UNIVERSITE PARIS DAUPHINE

Laboratoire d'Analyse et Modélisation de Systèmes pour l'Aide à la Décision

CNRS UMR 7024

CAHIER DU LAMSADE

161

Juin 1999

A new greedy-solution representation for the travelling salesman problem

Aristidis Likas, Vangelis Th. Paschos

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

Table of Contents

Une nouvelle représentation de tours pour le problème du voyageur du commerce et son utilisation dans des heuristiques

Résumé

Nous présentons une nouvelle approche pour le problème du voyageur du commerce basée sur une représentation originale gloutonne de l'espace des solutions. Ce type de représentation induit une définition différente des structures du voisinage, la notion de voisinage étant une notion-clé dans plusieurs méthodes utilisant des approches de recherche locale ou aléatoire. Nous proposons aussi une stratégie parallélisable de recherche qui s'appuie sur une recherche locale avec des ré-initialisations aléatoires et qui exploite les caractéristiques de la représentation proposée. Des résultats expérimentaux préliminaires sur plusieurs problèmes-tests montrent que cette représentation, couplée avec la stratégie de résolution établie, donne des solutions très proches de l'optimum en des temps d'exécution modérés.

Mots-clé : voyageur du commerce, optimisation combinatoire, heuristique.

A new representation of traveling salesman tours and its use in heuristics

Abstract

A new approach is presented to the traveling salesman problem relying on a novel greedy representation of the solution space and leading to a different definition of neighborhood structures required in many local and random search approaches. Accordingly, a parallelizable search strategy is proposed based upon local search with random restarts that exploits the characteristics of the representation. Preliminary experimental results on several sets of test problems, among which very well-known benchmarks, show that the representation developed, matched with the search strategy proposed, attains high quality near-optimal solutions in moderate execution times.

Keywords: traveling salesman, combinatorial optimization, heuristic.

1 Introduction

The traveling salesman problem (TSP) is stated as: given the distances between N cities, find the shortest (in the sense of total distance) closed tour through the set of N cities so that each city is visited exactly once ([7]). TSP is one of the most known NP-complete problems and has strongly influenced the emergence and evolution of domains as complexity theory, operations research, combinatorial optimization. In most published scientific works about TSP, the matrix of intercity distances is considered symmetric. This is the version of symmetric TSP with which we deal in this paper.

Since exact TSP-algorithms require excessive computation times on powerful machines, many approximation algorithms have been developed that aim at providing, for any instance, near-optimal solutions (a few percent from optimum) in reasonable computation times. But even this aim is far from being always achieved. In fact, one of the first negative results in polynomial approximation theory (the chapter of complexity theory dealing with finding, in polynomial time, sub-optimal solutions for NP-complete problems), is that for every polynomial time approximation algorithm for TSP, there exists a TSP-instance for which the worst-case approximation ratio "objective value of approximate solution over optimal TSP-value" is arbitrarily large ([3]). This strongly negative approximation context has been lightened by some positive approximation results for restrictive TSP-cases. We quote here the classical Christofides' algorithm achieving worst-case approximation ratio bounded above by $3/2$ for metric TSP ([2]), the Papadimitriou and Yannakakis' algorithm achieving worst-case approximation ratio $7/6$ for TSP with edge-distances 1 or 2 ([9]) or, even, the polynomial time approximation schema of Arora for Euclidean TSP ([1]).

Let us note that modeling natural problems in terms of TSP instances arises not only in economy, complex systems administration, decision making, etc., but also in mechanics, physics, chemistry or even in biology. This multiplicity and diversity of problems having TSP as common mathematical ground explains the great scientific interest that researchers (both mathematicians and computer scientists) always have for TSP and the intensive research-work about effectively solving it.

In general, TSP-heuristics can be classified as: tour construction procedures, tour improvement procedures and composite procedures based upon both construction and improvement techniques ([10]).

In the case of construction procedures, a tour is built gradually by selecting one city at each step and by appropriately inserting this city into the current tour. The local improvement techniques are based upon perturbations of the current solution that aim to generate a new improved tour. The most widely used techniques are the k_opt exchange heuristics and in particular the 2_opt, 3_opt and Lin-Kernighan heuristics ([8]). The k_opt heuristics generate a new tour from the current one by replacing k edges in the tour by k new edges. Usually, these heuristics are used as perturbation mechanisms in local search procedures, i.e., they are applied iteratively until no further improvement is possible. They have also been employed meta-heuristics, e.g., simulated annealing, tabu search and genetic algorithms. Finally, the composite heuristics are more recent and were developed from the combination of characteristics of the previous categories ([4, 5]). A detailed and very interesting presentation of the most-known techniques used to approximately solve TSP can be found in [6].

