Birth of a Salesman

John J. Shaw,* Ronald M. James,† and Daniel B. Grunberg‡
ALPHATECH, Inc., Burlington, Massachusetts

The AIAA Artificial Intelligence Design Challenge is eccentric and intriguing, combining features from traveling salesman and stochastic optimization problems together with several side constraints thrown in for additional appeal. The challenge is to devise a solution algorithm that will reliably find optimal (or near optimal) solutions and can be extended to more general problems. We present our algorithm for this challenge in this paper. The fundamental algorithm for finding near-optimal solutions is disarmingly simple and can be made quite efficient through the introduction of several bounds on the problem. The reader will quickly discover that these bounds do not require a deep understanding of traveling salesman algorithms and should enjoy devising additional bounds to add to the potpourri.

I. Introduction

AIAA Traveling Salesman Problem

THE AIAA Artificial Intelligence Design Challenge resembles the classic traveling salesman problem (TSP), albeit with several novel twists added. A salesman wishes to visit several cities. Each city has a number of points, and the salesman accumulates each city's points the first time he visits that city. In some cases, however, the salesman cannot actually accrue the points available from a city unless another city is visited later in the tour, a precedence constraint. At each leg in his tour, the salesman will either pay a base fare (with a certain probability) or will pay a premium fare (with the complementary probability); the premium fare is always a constant multiple of the base fare. Moreover, the cost of the direct leg from one city to the next may be higher than a path that passes through intermediate cities. In addition, the salesman must pay a fixed visiting cost every time he visits a city, with the exception of the "home" city. The salesman starts off with a budget and must begin his tour and complete his tour at the home city. The salesman's objective is to collect as many points as possible subject to the constraint that he stays within his budget with a certain probability. The salesman, however, need not visit every city and can visit any city more than once (although points are accrued only on the first visit to the city). In the event that two or more tours lead to the same accumulation of points, the salesman will select the tour having the lowest expected total cost.

Objectives of this Paper

A difficult problem? Yes. Without question, the effort of having to solve a classic traveling salesman problem is only offset by the frustration of having to solve a stochastic problem as well. Oddly enough, there is a very simple (but computationally prohibitive) strategy that will find the optimal solution. In truth, this strategy may not be practical to implement, but does lead to various heuristics that are remarkably efficient. Indeed, we are often able to establish bounds on the optimal solution that suggest our algorithm is arriving at or very near to the optimal solution.

One purpose of this paper is to illustrate how this apparently difficult problem yields to a very simple strategy and how this strategy, in turn, suggests some efficient heuristic algorithms. In particular, we wish to show how a slight rearrangement transforms the TSP from a stochastic traveling salesman problem, with points accruing to cities, into the classic *deterministic* traveling salesman problem.

A second purpose of this paper is to convey to the reader several methods for bounding the problem that lead to very efficient solution algorithms. Indeed, these bounding methods actually make the algorithm and lead to twofold or better reductions in execution time as Fig. 1 attests. This figure compares the execution time vs the tour points accumulated for the standard problem set[1] for four algorithms. The two greedy algorithms are based on some very simple heuristics for building a feasible tour in a single pass. The other two algorithms are based on the fundamental search strategy that we have alluded to, with SUBTOUR I denoting the "plain vanilla" algorithm that uses no enhancements to improve execution speed, and WILLY denoting the most advanced algorithm we have developed to date. From the point of view of execution time, it is particularly striking that WILLY is much more akin to its two greedy cousins than its unadorned brother.

Synopsis of This Paper

In Sec. II we discuss possible solution strategies and introduce a fundamental strategy that will yield an optimal solution, but is itself impractical to implement. In Sec. III, we examine a heuristic based on this strategy that will find near-optimal solutions to the problem and introduce enhancements that make this heuristic computationally efficient. We present computational results in Sec. IV and illustrate the computational advantage offered by these enhancements, and close this paper in Sec. V with a wrap-up of the algorithm.

II. Solution Strategies for the TSP

Greedy Algorithms

Since our friend need not visit every city, one reasonable solution strategy is to construct a tour by simply selecting cities according to some criteria and adding them to the end of a partial tour until no more cities can be added, the termination occurring when our friend has either exhausted all of his cities or cannot extend his partial tour and remain feasible.

