also, the authors believe the EA described in this paper exhibits several improvements over the techniques described in [12] for planning tours of theme parks. The TDM/RSP models in [12] segment the day into optimal morning, afternoon and evening tours of rides in close proximity to each other, but ignores globally optimal tours which would require walking slightly longer distances to save time standing in line. Our EA approach considers such tours while factoring in the customer's preferences for walking versus waiting in line.

In addition, while the TDM/RSP model creates TPs of only the rides in a theme park, our approach includes practical considerations such as scheduling time to eat lunch. The algorithm to schedule lunch can be extended in a straightforward manner to encompass other user-defined breaks in the day, such as dinner.

Finally, our EA has a convenient, user-friendly front end, and is available on the World Wide Web at http://www.eng.ncat.edu/ testa/html/ mk_ plan.html. A browser that supports both Java and JavaScript, such as Microsoft Internet Explorer 4.01 or Netscape Navigator 4, is required.

6. Summary

In this paper we have described the problem of planning efficient tours of a theme park as an instance of the time-dependent traveling salesman problem. Past approaches to solving similar problems through MILP models and dynamic programming were detailed, and an evolutionary algorithm with local search heuristics was described. The EA's exclusive use of a swap mutation operator, rather than traditional crossover operators, to produce offspring was stated, as was the effectiveness of the swap mutation operator over these traditional crossover operators. Test problems and results were shown which demonstrated that for touring plans with a significant number of rides, the EA produces better quality tours in less time than the previous best known algorithm, a dynamic programming heuristic. Additionally, it was shown that the EA efficiently scheduled lunch with either rigid or flexible time constraints, although the satisfaction of those constraints was deferred until the final step of the EA. It was stated that delaying the constraint satisfaction until the last step allowed generation of more offspring, which allows a better search of the solution space. Several other benefits of this new approach over existing models were also described, including the consideration of user-defined preferences (such as walking versus waiting in line) in the overall TP, and availability on the World Wide Web.

7. Future Directions

The authors are actively exploring several directions of future research for this application. The authors noted dramatic improvement in the quality of the TPs produced when the scheduling of lunch was deferred outside of the main loop. Whether this is due to the geography of the restaurants in the Magic Kingdom is an area of future research. The question of whether other constraints, such as stopping for snacks, shopping, or restroom breaks can be deferred successfully on other kinds of TDTSPs is another area of future research.

Many large organizations, such as church, school, or scouting troops that tour the Magic Kingdom often separate into smaller groups that rendezvous periodically throughout the day. A "head group" usually keeps track of, and handles the administrative tasks for the smaller groups. The addition of rendezvous constraints will allow for the efficient coordination of the smaller group's multiple, independent tours. Currently, most rendezvous are centralized, meaning all of the small groups must gather together at a single place and time to meet with the head group. By incorporating a decentralized rendezvous approach, a larger portion of the theme park can be visited in the same amount of time. The decentralized rendezvous approach will allow the head group to meet at different times, different locations, and with different sets of groups, and can be implemented through the use of a number of rendezvous constraints.

Finally, the customer should be able to rank the importance of each ride to their theme park visit, especially if the customer does not have much time to spend in the park. Ranking the rides by importance would allow the EA to generate TPs that visit the most important rides first, ensuring the customer does not spend valuable time

on less-important rides. To accomplish this, the cost function would have to be augmented to consider criteria other than time in the fitness of the TP.

One application of this research is adapting the algorithm to run in real time on a portable, hand-held device such as the Palm Pilott. The authors envision an integrated system where theme park customers would select the rides they wished to visit from a list on a rented hand-held device. Computers installed at each ride would send estimates of that ride's current wait time to a central computer. The central computer would broadcast these wait times to receivers attached to the hand-held devices (perhaps using current digital pager technology), which would dynamically recalculate and redisplay the remainder of the touring plan based on the most recent wait times at each ride. Global positioning could be used to keep track of where the customer is located in the park so the time to walk to the next ride could be included in the calculations. Such a system would assist management at the theme park by helping distribute customers evenly throughout the park. Customers would minimize waits in line, which should improve overall satisfaction with the theme park. A nominal rental fee for each hand-held device could offset the cost of the system.

8. Acknowledgements

The authors would like to thank Bob Sehlinger for his expert advice on touring the Magic Kingdom. We are also grateful to Nick Sahinidis, David Fogel, and Chryssi Malandraki for providing copies of their recent papers. Patricia Shanahan provided some tips on the implementation of the dynamic programming heuristic. Finally, we thank the anonymous referees for their thorough comments, which have greatly improved this paper.

