Heuristic Approach to the Artificial Intelligence Design Challenge

Ting-Yi Sung,* Her-Jiun Sheu,* and Denis Naddef†
New York University, New York, New York

We propose a heuristic approach to solve the AIAA design challenge problem. The procedure employs the repeated solution of knapsack problems with possibly changing coefficients to build a tour and to determine subsets of cities to be inserted or dropped. The local constraint and triangular inequality violations are taken care of in a "tour-arrangement" stage. Furthermore, general exchange routines for the traveling salesman problem are used to improve the tour. As for a given tour, we apply addition and replacement routines to satisfy the probability constraint.

I. Introduction

WE describe in this paper a heuristic method for the problem proposed as the AIAA Artificial Intelligence Design Challenge. It is designed to find good, if not optimal, solutions to large-scale problems of the kind proposed in the Ref. 1, and does in no way exploit the relatively small number of cities (11) considered in the proposed problem. For problems of small size heuristics based on trying to fix some variables, e.g., ruling out some cities and ruling others in, forbidding or imposing some compatible with those fixings will be near optimal and quite efficient. This kind of implicit enumeration method, however, loses its efficiency and becomes prohibitive as the number of cities involved in the problem grows.

Our method involves a three-step procedure:

1) Making a first choice of cities to be visited based on the solution to a knapsack problem (see, e.g., Chap. 6 in Ref. 2).

2) Arranging the selected cities in a tour using a farthest insertion algorithm and an exchange procedure.

3) Applying addition and replacement routines to satisfy the probability constraint and to improve the tour.

The main difficulties come from the facts that 1) Boston (city LC2) has to be visited at least once after Los Angeles (city LC1) in order to collect value for Los Angeles, and 2) the probability of exceeding the budget must be less than α. We will illustrate our approach in Fig. 1 and discuss the details below.

II. Choosing a First Set of Cities

The selection of a first set of cities to be visited is based on a knapsack calculation that goes as follows.

First we update the cost table in order to include the fixed cost of visiting a city:

$$C_{ij}^{+}=\begin{cases} C_{ij}+(f/2) & \text{when } i=\text{home or } j=\text{home} \\ C_{ij}+f & \text{otherwise} \end{cases} \tag{1}$$

where C_{ij} is economy airfare from city i to city j and f is the fixed cost incurred when visiting a city other than the home city. For every pair of cities i and j, we check for each city k other than i,j whether or not we have a triangular inequality violation, i.e., $C_{ij}^{+}\geq C_{ik}^{+}+C_{kj}^{+}$. If yes, we store it in the list.

For each city j we select the k cheapest routes coming into it and the k cheapest routes leaving it (in our program we choose $k=3$), and compute the average C_j of those $2k$ numbers. The reason for doing so is that we expect this average number to give us a good approximation of the cost incurred by visiting this city.

Given the expected cost C_j of visiting city j, we wish to select a subset of cities to be visited that satisfies the budget minus an adjustment for the risk of having to fly first class. To this end, we solve the following knapsack problem:

$$\max \sum_{j\in N} a_j x_j$$

$$\text{such that } \sum_{j\in N} C_j x_j \leq B^* \quad (KP^0)$$

$$x_j \in \{0,1\} \qquad \text{for all } j\in N$$

where N is the set of all the cities and a_j represents the profit or value of choosing city $j\in N$. B^* is equal to the budgeted amount minus a value B that represents an adjustment for the risk of having to fly first class. The value of the buffer cost B is calculated in the following way.

Let $\bar{c}$ be the expected value of the cost of visiting any city, i.e., the average of the numbers C_j for all $j\in N$ We estimate the expected number of cities to be visitied by $n=\text{budget}/\bar{C}$. In order to determine the right-hand side B^*, we need to take into consideration the allowed probability of exceeding the budget, the probability of having to take a first-class flight, and the first-class-flight mark-up factor. We then apply the central limit theorem to get this estimation of the risk adjustment. More precisely, let