We consider two different methods for selecting the next city add to a tour under construction: 1) maximum points to cost and 2) minimum cost. Under the first strategy, we select the city that has the largest ratio of points to cost, cost here being the cost to add the city to the partial tour. Without any

Received Aug. 4, 1987; presented as Paper 87-2335 at the AIAA 1987 Guidance, Navigation, and Control Conference, Monterey, CA, Aug. 17-19, 1987; revision received Oct. 28, 1987.

*Group Leader, Large Scale Systems Group.
†Member of Technical Staff, Large Scale Systems Group.
‡Member of the Technical Staff, Marketing, Research, and Development.

further enhancements, the antecedent city in a precedence constraint pair is never selected under this method since we have no guarantee that we can add its partner later in the tour. Under the minimum cost strategy, we simply select the city not already in the partial tour that can be added to the tour at minimum cost. Here, we do allow the antecedent city in a precedence constraint to be selected on this basis.

We have implemented algorithms using both selection methods and will compare them with our optimization algorithm when we present computational results in Sec. IV.

Optimal Strategy

In spite of the complexity of the TSP, our traveling salesman friend can adopt a strategy that will find a feasible tour that will accumulate the maximum possible number of points; the strategy is simple in concept, but may be computationally impractical to pursue.

Before we begin, we need to cover three points. First, we will always allow our friend to follow a general tour where each city in the tour can be visited more than once.

Second, our friend can always construct a general tour from a Hamiltonian tour, where each city in the tour is visited once and only once, by replacing the original cost matrix C by a new cost matrix C' where

$$c'_{i,j} = \text{the minimum cost from city } i \text{ to city } j$$

The Hamiltonian tour is constructed using C' and then expanded (by filling in the intermediate cities implied by $c'_{i,j}$) to construct the general tour. C' can be constructed from C in $O(N^3)$ steps.[2]

Third, we assume that our friend has a decision algorithm FEASIBLE_DECISIONS(S) that returns the value TRUE if there exists an acceptable feasible tour that passes through every city in the subset of cities S at least once, and the value FALSE otherwise. We hasten to point out that FEASIBLE_DECISION simply returns a TRUE/FALSE decision and does not return with the tour that leads to that decision. (This algorithm might be able to arrive at its decision without having to construct a tour per se.)

We need to clarify what we mean by an "acceptable" tour for a subset of cities. If the subset contains two cities bound by a precedence constraint, then the antecedent city must appear at least once before its partner in the general tour. Alternatively, if the subset contains only one of the cities in a precedence constraint, then the complementary city is still permitted in the general tour (e.g., it is an intermediate city in the minimum cost path between two other cities) and can appear anywhere in the general tour. We hasten to add, however, that we always honor the precedence relation when we accrue city points for any general tour.

The basic solution strategy is now quite simple to describe. Since our friend can accumulate a city's points only once, he begins by enumerating every subset of cities that can be created from the original set of cities he can visit. A problem set having N "free" cities (i.e., all cities excluding the "home" city) will have 2^N such subsets since a city is either included or excluded from each subset. He then tallies up the points associated with the cities in each subset and sorts the subsets in descending order of their total point count. For reasons that will be clear when we discuss problem bounding methods, we also assume our friend assigns each city subset a unique label,

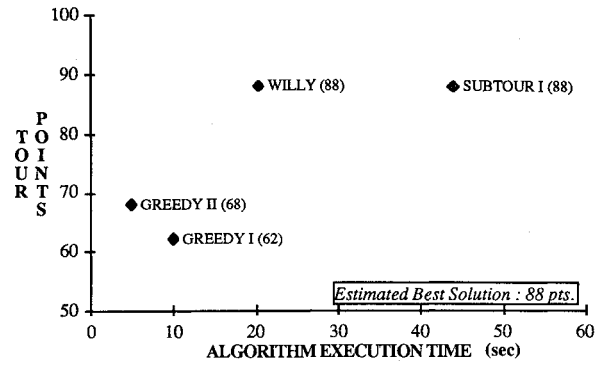


Fig. 1 Algorithm performance for the standard TSP problem set (budget = $3000).

with each city represented by a unique bit position in each label with the highest valued (lowest valued) city in the problem represented by the msot significant (least significant) bit. Subset j has the label

$$\text{Label}_j = \sum_{i=1}^{N} 2^{(i-1)} \delta_i \tag{1}$$

where $\delta_i = 1$ if the ith city is included in this subset, and 0 otherwise. For the original problem given in Ref. 1 the cities (except for the home city), their point values, and their bit-position values are shown in Table 1. The sorted subsets for the original problem appear in Table 2.