The approach proposed in this paper belongs to the category of composite heuristics. The current solution is not directly represented as a sequence of cities as it happens in most heuristics, rather each string encodes the strategy that will be used for tour construction. Therefore, the search is oriented towards the identification of tour construction strategies that lead to near-optimal solutions. Such strategies result as appropriate modifications to the greedy heuristic

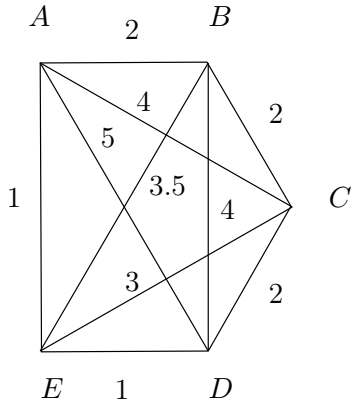

Figure 1:

strategy which assumes that the best decision is made at each step during tour construction. The proposed representation allows for significant improvements over the greedy heuristic by allowing the possibility of inferior decisions at each step of tour construction that may finally lead to tours of optimal length. The identifications of the appropriate decisions to be made at each step of tour construction constitutes the objective of the search algorithms that are based upon local improvements. Nevertheless the perturbation principle (i.e., the way new solutions are generated from the current one) is totally different from the $k_$opt principle. In general, heuristics working in a very satisfactory way when dealing with the classical representation (sequence of cities) may be unsatisfactory or even senseless dealing with our representation. Consequently new heuristics, compatible with this novel representation need to be developed.

2 The greedy-solution representation

Consider a general TSP (symmetric or asymmetric) with N cities. In the proposed method each tour is described by a string $S = (s_1, \ldots, s_{N-1})$ with $0 \leq s_i \leq N-i-1$ for each $i = 1, \ldots, N-1$ which encodes a modified greedy search strategy. In order to map this string to a valid tour and evaluate the corresponding tour length, the following procedure is used:

- a city is randomly selected that will always correspond to the first city in the tour specification procedure; we shall refer to this city as the *initial* city;
- the remaining $N-1$ cities are ordered with respect to their distance from the initial city and we select the second city of the tour according to the value of s_1; if $s_1 = 0$ we select the closest city, if $s_1 = 1$ we select the second closest, and so on; the city selected according to the s_1 value is added to the tour and becomes the new current city;
- in the same way, in order to specify the ith city of the tour, we rank the remaining $N-i$ cities with respect to their distances from the current city (selected at step $i-1$) and select one of them according to the value of s_i, $0 \leq s_i \leq N-i-1$.

Example 1. Consider the TSP-instance of Figure 1 and suppose that the tour is described by the string (0,2,1,0); finally, suppose that the initial city is A.

Following the representation string, starting from A we move to its closest city, E. Then, starting from E, we will move to its third-closest city (excluding A); this city is B. From B, we move to its second-closest city (excluding A and E); this city is D. Finally, from D, we move to

its closest city (excluding A, E and B); this city is C. Thus, given that the initial city is A, the tour represented by the string (0,2,1,0) is (A, E, B, D, C, A). ∎

Remark 1. Under the above interpretation schema, the tour corresponding to the string with $s_i = 0$ for each i is the greedy solution starting from the specified initial city. We shall call this string zero_State. ∎

Remark 2. The above interpretation schema works for asymmetric TSP as well. ∎

Remark 3. The evaluation of a string under the proposed representation is computationally more expensive compared to the conventional descriptive approach since at each step a sorting of the remaining cities is required with respect to their distances from the current city. ∎

Nevertheless, complexity can be significantly reduced by adding a preprocessing stage in which a list is constructed for each city that contains the other cities sorted according to their distance from that city. During tour construction, if $s_i = k$, we scan the list of the current city and ignore the entries of the list corresponding to cities that have already been allocated. The kth entry in the ordered list of unallocated cities corresponds to the city that will be added to the solution tour.

On the other hand, the method described above has the advantage of incorporating knowledge about the relative values of the intercity distances. So, we almost always know from the beginning (unless the problem is a very strange one) that near-optimal solutions are expected in the neighborhood of zero_State. In fact, as experiments indicate, even simple local search starting from strings in the neighborhood of zero_State is sufficient for obtaining solutions about 5–10% from optimal in a relative small number of iterations for many problem instances with up to 200 cities. In the case where local search is repeated (random restarts) much better results are obtained as shown in the next sections.