α = allowed probability of exceeding the budget
δ = first-class premium
p = probability of having to pay the first-class premium
$\bar{C}'$ = average air fare, without fixed cost, to visit a city
μ = expected value of excess fare for first class to visit a city
σ^2 = variance of excess fare for first class to visit a city
B = buffer cost for excess fare of the tour visiting n cities
$Z_{1-\alpha}$ = fractile of the standardized normal distribution whose cumulative probability is $1-\alpha$, i.e., $Pr\{Z\leq Z_{1-\alpha}\}=1-\alpha$

Recall that $\bar{C}$ is the average cost, including fixed cost, to visit a city and n is the expected number of cities that we are going to visit, expressed as budget/$\bar{C}$. Here, we implicitly assume that n is a good approximation for number of cities to

Received June 23, 1987; presented as Paper 87-2336 at the AIAA 1987 Guidance, Navigation, and Control Conference, Monterey, CA, Aug. 17-19, 1987; revision received Feb. 1, 1988.

*Department of Statistics and Operations Research.
†Department of Statistics and Operations Research; permanently with Institut Imag, Saint Martin d'Heres Cedex, France.

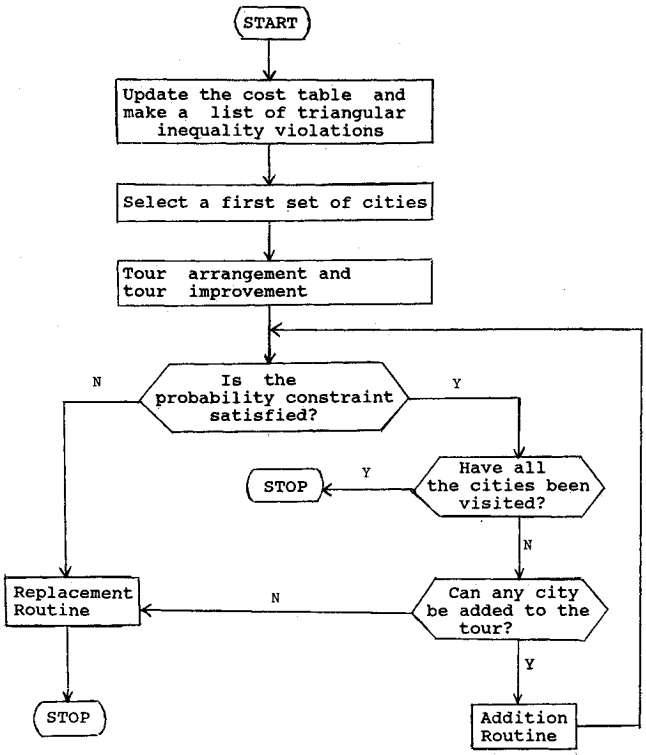

Fig. 1 Flowchart of heuristics.

be visited. Furthermore, we assume that the excess costs for the first-class fare on each flight-leg are identically independently distributed random variables. Then we have the following equalities:

$$\mu = (1-p)*0 + p(\delta\bar{C}') = p\delta\bar{C}' \tag{2a}$$

$$\sigma^2 = (1-p)(p\delta\bar{C}')^2 + p(\delta\bar{C}-\delta\bar{C}')^2 = (\delta\bar{C}')^2 p(1-p) \tag{2b}$$

$$B = n\mu + Z_{1-\alpha}\sigma\sqrt{n} \tag{2c}$$

Using the above result B^*, by our definition $B^* = \text{budget} - B$, can be rewitten as

$$B^* = \text{budget} - n\mu - Z_{1-\alpha}\sigma\sqrt{n} \tag{3}$$

We solve the 0-1 knapsack problem using the branch-and-bound algorithm developed by Martello and Toth,[3] which has been proven to be efficient. The optimal solution x^0 to the knapsack problem (KP^0) provides a good set of cities S to be visited, where $S = \{j \in N : x_j^0 = 1\}$.

III. Building a Tour from the Seleted Cities

The next phase of our approach consists of building a tour from the set of cities S selected previously. Due to the local condition, we distinguish three different cases for the starting tour.

Case 1

Cities LC1 and LC2 are among the chosen cities. We start with the tour of home to city LC1 to city LC2 and back to home so that we can ensure to collect the value of city LC1.

Case 2

City LC1 is not among the chosen cities. Let j be among the chosen cities the one that is the farthest from the home city, i.e., the one with highest two-way fare from home among all the chosen cities. We start with the tour from home to j and back.