Our friend then examines each subset in order, evaluating FEASIBLE_DECISION(S) for every subset until he finds a first feasible solution, at which point he stops. At this point, our friend will actually know only the maximum possible number of points he can accumulate over all feasible tours; he also knows the first city subset to indicate this answer. He must still construct a feasible tour that passes through every city at least once in that subset; this is a matter we will ignore for now.

Although the underlying search strategy is disarmingly simple, the decision algorithm FEASIBLE_DECISION is not: It can return a value of TRUE as soon as it finds a first feasible tour that passes through all cities in the current subset, but cannot return a value of FALSE until it either has exhausted all possible permutations of cities in the subset, or has established a lower-bound test on feasibility that cannot be met by partially completed permutations. Indeed, it is as difficult as the standard traveling salesman problem in the worst case. Moreover, having found a first feasible solution using this decision algorithm, our friend still needs to find a minimum-cost feasible tour to find the optimal solution to the TSP. At first glance, it does not appear that we have made life any easier for our firend; we have not yet, but we are getting there.

III. Heuristic Approximations to the Optimal Strategy

Approximating FEASIBLE_DECISION

Our friend can simplify his task by replacing the decision algorithm FEASIBLE_DECISION with an approximate solution algorithm that first finds a minimum cost tour that passes

Table 1 Point and bit-position values for cities

	City									
	SEA	PHX	MSY	MSP	DEN	ATL	DEN	BOS	CHI	LAX
Value	6	6	6	6	6	6	8	10	116	20
Bit value	1	2	4	8	16	32	64	128	256	512

Table 2 Sorted subsets

Subset no.	Cities in the subset	Point value of the subset	Subset label
1	All	100	1023
2	All but SEA	94	1022
3	All but PHX	94	1021
4	All but MSY	94	1019
.	.	.	.
1023	Home + ATL	22	32
1024	Home	16	0

through every city in the current subset at least once and then determines whether that tour is feasible; we call this solution algorithm FIND_FEASIBLE_MINIMUM_TOUR. A little reflection suggests that this approach involves a more conservative test for feasibility than using FEASIBLE_DECISION, since it returns a value of FALSE if the minimum-cost tour passing through the cities in question is not feasible, whereas FEASIBLE_DECISION cannot return a value of FALSE until all tour permutations that can be constructed from the city subset in question have been checked for feasibility.

Although FIND_FEASIBLE_MINUMUM_TOUR involves a weaker test than FEASIBLE_DECISION, we argue it is a "reasonable" approximation on two grounds. First, our friend does ultimately want to find a minimum-cost feasible tour that accumulates the largest possible number of points, and this approximation certainly works toward that goal. Second, he can reasonably expect that the tours having the greatest likelihood of being feasible are also those tours having the minimum expected cost. This relationship is not perfect, however, since minimizing the expected cost of a tour passing through cities does not guarantee feasibility.

So far, we have given our friend a weaker test to pursue in his overall strategy, but we have not made his life easier since he must still solve a full traveling salesman problem. FIND_FEASIBLE_MINIMUM_TOUR, however, can itself be approximated by replacing the exact traveling salesman algorithm with heuristic algorithms that will find near-minimum cost tours. The advantage here is that we can replace an algorithm having a worst-case complexity of $O(N!)$ for a subset having N cities with an algorithm having substantially better worst-case behavior. The farthest insert algorithm[3] for instance has worst-case complexity of $O(N^3)$. Of course, this approach has the disadvantage that we have further weakened our test for feasibility. Along these lines, one compromise strategy is to use a quick heuristic traveling salesman algorithm to filter out city subsets that are not likely to have feasible tours passing through them and use either an exact algorithms[5] or a locally optimizing algorithm (such as the three-exchange algorithm[3-4]) to further examine those subsets that do appear promising. This is the actual strategy we have selected for our algorithm.