Appendix A: Rides and Restaurants at Walt Disney World's Magic Kingdom

The Rides

1. Swiss Family Treehouse
2. The Jungle Cruise
3. Pirates of the Caribbean
4. Cinderella's Golden Carousel
5. Dumbo the Flying Elephant
6. It's a Small World
7. Legend of the Lion King
8. Mad Tea Party
9. Mr. Toad's Wild Ride
10. Peter Pan's Flight
11. Snow White's Adventure
12. Twenty Thousand Leagues Under the Sea
13. Big Thunder Mountain Railroad
14. Country Bear Jamboree
15. Diamond Horseshoe Saloon Revue
16. Splash Mountain
17. Tom Sawyer's Island
18. Riverboat
19. Mike Fink Keelboats
20. Hall of Presidents
21. Haunted Mansion
22. Minnie's Country House
23. Mickey's Country House
24. The Barnstormer @ Goofy's Wiseacres Farm
25. Astro-Orbiters
26. Take Flight
27. Alien Encounter
28. Grand Prix Raceway
29. Space Mountain
30. Carousel of Progress
31. Tomorrowland Transit Authority
32. Skyway to Fantasyland
33. The Timekeeper
34. Donald's Boat
35. Enćanted Tiki Room
36. Skyway to Tomorrowland
37. End of Main Street

The Restaurants

38. Casey's Corner
39. The Plaza Restaurant
40. Tony's Town Square Restaurant
41. The Crystal Palace
42. El Pirate y El Perico
43. Columbia Harbor House
44. Liberty Tree Tavern
45. Diamond Horseshoe Saloon Revue
46. Pecos Bill Cafe
47. Aunt Polly's
48. The Pinočhio Village Haus
49. King Stefan's Banquet Hall
50. Cosmic Ray's Starlight Cafe

Appendix B: Sample of Distance Matrix

A sample of the distance matrix is shown below. All times are in minutes.

Origination / Destination	Swiss Family Treehouse	Jungle Cruise	Pirates of the Caribbean	Cinderella's Golden Carousel
Swiss Family Treehouse	0	2	3	9
Jungle Cruise	2	0	2	12
Pirates of the Caribbean	3	2	0	11
Cinderella's Golden Carousel	9	12	11	0

Appendix C: Sample Ride Time Array

Name of Ride	Ride time (minutes)
Swiss Family Treehouse	13
Jungle Cruise	9
Pirates of the Caribbean	8
Cinderella's Golden Carousel	2

Appendix D: Sample Wait Time Matrix

A sample of the wait time matrix is shown below. All times listed are in minutes.

Time / Ride	9:00AM	9:15AM	9:30AM	9:45AM
Swiss Family Treehouse	0	0	0	5
Jungle Cruise	0	10	15	20
Pirates of the Caribbean	0	5	5	10
Cinderella's Golden Carousel	0	10	10	10

Appendix E: The Modified Lebby Crossover (MLX) Function

The original version of this crossover function was devised by Dr. Gerald Lebby of the Department of Electrical Engineering at North Carolina A&T State University for use in EAs to solve traditional TSPs. When used on two parent strings of equal length, it avoids the problem of generating an offspring with an invalid tour [1]. For a TSP with N cities, each parent string consists of a permutation of the numbers 1..N. The original Lebby crossover function steps through each position in the first parent string and uses the value found in each position as an index into the second parent string. The value found at that index in the second parent string is transferred to the offspring. The original Lebby crossover function is shown in figure E1 below.

The following example shows how the OLX function would produce an offspring from two parent strings of five cities each.

Let *parent*1 = {4, 5, 1, 3, 2}
Let *parent*2 = {1, 2, 3, 5, 4}

Step 1

The value of the 1st element in *parent1* is 4. Find the value in the 4th position of *parent2* and insert it into the 1st position in offspring. The value in the 4th position of *parent2* is 5, so the partially constructed path in *offspring* is {5}.

Step 2

The value of the 2nd element in *parent1* is 5. The value in the 5th position of *parent2* is 4, so place 4 in the 2nd position of *offspring*: {5, 4}.

Step 3

The value in the 3rd position of *parent1* is 1, and the value in the 1st position of *parent2* is 1, so the 3rd position of *offspring* is 1: {5, 4, 1}.

Step 4

The value of the 4th element of *parent1* is 3. The value in the 3rd position of *parent2* is 3, so the value in the 4th position of *offspring* should be 3: {5, 4, 1, 3}.