Case 3

City LC1 is among the chosen cities but city LC2 is not. We reduce this case to either case 1 or case 2 by solving two 0-1 knapsack problems, one with the two variables associated with LC1 and LC2 set both to be equal to 1, another one with the variable associated with LC1 to be equal to 0. We compare the resulting two objective function values and choose the one with higher value; then we can return to either case 1 or case 2.

Many enumerative heuristic algorithms are well developed in the literature for tour arrangement and tour improvement.[4-7] The construction of the initial tour consists of repeatedly applying the following two procedures until all of the chosen cities are arranged in the tour.

1) Insert a new city into the "current tour" by a farthest insertion criterion. For each chosen city j not yet in the tour, we compute in turn the insertion cost d_j of inserting it in the cheapest way between any two consecutive cities in the tour, i.e.,

$$d_j = \min_{1 \le k \le m} \{C^+_{t(k),j} + C^+_{j,t(k+1)} - C^+_{t(k),t(k+1)}\}$$

where $t(k)$ is the city on the kth position in the tour and m is the number of cities in the current tour. We then choose the one with *highest* insertion cost to insert in the tour.

2) Check for triangular inequality violation. In the initialization of the program, the data are processed to include fixed cost, and a list of violations to the triangular inequality is made. Each time a new route is selected, we check whether or not it is on the list. If yes, replace the route by the shortest one to reduce the cost or increase the total collected value by including the intermediate city that is not yet in the tour. If the intermediate city is already in the tour, we need to drop the redundant ones, i.e., the previous ones whose inclusion were not due to a triangular inequality violation, or were due to a triangular inequality violation that no longer exists.

Once all chosen cities are arranged in a tour, we use two tour improvement procedures, called 1-Opt and Restricted 2-Opt, repeatedly to reduce the total cost until no improvement is found.

1-Opt

For each city in the tour, connect its preceding and succeeding cities by an arc and try to insert it between two others. If this gives a better tour, i.e., if the total cost is reduced, replace the former tour with the new one and repeat by applying the same procedure to the new tour. (See Fig. 2)

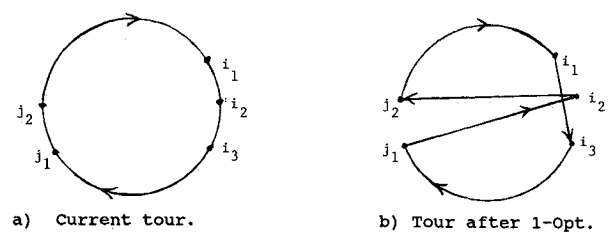

Fig. 2 Example of 1-Opt.

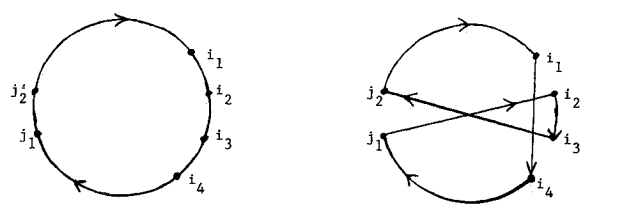

Fig. 3 Example of Restricted 2-Opt.

Restricted 2-Opt

For every pair of two consecutive cities (the reason we call it Restricted 2-Opt), connect the predecessor of the first city to the successor of the second city and try to insert these two cities between two others. (See Fig. 3)

In both procedures some restrictions apply because of the local constraint on city LC1 and city LC2. In order to collect the value of city LC1, we must visit city LC2 at least once after we visit city LC1. In this case, we must beware of not inverting the sequence of cities LC1 and LC2. The following cases apply and are taken care of automatically.

1) For the tour with city LC1 to be visited only once: When we try to insert the cities including city LC1 to other positions, we cannot insert them after the last position of city LC2. Furthermore, we cannot insert the cities including the home city to the position between city LC1 and its succeeding city LC2.

2) For the tour with city LC2 to be visited only once: When we try to insert the cities including city LC2, we cannot insert them before the first position of city LC1. Furthermore, we cannot insert the cities including the home city to the position between city LC2 and its preceding city LC1.