Basic Algorithm

Let us briefly review the heuristic solution strategy that is emerging for our friend. Our friend enumerates all possible city subsets, tallies up the points for each subset (ENUMERATE_CITY_SUBSETS), and sorts these sets into a list (SORT_SUBSETS). He then examines each subset in turn, executes the solution algorithm FIND_FEASIBLE_MINIMUM_TOUR, and stops at the first subset that produces a feasible solution. He records the points accumulated, and then continues his search down the list to try to find a solution that might have a lower expected cost, and stops when he comes across a subset whose point tally is lower than the points accumulated on his best feasible solution.

Overall, the algorithm appears as follows:

```
begin
  ENUMERATE_CITY_SUBSETS;
  SORT_SUBSETS;
  SubSet_Pointer: = 1;
  Feasible_Solution_Found: = false;
  {*Find a first feasible solution*}

repeat
  if [FIND_FEASIBLE_MINIMUM_TOUR (Subset_Pointer)]

begin
  record points accrued, expected cost, and tour;
  Feasible_Solution_Found: = true;

end;
  Increment Subset_Pointer;
until (Feasible_Solution_Found);
{*Now, try to find a cheaper solution that accumulates the same
point tally*} record the current solution as the best observed so far;


repeat
  if [FIND_FEASIBLE_MINIMUM_TOURS (Subset_Pointer)]
    if (the current subset point tally equals the best observed)
    if (expected cost is less than the best observed) record the cur-
  rent solution as the best observed so far;
      Increment Subset_Pointer;
until (the current subset point tally < the best observed);
  end.
```

The algorithm FIND_FEASIBLE_MINIMUM_TOUR comprises three algorithms in its own right. The first finds a minimum (or near-minimum) expected-cost Hamiltonian tour through all of the cities in the current subset; this algorithm uses the minimum expected travel costs between each pair of cities (not necessarily the direct costs). The second algorithm expands that Hamiltonian tour by filling in the intermediate cities that are visited in the minimum cost link between every pair of cities in the Hamiltonian tour and records the points accumulated over all of the cities visited (including the intermediate cities), honoring any precedence relations that may be in effect. The third and final algorithm determines whether the expanded tour satisfies the budget probability constraint.

Bounding the Search Space

For a TSP problem having N free cities plus one home city, our salesman friend might need to consider almost all of the 2^N possible city subsets using our algorithm before he finds a first subset that has a feasible tour. In practice, he can reduce the actual number of subsets that need be considered through five means. First, he can exploit the precedence relations to reject any subset that has the antecedent city in a precedence pair but not its partner.

Second, he can try to determine an upper bound on the number of unique cities that could ever appear in a feasible tour and reject subsets involving more cities than this limit. An upper bound can be determined by solving a parametric assignment problem with the free parameter being the maximum number of unique cities that can be present in any feasible tour. This approach has one other advantage if this upper bound is sufficiently tight: He can place a cap on the maximum number of points any feasible tour can accumulate and use this cap to determine the maximum possible difference between his solution and the optimal solution.

Third, he can set a lower bound on the number of points that can be accumulated in a feasible tour, thereby rejecting city subsets prior to sorting that have point totals less than this lower bound. He can easily find a lower bound using a "quick and dirty" heuristic algorithm; we use the second "greedy" algorithm described earlier.

Fourth, our friend can also use this lower bound on city points to reduce the range of unique subset labels that he needs to consider when enumerating over city subsets. We assume without any loss of generality that he has already sorted the cities by their point count in ascending order. We will let P_i denote the points for city i (city 1 having the smallest and city N the largest number of points), P_h the points for the home city (which will appear in every subset), and P the lower bound

on the total points he will be able to collect. Now, suppose our friend considers the subset containing just the home city and the M lowest valued cities. If he considers the largest value of M for which the following inequality holds:

$$P > P_h + \sum_{i=1}^{M} P_i \tag{2}$$

then he can ignore any tour ID less than ϵ, where

$$\epsilon = 2^M - 1 \tag{3}$$

Finally, our friend might try to find cities that must be in an optimal solution. Suppose he has determined an upper bound U on the number of unique cities that can be present in any feasible tour. Suppose he also has his lower bound P on the total points that he will be able to collect. With a little reflection, it becomes clear than any city i for which the relation

$$P_i > P_h + \sum_{k=N-U}^{N} P_k - P \tag{4}$$

holds must be present in the optimal solution and, like the home city, can be included as a de facto member in every city subset. Suppose our friend has located K cities that satisfy this relation. Letting ϵ' denote the maximum range of the subset labels, he can now reduce ϵ' from 2^N to 2^{N-K}. At present, we have not included this bounding method in our algorithm; we leave this for a future extension.