Step 5

The 5th position of *parent1* is 2, and the 2nd position of *parent2* is 2, so let the 5th position of *offspring* be 2.

The algorithm stops when there are no more elements in the parents to process. The complete path of the offspring is {5, 4, 1, 3, 2}.

The original OLX function was modified to mate parents whose strings contained tours of M cities chosen from N possible cities, $M \leq N$. The MLX algorithm expects *parent2* to be a permutation of *parent1*. That is, each city in the tour of *parent1* must also be in *parent2*, and vice versa. Along with the two parent strings the MLX function takes a third parameter which must be a list of the cities in *parent1*, sorted in ascending order. The MLX algorithm is shown in figure E2.

As an example of the MLX function, let *parent*1 = {4, 5, 9, 3, 2}, *parent*2 = {9, 2, 3, 5, 4} and *ordered_list*= {2, 3, 4, 5, 9}. The output of the MLX function would be an offspring with a path {3, 5, 4, 2, 9}.

```
Function OLX( parent1, parent2) returns offspring
  Begin
    For k = 1 to (number of elements in parent1) do
      Let j = the value of the kth element in parent1
      Let m = the value of the jth element in parent2
      Set the kth element of offspring to m
    Done

    Return offspring
  End
```

Fig. E1 The Original Lebby Crossover Function

```
Function MLX(parent1, parent2, ordered_list) returns offspring
  Begin
    For k =1 to (number of elements in parent1) do
      Let j = the value of the kth element in parent1
      Let m = the position in ordered_list that contains the value of j
      Let s = the value of the mth element in parent2
      Set the kth element of offspring to s.
    Done

    Return offspring
  End
```

Fig. E2.

References

[1] D. GOLDBERG, "Genetic Algorithms in Search, Optimization, and Machine Learning" pp. 170–174.

[2] G. LEBBY, Department of Electrical Engineering, North Carolina A&T State University, 1996.

[3] C. C. HSU, J. MARTIN, S. YAMADA, H. FUJIKAWA, K. SHIDA, "An Effectiveness Analysis of Genetic Operators for TSP" Proceedings of the IEEE International Symposium on Industrial Electronics, pp. 766–769, 1995.

[4] B. FREISLEBEN AND P. MERZ, "A Genetic Local Search Algorithm for Solving Symmetric and Asymmetric Traveling Salesman Problems" Proceedings of the 1996 IEEE International Conference on Evolutionary Computation, pp. 616–621, 1996.

[5] H. TAMAKI, H. KITA, N. SHIMIZU, K. MAEKAWA AND Y. NISHIKAWA, "A Comparison Study of Genetic Codings for the Traveling Salesman Problem" IEEE International Conference on Evolutionary Computation, p. 2, 1994.

[6] B. SEHLINGER, "The Unofficial Guide to Walt Disney World '98", Macmillan Travel, 1996.

[7] S. BIRNBAUM, T. PASSAVANT, J. SAFFRO(eds.), "Birnbaum's Walt Disney World: The Official Guide", Hyperion, 1997.

[8] R. AERO, "Rita Aero's Walt Disney World for Adults: And Families, Too! (3rd ed.)", Fodor's Travel Publications, 1997.

[9] C. MALANDRAKI AND M. DASKIN, "Time Dependent Vehicle Routing Problems: Formulations, Properties and Heuristic Algorithms" Transportation Science, Vol. 26, No. 3, pp. 185–200, 1992.

[10] N. CHRISTOFIDES, A. MINGOZZI AND P TOTH, "The Vehicle Routing Problem" in Combinatorial Optimization, Chapter 11, pp. 315–338, N. Christofides, A. Mingozzi, P. Toth and C. Sandi (eds.), John Wiley & Sons, 1979.

[11] R. VANDER WIEL AND N. SAHINIDIS, "Heuristic Bounds and Test Problem Generation for the Time-Dependent Traveling Salesman Problem" Transportation Science, Vol. 29, No. 2, pp. 167–183, 1995.

[12] R. AHMADI, "Managing Capacity and Flow at Theme Parks" in Operations Research, Vol. 43, No. 1, pp. 1–13, 1997.

[13] L. BODIN, B. GOLDEN, A. ASSAD AND M. BALL, "The State of the Art in Routing and Scheduling of Vehicles and Crews," Office of Policy Research, Urban Mass Transportation Administration, U.S. Department of Transportation, Report UTMA/BMGT/MSS#81–001, Washington D.C., 1981.