3) For the tour with multiple visits to both city LC1 and city LC2: If there is a city LC1 visited between the first visit and the last visit to city LC2, then:

a) If there is a city LC2 to be visited between the first visit and the last visit to city LC1, no matter how we insert the home city we never invert the sequence of city LC1 and city LC2. (See Fig. 4a)

b) Otherwise, we cannot insert the home city to the position between the last visit to city LC1 and its succeeding city LC2. (See Fig. 4b.)

On the other hand, if there is no city LC1 to be visited between the first visit and the last visit to city LC2, then

1) If all the city LC1 visits occur before the first visit to city LC2, we cannot insert the home city in the position between the last visit to city LC1 and the first visit to city LC2. (See Fig. 5a.)

2) Otherwise, we cannot insert the home city in the position between the first visit to city LC2 and its preceding city LC1. (See Fig. 5b.)

Because the set of cities in the tour is fixed, we cannot increase the total collected value by performing the 1-Opt and Restricted 2-Opt routines. It can only result in a tour with smaller total cost. Every time a better tour is obtained, we replace the current tour with the new tour and repeat the procedure until no improvement can be made. Although we have not computed an exact worst-case bound for the tour improvement procedures, the computational effort for each of the procedures is certainly polynomially bounded by $0(n^3)$.

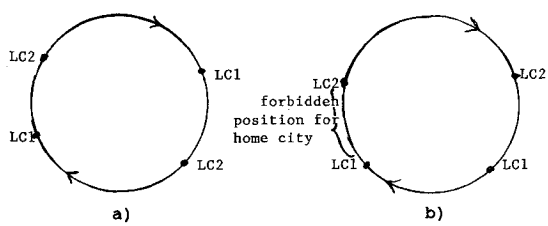

Fig. 4 Some restrictions on Restricted 2-Opt.

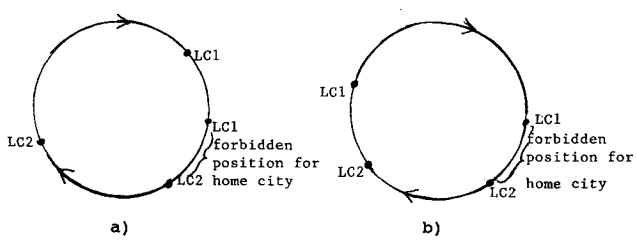

Fig. 5 Some restrictions on Restricted 2-Opt.

IV. Addition and Replacement Routines for the Current Tour

If the problem is of small size, the material of this section may seem less important, since a complete enumeration of the possible improvements might be feasible. As pointed out earlier, our aim is to devlop a heuristic approach that can be expected to work for problems of large size and that does not rely on a "brute force" enumeration of all possiblities.

When the tour improvement procedures 1-Opt and Restricted 2-Opt fail to improve the current tour, we check the probability constraint. For this purpose we use a program to calculate the exact probability of exceeding the budget for the current tour. If the current tour satisfies the probability constraint, we call an addition routine that attempts to enlarge the set S of cities in the current tour; otherwise, we call a replacement routine that will either reduce the set S or exchange cities in S with cities in W, which is defined as $N-S$.

The Addition Routine

If $S=N$, i.e., W is empty, the algorithm terminates; otherwise, we examine the possibilites of adding one or more cities to the current tour systematically. The difficulty is that it may be preferable to add, for example, two cities each with a value of 3 rather than one having a value of 5. Of course, the more cities the underlying problem has, the more difficult this selection problem becomes. To resolve this difficulty we again make use of a (different) 0-1 knapsack problem.

W is defined as before $N-S$, i.e., the set of cities not in the current tour. For every city j in W, we compute its cheapest insertion cost d_j (the same idea as in Sec. III). We then solve a knapsack problem for the cities in W with the same a_j in the objective function coefficients and with coefficients d_j in the knapsack constraint, i.e.,

$$\max \sum_{j\in W} a_j x_j$$

$$\text{such that } \sum_{j\in W} d_j x_j \le B^* \qquad (KP_W)$$

$$x_j \in \{0,1\} \text{ for all } j \in W$$

The right-hand side value B^* for the knapsack problem (KP_W) is computed by replacing the total budget with the remaining budget and adjusting it for the possible addition of some more cities to the current tour. The remaining budget is defined as

$$\text{remaining budget} = \text{budget} - C(T) - B$$

where T is the current tour, $C(T)$ the total cost of the tour T, and B the buffer cost for the excess fare of the current tour. Since we now have an actual tour, we discard our previous estimate for the buffer cost and recalculate the value for B applicable to the given tour by the following bisection method.