The first three bounding techniques do not reduce the range of subset labels that must be considered by the algorithm when enumerating over the possible city subsets, but do reduce the number of subsets that are actually recorded and subsequently sorted. The first two methods can also drastically reduce the number of city subsets that have to be examined by FIND_FEASIBLE_MINIMUM_TOUR to find the first subset that has a feasible tour. The last two bounding methods actually reduce the range of subset labels examined when enumerating city subsets, the former one reducing the range from below, and the later one reducing the range from above.

Algorithm Wrap-up

In this section, we have presented a heuristic approximation to the optimal strategy described in Sec. III. We summarize the main points of this algorithm as follows:

1) It replaces the decision algorithm FEASIBLE_DECISION with a weak but more computationally tractable test, FIND_FEASIBLE_MINIMUM_TOUR.

2) FIND_FEASIBLE_MINIMUM_TOUR simply involves finding the minimum expected-cost Hamiltonian tour for a subset, expanding the tour by filling in the intermediate cities that are visited in the minimal cost path between each pair of cities in this tour, and determining whether this expanded tour is feasible.

3) The number of subsets that need to be considered by our algorithm is reduced using a number of bounding methods; our current algorithm uses the first four methods described.

IV. Computational Results

Objectives

Our objectives in this section are to demonstrate the performance of the algorithm we have built for our salesman friend under a number of different and challenging problem conditions. We concentrate our attention here on both the performance of the algorithm in an absolute sense (execution time, quality of the solution) and the contribution afforded the basic algorithm by the various bounding methods we described in the last section.

Nominal Problem Set

The nominal problem set is provided in Ref. 1.

Bounds for the Nominal Problem Set

Using a parametric assignment algorithm, we can first determine that no feasible solution to the nominal problem set can have more than 9 cities. This also leads us to conclude that the optimal tour cannot accumulate more than 88 points.

Using the second greedy algorithm described earlier, we can also determine that the best feasible solution should be able to accumulate at least 68 points. This also leads us to conclude that we should not bother examining and subset involving only combinations of ATL, DEN, MSP, MSY, PHX, SEA, DFW, BOS, and DTT (the home city) since these cities in total represent only 64 points. From Eqs. (2) and (3), we can immediately ignore any tour ID less than 255 ($2^8 - 1$).

Results

Figure 1 illustrates the performance of our greedy heuristic and optimization algorithms for the nominal problem set. GREEDY I and GREEDY II denote the heuristic tour construction algorithms that use the maximum points-to-cost criterion and minimum cost, respectively, to select the next city to add to the tour. SUBTOUR I denotes the optimization algorithm described in Sec. III without any bounding methods added, whereas WILLY denotes the same algorithm using the first four bounding methods just described. WILLY and SUBTOUR I both arrive at the same solution, but SUBTOUR requires roughly twice as much time to get there. From the point of view of execution time, WILLY is more akin to its two greedy cousins than its unadorned brother.

WILLY required roughly 7 s to set up the problem and determine the relevant bounds, 9 s to enumerate and sort the potentially feasible city subsets, and 5 s to actually search through subsets to locate its final solution. (All reported times are for execution on a Macintosh personal computer.) In short, a significant portion of the computation time is spent simply enumerating the subsets.

The basic search strategy locates the tour

$$\text{DTT} - > \text{CHI} - > \text{LAX} - > \text{PHX} - > \text{DEN} - > \text{DFW} - > \text{MSY} - > \text{ATL} - > \text{BOS}$$

which accumulates 88 points and leads to an expected cost of \$2703. Since our second bounding method provides us with an estimate that no feasible tour can have more than 9 cities, we are reasonably confident that this tour is either an optimal tour or very close to an optimal tour.

Scenarios 2 and 3: Budget Constrained

Our optimization algorithm is, in principle, a "brute force" search that examines possible solutions until it finds a first feasible tour. The drawback here is the search can start some distance away from any feasible solution, requiring many futile calls to FEASIBLE_DECISION or FIND_FEASIBLE_MINUMUM_TOUR (for the exact and approximate algorithms, respectively) in the process. This problem is particularly acute when the budget constraint is rather tight. The second bounding method we discussed in Sec. III tries to ameliorate the problem by finding the maximum number of unique cities that can possibly be present in a feasible tour, thereby allowing the algorithm to skip over clearly unfavorable city subsets during its search.