[14] D. WHITLEY, T. STARKWEATHER, AND D. FUQUAY, "Scheduling Problems and Traveling Salesmen: The Genetic Edge Recombination Operator," Proceedings of the Third International Conference on Genetic Algorithms, pp. 133–140, 1989.

[15] L. ZADEH, R. R. YAGE (ed.), "Fuzzy Sets and Applications: Selected Papers by L. A. Zadeh", John Wiley and Sons, 1987.

[16] L. J. ESHELMAN AND J. D. SCHAFFER, "Crossover's Niche", Proceedings of the Fifth International Conference on Genetic Algorithms and their Applications, S. Forrest, ed., Morgan Kaufmann, 1993.

[17] J. BOWEN AND G. DOZIER, "Solving "Randomly Generated Fuzzy Constraint Networks Using Evolutionary/Systematic Hill-Climbing", Proceedings of the Fifth IEEE Conference on Fuzzy Systems (Fuzz-IEEE '96), pp. 226–231, 1996.

[18] D. E. GOLDBERG AND K. DEB, "A Comparative Analysis of Selection Schemes Used in Genetic Algorithms", Foundations of Genetic Algorithms (FOGA-1), pp. 69–93, Gregory J. E. Rawlins, Ed., Morgan Kaufman, 1991.

[19] T. BAECK, F. HOFFMEISTER, AND H. P. SCHWEFEL, "A Survey of Evolution Strategies", Proceedings of the Fourth International Conference on Genetic Algorithms, pp. 2–9, Richard K. Belew and L. Booker, Eds., Morgan Kaufmann, 1991.

[20] Z. MICHAELWICZ, "Genetic Algorithms + Data Structures = Evolution Programs", 2nd Edition, Springer–Verlag Artificial Intelligence Series, 1994.

[21] S. COOMBS AND L. DAVIS, "Genetic Algorithms and Communication Link Speed Design", Proceedings of the Second International Conference on Genetic Algorithms, pp. 252–256, 1991.

[22] C. MALANDRAKI AND R. B. DIAL, "A restricted dynamic programming heuristic algorithm for the time-dependent traveling salesman problem", European Journal of Operational Research 90, pp. 45–55, Elsevier Science B.V., 1996.

[23] H. M. DEITEL AND P. J. DEITEL, "Java How to Program", Prentice Hall, pp. 16–19, 1997.

[24] R. VANDER WIEL AND N. SAHINIDIS, "An Exact Solution Approach for the Time-Dependent Traveling Salesman Problem", Naval Research Logistics, Vol. 43, pp. 797–820, 1996.

[25] K. RULAND, Polyhedral Solution to the Pickup and Delivery Problem, PhD Thesis, Washington University (Saint Louis), 1995.

[26] D. B. FOGEL AND J. K. ATMARM, "Comparing Genetic Operators with Gaussian Mutations in Simulated Evolutionary Processes Using Linear Systems", Biological Cybernetics 63, Springer–Verlag, pp. 111–114, 1990.

Received: June, 1998
Revised: October, 1998
Accepted: January, 1999

Contact address:
Leonard Testa
North Carolina A &T State University
Greensboro
North Carolina
USA
e-mail: testa@ncat.edu

Albert C. Esterline
North Carolina A &T State University
Greensboro
North Carolina
USA
e-mail: esterlin@ncat.edu

Gerry V. Dozier
Department of Computer Science and Engineering
Auburn University
Alabama
USA
e-mail: gvdozier@eng.auburn.edu

LEN TESTA is a graduate student at North Carolina A&T State University. His research interests include the evolutionary computing and scheduling problems. Len is a member of IEEE and the ACM.

ALBERT ESTERLINE received a Ph.D. in computer science and an MS in mathematics from the University of Minnesota and a Ph.D. in philosophy from the University of St. Andrews (Scotland). He is currently an assistant professor of computer science at North Carolina A&T State University. His research interests include multi-agent systems, formal methods, and soft computing.

GERRY VERNON DOZIER received a B.S. in Computer Science and Mathematics from Northeastern Illinois University in 1988 as well as an MS and Ph.D. in Computer Science from North Carolina State University in 1991 and 1995, respectively. From the fall of 1995 to the spring of 1997, Gerry was an assistant professor in the Computer Science Department at North Carolina A&T State University. In the fall of 1997, Gerry became an assistant professor in the Computer Science & Engineering Department at Auburn University. His current research interests include the hybrid application of traditional, evolutionary, neural, and fuzzy computing in an effort to solve constrained optimization problems. Gerry is a member of IEEE and AAAI.