Dealing with the neighborhood of a given state (used in local improvement techniques), the proposed representation leads to a more natural definition than the ones used in common TSP-heuristics like the two or three city exchange heuristic ([8]). In our case, since the representation is numerical, it is natural to define the neighboring states as those that differ from the current one in the values of one or more string positions. In the case where only one string position i is allowed to change its value, but can take any value in the range $[0, N - i - 1]$, the size of the neighborhood is $O(N^2)$. In our tests we have considered a stricter neighborhood definition: the value s_i of a string position i is allowed to take integer values in the range $R_i = [\max\{0, s_i - a\}, \min\{s_i + a, N - i - 1\}]$ with $a \ll N$, since from the representation definition it must always hold that $0 \leq s_i \leq N - i - 1$. This leads to a neighborhood definition with size $O(N)$, since two neighboring strings are allowed to differ in the value of only one position i, and the number of allowable different values for s_i is less than, or equal to, $2a$.

Another issue that must be emphasized deals with the initial city specification that is necessary in order to evaluate a given string. Obviously, different assumptions about the initial cities lead to different tours of different lengths. It is possible to make the evaluation of a given string independent of the initial city selection as follows: we evaluate all the N tours that result by considering every city i, $i = 1, \ldots, N$, as the initial one, and find the tour of minimum distance which will constitute the final evaluation of the given string. Of course, this evaluation schema leads to solutions of better quality at the expense of being N times slower. Another possible approach is to perform N different local searches, each one assuming a different initial city for string evaluation. In the last two cases, it is natural to use parallel processors in order to face up with increasing computational times.

3 An optimization strategy with random restarts

Once a solution representation and a neighborhood structure have been defined, there are many alternative search schemas that can be used for exploration of the state space. For example, it is possible to test several variants of local search, various kinds of random search, e.g., simulated annealing, tabu search, genetic algorithms, etc.

In our case we have considered an optimization strategy based upon local search with random restarts where the initial state of each restart was not selected completely at random, but in a more specific way in order to deal with the special characteristics of the solution space. Moreover, a search phase, called phase 1, takes place at the beginning of the procedure in order to specify a promising initial city that will be subsequently used for the construction of the tour corresponding to each solution string.

Concerning local search starting from an initial state, we have considered the steepest descent approach according to which the whole neighborhood of the current state is visited at each step and we move to the state of minimum cost if this cost is lower than the cost of the current state, otherwise we consider that a local minimum state has been encountered. In all our experiments the value of the parameter a, described in the previous section, which determines the neighborhood size, was set equal to 4. If the local minimum state is of lower cost than any other previous local minimum state, then we consider it as the current best state. The global search terminates if no state better than the current best one is found for a prespecified number of restarts.

The proposed search algorithm proceeds in two phases: In phase 1 we perform local search with random restarts, where at each restart a different city is considered as the initial one. Moreover the initial state for each restart is obtained through small perturbations of the zero state. All cities are sequentially considered as initial ones and the city that leads to better results is permanently considered as the initial one in the computations of the next phase. In phase 2, each tour is constructed by considering the same initial city and the starting state at each restart is obtained through small perturbations of the current best state.

Random perturbations were carried out in a manner analogous to the mutation operation in genetic algorithms, i.e., by deciding for each string position i with low probability p_m whether its value will change or not. In the case of positive decision, the new s_i-value is randomly selected in the interval $[\max\{0, s_i - 3\}, \min\{s_i + 3, N - i - 1\}]$. The mutation probability was taken equal to $p_m = 0.15 * (\#\text{Restarts}/\text{max\_Restarts}) + 0.05$, where #Restarts denotes the number of restarts already performed for the current best state. If a new best state is found then we set #Restarts = 0 and we start counting restarts from the beginning.

An overall specification of the whole search strategy is described in what follows.