Step 1: Set $L=0$, $U=\delta[C(T)-\text{total fixed cost}]$.

Step 2: Set $M=(L+U)/2$.

Step 3: Check the probability constraint. If it is satisfied at a level between $\alpha-\epsilon$ and $\alpha+\epsilon$, stop. (ϵ is a chosen tolerance level.) If it is satisfied at a level greater than $\alpha+\epsilon$, set $L=M$ and return to step 2.

Otherwise, set $U=M$ and return to step 2.

In several finite steps we obtain a value B of the buffer cost for the current tour and can thus calculate the remaining budget. We can now use the method described earlier to get a buffer cost B' for adding some cities to the current tour. Denote

$\bar{d}$ = average insertion cost including fixed cost

$\bar{d}'$ = average insertion cost excluding fixed cost

n = expected number of cities that we can add to the current tour
μ = expected value of the excess cost for adding a city
σ^2 = variance of excess cost to add a city
B' = buffer cost for excess fare of adding n cities to the tour

Then, n is computed as remaining budget/$\bar{d}$, and

$$\mu = p\delta\bar{d}' \tag{4a}$$

$$\sigma^2 = (\delta\bar{d})^2\, p(1-p) \tag{4b}$$

$$B' = n\mu + Z_{1-\alpha}\sigma\sqrt{n} \tag{4c}$$

Consequently, the (new) right-hand side value B^* for the knapsack problem to be solved is given by

$$B^* = \text{remaining budget} - B'$$

$$= \text{budget} - C(T) - B - n\mu - Z_{1-\alpha}\sigma\sqrt{n} \tag{5}$$

Let x^1 be the optimal solution to the knapsack problem (KP_W) and $A \subseteq W$ be a subset of W such that $A - (j \in W : x_j^1 = 1)$. The cities in set A are cities to be included in the current tour. If A is empty, we call the replacement routine, which will be described in next section. Otherwise, a new tour T' will be formed from the set of cities in $S \cup A$. Now we are back to the same point as in Sec. III; that is, we insert each city j in A in its cheapest way, i.e., in the position where d_j, as defined in Sec. III, is attained. We repeat until A is empty. Then the routines 1-Opt and Restricted 2-Opt are called to improve the tour. Once we complete the construction of the new tour T', we redefine T as T', S as the set of cities in the tour T, and W as $N-S$. If the new tour satisfies the probability constraint, we repeat the addition routine using the updated T, S, and W until no cities can be added or W is empty; otherwise, the replacement routine is called.

The Replacement Routine

This routine has two functions. If the current tour does not satisfy the probability constraint, it examines the possibilities of dropping a city in the tour systematically; if one cannot add any city to the current tour, it examines the possibilities of replacing some city in the tour with some city not in the tour. We denote, as before, m the number of cities in the current tour, $t(k)$ the city in the kth position of the tour, $1 \le k \le m$. To express the cyclic relationship of the tour, we define $t(0) = t(m)$ and $t(m+1) = t(1)$ = home city. As before, T is the current tour, S is the set of cities in the tour T, and W is the set of cities not in the tour T. We have the total cost and the total collected value associated with the tour T, denoted as $C(T)$ and $V(T)$, respectively. The replacement routine works as follows (see also Fig. 6):