We illustate the computation efficiency brought about by this upper bound in Figs. 2 and 3 where we set the tour budget to \$2000 and \$1000, respectively.

With the budget equal to \$2000, the basic version of the optimization algorithm (SUBTOUR I) evaluates 242 possible city subsets before it locates the best solution in its search. By contrast, the version using the city upper bound (WILLY) examines only 116 city subsets before it locates the best solution in its search.

The differences between SUBTOUR I and WILLY are even more pronounced when the budget is equal to \$1000 (Fig. 4).

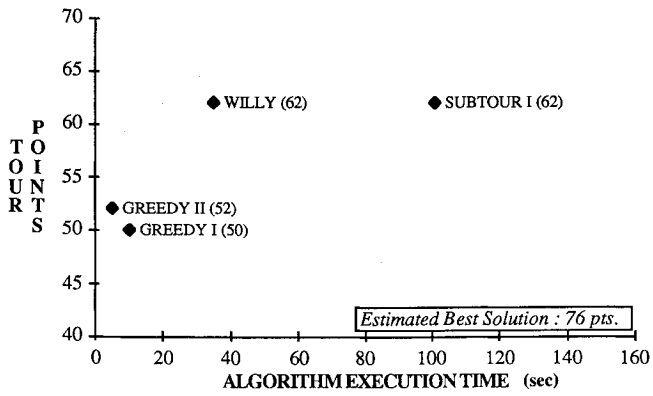


Fig. 2 Algorithm performace with a $2000 budget.

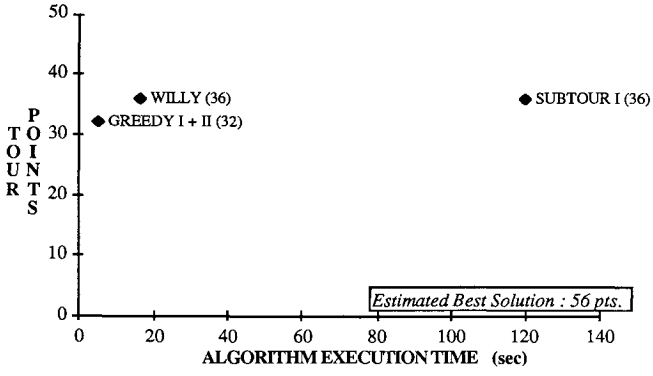


Fig. 3 Algorithm performance with a $1000 budget.

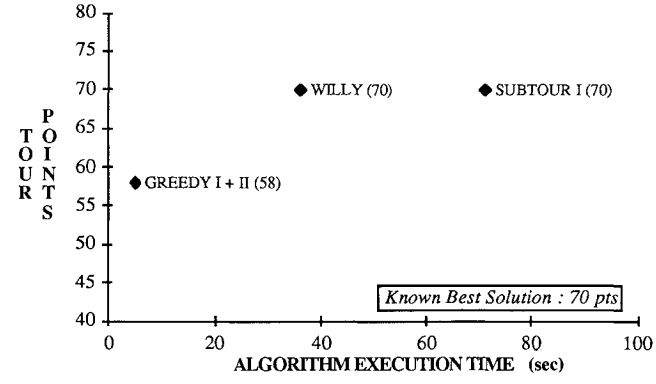
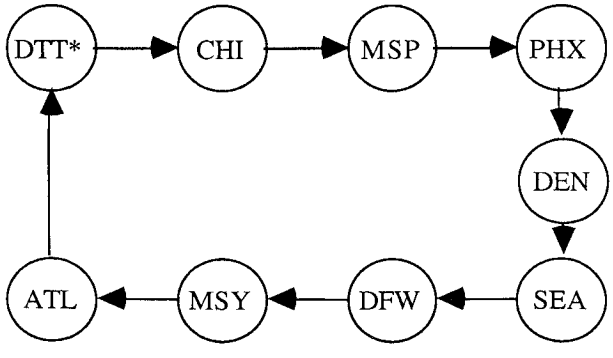
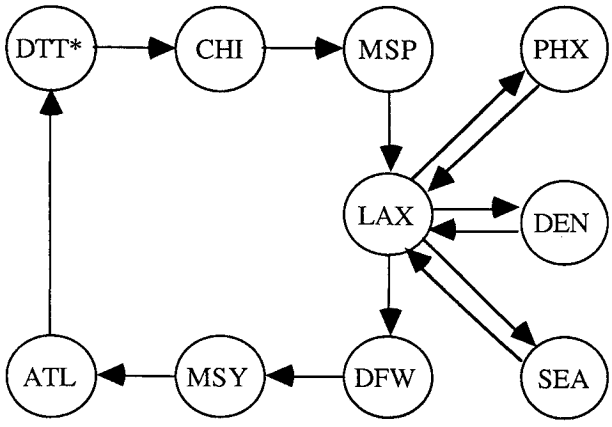