- Phase 1: **search for a good initial city**
  - Initializations
    - #Restarts $\leftarrow 0$
    - best_State $\leftarrow$ zero_State
    - best_Initial_City $\leftarrow 0$
    - max _Restarts $\leftarrow 2 * N$
    - $i = 0$
  - repeat steps 1–5 below until #Restarts = max_Restarts
    1. initial_City $\leftarrow i$;
    2. define Initial_State by mutating zero_State
    3. starting from the Initial_State perform local search to find Final_State
    4. if Final_State better than best_State then

        * best_State ← Final_State
        * best_Initial_City ← i
        * #Restarts ← 0
    5. $i \leftarrow i + 1 \bmod N$

- Phase 2: **global search using best_Initial_City as the initial city**
  - Initializations
    * #Restarts ← 0
    * max_Restarts ← 500
    * initial_City ← best_Initial_City
  - repeat steps 1–3 below until #Restarts = max_Restarts
    1. define Initial_State by mutating best_State
    2. starting from the Initial_State perform local search to find Final_State
    3. if Final_State better than best_State then
       * best_State ← Final_State
       * #Restarts ← 0

4 Experimental results

The proposed method has been tested on two categories of TSP-instances.

The first category concerns small instances with 10–15 cities, for which the optimal solution was obtained using a branch-and-bound technique. Here we have tested three kinds of random instances:

1. 209 randomly generated TSP-instances, i.e., complete graphs on 10–15 vertices, with random integer edge-distances from the interval [1,150];

2. 202 Euclidean TSP-instances, i.e., complete graphs of order 10–15, vertices of which are points of the rectangle $[0,1] \times [0,1]$ with random (x, y)-coordinates; edge-distances are, obviously, the Euclidean distances between points;

3. 173 TSP-instances with edge distances randomly chosen in {1,2}.

Ten experiments were performed for each instance and the method exhibited excellent performance. More specifically, in 95% of the instances all 10 runs were successfully terminated at the optimal solution, while in the remaining 5% of the instances, at least 7 of the 10 runs gave the optimal solution and the remaining runs terminated at solutions less than 1% from optimal. Moreover the execution time was very small (less than 2 seconds of CPU time) for all graphs.

The second category of experiments have been conducted with 14 benchmark-problems from the TSP data library (TSPLIB, [11]). For these benchmarks, the number of cities ranged from 100 to 200. For each instance 10 runs have been performed.

The solution-quality criterion is the so-called optimality ratio, defined as:

$$\rho = 100 \left(\frac{\text{solution\_Cost} - \text{optimal\_Cost}}{\text{optimal\_Cost}} \right) \%$$

where solution_Cost denotes the cost of the obtained solution and optimal_Cost is an estimate of the value of the optimal tour (available in [11]). For each of the problems tested, three ratio-values are retained, namely, the minimum over all runs, called best ρ, the maximum one, called

Problem	Best ρ (%)	Average ρ (%)	Worst ρ (%)
eil101	0.6	1.2	1.7
kroA100	0.4	0.8	1.1
kroA150	0.8	1.8	1.1
kroA200	1.9	2.1	2.5
kroB100	0.4	0.8	1.5
kroB150	1.0	3.3	4.1
kroB200	3.1	4.0	5.2
kroC100	0.5	0.8	1.3
kroD100	2.3	3.2	3.6
kroE100	0.2	1.0	1.8
lin105	0.3	0.6	1.0
rat99	0.9	1.7	2.5
rat195	2.2	3.4	3.6
rd100	0.4	2.2	3.2

Table 1: Optimality ratios for several test problems from the TSPLIB library.

worst ρ, and finally, the average one, called average ρ, computed as the sum, over all runs, of the ratio-values divided by the number of runs.

Table 1 displays statistical results for several benchmarks, where 10 experiments were run for each problem. As one can see from this table, the proposed method provides high quality solutions to several problem categories with very good best case and worst case ratios and, in addition, relatively low standard deviation values. Finally, it must be noted that all ratios were computed by considering as optimal values those provided by the TSPLIB library which are often estimates (lower bounds) of the optimal solution values ([11]). So, the real approximation ratios for the problems tested are in some cases even better than the ones presented in table 1.

Dealing with execution time, the average values were up to 1000 CPU seconds for 200-city problems. Experiments were carried out on a ULTRA SPARCstation with 512Mb RAM. Let us note that it is possible to obtain results of better quality by increasing the neighborhood size and performing more than 500 random restarts from the current best state, which will lead to an increase of the required CPU time. This is not a matter for problem sizes up to 150 cities, but leads to increasing CPU times for larger problem-instances. Therefore, we have decided to specify the parameters of the search strategy under the constraint that the execution time will be less than 1000 CPU seconds for the largest problems examined (200-city problems). Moreover, we have chosen to use the same number of restarts (in phase 2) regardless of the problem size.