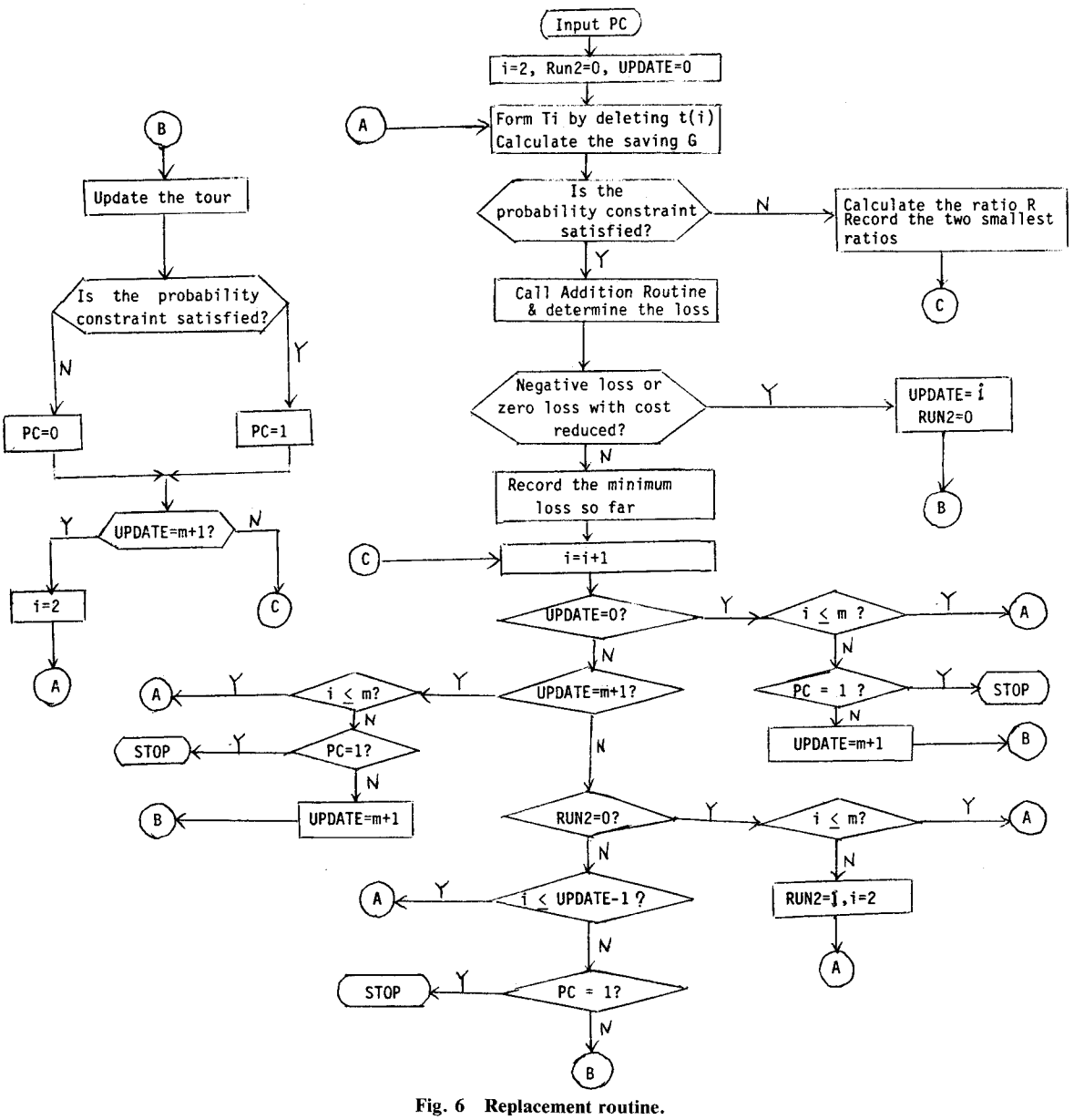

Fig. 6 Replacement routine.

Step 1: If the probability constraint is already satisfied, set $PC=1$. Otherwise, set $PC=0$. Set $i=2$, RUN2=0 and UPDATE=0.

Step 2: Form a tour $T_i=T-t(i)$ or $T-$(LC1, LC2), depending on $t(i)$.

If $t(i)\neq LC2$, drop $t(i)$ by adding the arc $[t(i-1), t(i+1)]$ and deleting the arcs $[t(i-1), t(i)]$ and $[t(i), t(i+1)]$. The saving G is then calculated as $C^+_{t(i-1),t(i)}+C^+_{t(i),t(i+1)}-C^+_{t(i-1),t(i+1)}$. (It is the same as the insertion cost.) Otherwise, drop LC2 as well as LC1 if we can collect the value of LC1. The saving is calculated similarly as the change of the total cost, i.e.,

$$G=C(T)-C(T_i)$$

Step 3: Check the probability constraint for the resulting tour T_i.