Fig. 4 Algorithm performance for scenario 4.



*"HOME" CITY

Fig. 5 Optimal Hamiltonian tour for scenario 4.



*HOME CITY

Fig. 6 Optimal general tour for scenario 4.

Under this condition, the basic algorithm evaluates 849 city subsets before it locates the best solution, whereas the bounded version examines only 37 subsets.

Scenario 4: Tour with a Hub and a Barreir

Two potential problems can befall our traveling salesman friend: 1) the optimal tour may pass through a "hub" city several times, and 2) a city having a lage number of points may have an unusually high visiting cost, resulting in its exclusion from any feasible tour (a "barrier"). Our solution algorithm easily tackles the first problem by first constructing strictly Hamiltonian tours using the modified cost matrix C' described in Sec. II, and then expanding each tour by filling in the intermediate cities that are visited in the process.

The second problem is implicitly tackled through the brute force search adopted by our algorithm: City subsets containing a desirable city so described will appear at the top of the sorted list (and thus be examined early on in the search) but will quickly be rejected.

We illustrate the performance of our optimization algorithm for a condition in which the budget is set to the nominal $3000, the precedence constraint involving BOS and LAX is maintained (LAX must be followed by BOS for points to accrue to LAX), all costs entering and leaving LAX are set to $100, all costs entering BOS are also set to $100, but all costs leaving BOS are set to $2000. In short, BOS cannot be included in a tour without displacing most of the other cities. As it happens, the optimal solution is to visit all cities but BOS and accumulate 70 points in the process.

The performance of our heuristic and optimization algorithms for this problem is shown in Fig. 4. The optimization algorithms are indeed able to locate the optimal solution with WILLY, again, reaching the solution in less time that SUBTOUR I.

The optimization algorithms locate the Hamiltonian tour shown in Fig. 5. The general tour created from an expansion of that tour is shown in Fig. 6; here, LAX clearly serves as a hub for a significant portion of the tour.

V. Summary and Conclusions

We have presented an efficient solution strategy for a modified traveling salesman problem that is capable of finding optimal or near-optimal tours. The basic strategy recognizes that points are accrued from each city visited only once and sets up a search strategy to exploit this special structure. The search to locate a feasible tour accumulating the maximum number of points has a very simple termination condition: Stop at the first feasible tour found. The strategy for finding a minimum expected-cost tour that accumulates the maximum

number of points requires only a trivial extension to the basic search.

The computational efficiency of the basic search algorithm can be improved considerably by bounding the problem to exclude clearly unattractive tour combinations from consideration. These bounds are particularly important when the budget is a very limiting constraint.

The algorithm described here suffers from one considerable handicap: It explicitly enumerates and sorts all possible city subsets before proceeding with the search. In computation experiments not reported here, we have observed that the enumeration and sorting steps alone require ~9 s for an 11-city problem. One remedy is to conduct a graph search to implictly enumerate the subsets.

References

[1]Deutsch, O., "The A.I. Design Challenge—Background, Analysis, and Relative Performance of Algorithms," *Proceedings of the AIAA Guidance, Navigation, and Control Conference,* AIAA, New York, Aug. 1987, pp. 465-476.

[2]Floyd, R.W., "Algorithm 97: Shortest Path," *Communications of the ACM,* Vol. 5, 1962, p. 345.

[3]Syslo, M.M., Deo, N., and Kowalik, J.S., *Discrete Optimization Algorithms,* Prentice-Hall, Englewood Cliffs, NJ, 1983.

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

[5]Knodel, W., "Algorithm 7: Traveling Salesman Problem," *Computing,* Vol. 3, 1968, pp. 151-156.