Finally, observe that the neighborhood exploration phase exhibits high degree of parallelism, since the cost of each neighboring state is computed independently and therefore a pool of processors can be easily employed to perform this task, leading to the expectation of high speedup values and therefore significant improvements in execution time.

5 Conclusions

We have proposed and studied a new state space representation for the TSP that is based upon a ranking of the intercity distances, in order for the corresponding tour to be specified. The

method is general and can be applied to any kind of TSP, even asymmetric one. Experiments have been conducted (on problems with up to 200 cities) by performing local search with random restarts where a simple neighborhood definition has been used (leading to neighborhood size less than $8N$) and a specific strategy for specifying restart states has been developed in order to exploit the characteristics of the new representation. Of course, a systematic validation of our method, by conducting further experiments is needed. But our preliminary experimental results show that this approach provides solutions of high quality in relatively small execution time. Moreover, as mentioned above, the devised algorithm is easily parallelizable.

There are many open problems which can be addressed concerning the proposed representation and which will constitute the subject of our future work. First of all, we aim at integrating the proposed representation with global optimization techniques (tabu search, simulated annealing, genetic algorithms and others) that have already been successfully employed for the solution of the TSP using the conventional representation and the corresponding heuristics. In addition, we aim at examining alternative neighborhood definitions that may lead to solutions of better quality. Finally we are interested in implementing the method on parallel machines following the directions suggested in previous sections.

Another promising direction of future research is the use of the proposed greedy representation to tackle other combinatorial optimization problems for which greedy heuristic approaches exist. So, further studies are to be performed concerning the application of the method to problems, structurally different from TSP, for which tailored heuristics (as the Lin-Kernighan one for the TSP) have not been developed. Such studies, except their evident operational interest, will render clearer the importance and effectiveness of the greedy representation, as well as its field of applicability.

For example, for the maximum independent set, instead of constructing the greedy solution (minimum-degree heuristic) by including in the solution set the best vertex at each step (the one minimizing the number of the immediately excluded vertices), we could use the proposed representation and rank the vertices according to the number of the remaining neighboring vertices that will be excluded in the case where the specific vertex will be added to the solution set. From the string representation, a solution is constructed by selecting at each step i the vertex suggested by the corresponding s_i value. An analogous thought process could be followed for minimum set-covering, where the ranking criterion could be the number of remaining elements of the ground-set in the case where a specific subset will be included in the solution.

Finally, let us conclude this paper with a very interesting theoretical problem, namely the achievement of theoretical approximation guarantees for the proposed method. This kind of theoretical analysis is all the more interesting since such results are very rare for the scientific area dealing with meta-heuristics.

References

[1] S. Arora. Polynomial time approximation schemes for Euclidean TSP and other geometric problems. In *Proc. FOCS'96*, pages 2–11, 1996.

[2] N. Christofides. Worst-case analysis of a new heuristic for the traveling salesman problem. Technical Report 388, Grad. School of Industrial Administration, CMU, 1976.

[3] M. R. Garey and D. S. Johnson. *Computers and intractability. A guide to the theory of NP-completeness*. W. H. Freeman, San Francisco, 1979.

[4] M. Gendreau, A. Hertz, and G. Laporte. New insertion and post-optimization procedures for the traveling salesman problem. *Oper. Res.*, 40:1086–1094, 1992.

[5] D. S. Johnson. Local optimization and the traveling salesman problem. In G. Goos and J. Hartmanis, editors, *Proc. ICALP'90*, number 443 in Lecture Notes in Computer Science, pages 446–461. EATCS, Springer, 1990.

[6] D. S. Johnson and L. A. McGeoch. The traveling salesman problem: a case study. In E. Aarts and J. K. Lenstra, editors, *Local search in combinatorial optimization*, pages 215–310. John Wiley, 1997.

[7] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, editors. *The traveling salesman problem: a guided tour of combinatorial optimization*. Wiley, 1985.

[8] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling salesman problem. *Oper. Res.*, 21:498–516, 1973.

[9] C. H. Papadimitriou and M. Yannakakis. The traveling salesman problem with distances one and two. *Math. Oper. Res.*, 18:1–11, 1993.

[10] J.-Y. Potvin. Genetic algorithms for the traveling salesman problem. *Ann. Oper. Res.*, 63:339–370, 1996.

[11] G. Reinelt. Tsplib - a traveling salesman problem library. *ORSA J. Computing*, 3:376–384, 1991.