If it is satisfied, call the addition routine and find a subset of cities to be added in lieu of the dropped city. The difference between the change of the total collected value resulting from deletion of city $t(i)$ and the objective function value of the knapsack problem defines the loss, denoted as L, that results from the deletion of the city, i.e.,

$$L=\Delta V(T_i)-Z^*$$

where $\Delta V(T_i)$ is equal to $V(\text{LC1})+V(\text{LC2})$ if $T_i=T-$(LC1, LC2) and $V[t(i)]$; otherwise, $V(j)$ is the value of city j and Z^* is the objective value of the knapsack problem.

If the loss is negative, i.e., if the total collected value is increased, or the loss is zero but with total cost reduced, set RUN2=0, UPDATE=i and go to step 6. Otherwise, go to step 4. If it is not satisfied, calculate the ratio R, defined as change of total collected value $\Delta V(T_i)$ over the saving G, i.e.,

$$R=\Delta V(T_i)/G$$

and go to step 4.

Step 4: (Stopping rule)

Set $i=i+1$.

If UPDATE=0 and $i\leq m$, go to step 2.

If UPDATE=0, and $i=m+1$, then stop if $PC=1$. Otherwise, set UPDATE=$m+1$ and go to step 5.

If UPDATE=$m+1$, then:

a) If $i\leq m$, go to step 2.

b) If $i=m+1$, and $PC=1$, stop, otherwise, set UPDATE=$m+1$ and go to step 5.

If $2\leq$UPDATE$\leq m$, then

a) If RUN2=0 and $i\leq m$, go to step 2.

b) If RUN2=0 and $i=m+1$, set RUN2=1, $i=2$, and go to step 2.

c) If RUN2=1 and $i\leq$UPDATE-1, go to step 2.

d) If RUN2=1, $i=$UPDATE, and $PC=1$, stop. Otherwise, set UPDATE=$m+1$ and go to step 5.

Step 5: (Deletion step)

If the tour is such that the probability constraint can be satisfied by dropping a single city from the current tour, we drop the one having the minimum loss; otherwise, two or more cities must be dropped from the current tour. We choose to drop two cities having the smallest ratio R.

Step 6: (Tour updating step—construct a new tour T')

Update the tour by deleting the respective city in the tour and, possibly, adding the cities determined by the addition routine. Then apply the tour improvement procedure. Redefine T as T', S as the set of cities in T, and $W=N-S$.

Step 7: Check the probability constraint for the tour T produced in step 6. If it is satisfied, set $PC=1$. Otherwise, set $PC=0$. If UPDATE=$m+1$, set $i=2$ and go to step 2. Otherwise, go to step 4.

We believe that the solution of the knapsack problems, in choosing a first set of cities to visit and in the addition routine to find a set of cities to add, provides a good guess for the cities to be included in a good tour; then it is most likely that we need to drop only one city in step 5 of the replacement routine. We check each city except the home city in turn, whether or not its deletion will result in a tour that satisfies the probability constraint. As described in step 3, if it does not result in a tour that satisfies the probability constraint, we leave this city in the tour and calculate the ratio R. Otherwise, we delete this city, but call the addition routine, since we want to determine whether or not we can add some cities into the resulting tour T_i. It may happen that we prefer to delete a city having a high value and a large saving as well. For example, we may prefer to delete a city having a value of, say, 10 rather than one having a value of, say, 3. It could happen in the former case that, due to a large saving, one could then add one or several cities with value summing up to 8, yielding a loss of L of 2 rather than 3, if in the latter case we are unable to add any city to the tour. So we do not necessarily wish to delete a city with the smallest value. In order to determine which city is going to be dropped, we should take into consideration not only the value of the city $V[t(i)]$, or $V(\text{LC1})+V(\text{LC2})$ in some cases, [i.e., $\Delta V(T_i)$], but also the resulting saving G. To this end, we then apply a knapsack problem defined on the subset W as in the addition routine, as stated in step 3. The loss of deleting a city is the difference between the value of this given city and the values of the cities we can add afterward. We choose to delete the city with minimum loss.

We still realize that it is possible for some cases to drop more than one city in the tour. This is the case that deletion of any single city in the current tour cannot result in a tour satisfying the probability constraint. The small ratio R, defined as $\Delta V(T_i)/G$, can be attributed to either small value of the city or a large saving. So, we choose to drop two cities with the smallest ratios.

If deletion of one or two cities cannot result in a tour satisfying the probability constraint, we need to delete some further city in the tour. As stated in step 4, when $PC=0$ we always go to step 5, the deletion step, and try to satisfy the probability constraint.

If the updated tour satisfies the probability constraint, we still need to perform the routine one more run to insure that no improvement can be made before we stop. More precisely, we use the variable UPDATE to indicate the position where the algorithm terminates. The variable UPDATE also indicates whether or not there is a negative loss or zero loss with $C(T)$ reduced resulting from the deletion of a city. If yes, UPDATE indicates the position of the deleted city and $2\leq$UP-

Table 1 Three examples of program implementation

Tour 1: DTT—>CHI—>LAX—>PHX—>DFW—>DEN—>MSY—>ATL—>BOS—>DTT
Tour 2: ATL—>DEN—>PHX—>LAX—>MSP—>CHI—>DTT—>BOS—>ATL
Tour 3: CHI—>LAX—>PHX—>DFW—>DEN—>MSY—>ATL—>BOS—>DTT—>CHI

Tour	Value	Minimum deterministic cost	Prob>$3000	CPU on IBM PC-AT, s
1	88	2500	0.004791	85 s
2	82	2500	0.0166	71 s
3	88	2500	0.00517	80 s

DATE $\leq m$. Otherwise, UPDATE is equal to 0 or $m+1$. UPDATE = 0 means that we do not update the tour. It is the case that starting tour of replacement routine already satisfied the probability constraint and no improvement can be made in the replacement routine. "UPDATE = $m+1$" means that we have updated the tour with $V(T)$ being reduced. If there is a negative loss in the previous run, i.e., $2 \leq$ UPDATE $\leq m$, the algorithm terminates after we check one more run up to the position right before UPDATE, i.e., UPDATE -1. Otherwise, the algorithm always terminates after we check one more run up to the last city in the current tour. In summary, our algorithm stops when the (updated) tour satisfies the probability constraint and no increase in the total collected value $V(T)$ and no decrease in the total cost $C(T)$ can happen.

V. Implementation

We ran the program on the IBM Personal Computer AT using the data provided by Dr. Owen Deutsch,[1] with $\alpha = 5\%$, $p = 30\%$, and $\delta = 40\%$. Table 1 illustrates three sample tours each with a different home city.

Acknowledgments

We wish to thank Dr. Manfred W. Padberg for his encouragement and advice and for reviewing several drafts of this paper. We are indebted to our friend Giovanni Rinaldi for his suggestions and assistance in the programming. We also wish to thank Dr. S. Martello and Dr. P. Toth for their generosity in allowing us to use their 0-1 knapsack program.

Reference

[1]"Artificial Intelligence Design Challenge," *AIAA Bulletin,* Vol. 24, Sept. 1986.

[2]Garfinkel, R.S. and Nemhauser, G.L., *Integer Programming,* Wiley, New York, 1972.

[3]Martello, S. and Toth, P., "An Upper Bound for the Zero-One Knapsack Problem and a Branch and Bound Algorithm," *European Journal of Operations Research,* Vol. 1, 1977, pp. 169–175.

[4]Lin, S., "Computer Solutions of the Travelling Salesman Problem," *Bell System Technical Journal,* Vol. 44, 1965, pp. 2245–2269.

[5]Lin, S. and Kernighan, B.W., "An Effective Heuristic Algorithm for the Travelling Salesman Problem," *Operations Research,* Vol. 21, 1973, pp. 498–516.

[6]Golden, B.L. and Stewart, W.R., "Empirical Analysis of Heuristics," *The Travelling Salesman Problem,* edited by E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys, Wiley, New York, 1985, pp. 207–250.

[7]Johnson, D.S. and Papadimitriou, C.H., "Performance Gurantees for Heuristics," *The Travelling Salesman Problem,* edited by E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys, Wiley, New York, 1985, pp. 37–86.