A Deep Reinforcement Learning Inspired Local Search for TSP

Jinchang Fan, Han Wang, Xiaoyu Qiao, Ziyan Lin

May 2, 2019

Abstract

Many solutions for Traveling Salesman Problem (TSP) have been raised and discussed for a long time, and one of the best approximate solutions is Genetic Algorithm for global search combined with 2-opt for local search. Though accuracy is satisfying, the efficiency of this method is limited by the iterations of 2-opt Heuristic to reach a local optimal. The newly raised Deep Reinforcement Learning also performs well when applied to TSP, but it requires interminable processing time. In this project, half trained Deep Reinforcement Learning Pointer Network is borrowed to provide elitist tour candidates for 2-opt, which would increase the efficiency to find a solution.

1 Introduction

Traveling Salesman Problem is a famous optimization problem that has been discussed for a long time. The problem asks: given a set of cities and cost of travelling from one to another, what is the best way in terms of cost to visit all cities and come back to the starting point? In contract to its simple description, TSP is found to be an NP-hard problem[1]. Considering its popularity and various real-world applications, TSP now performs as a touchstone for different optimizing algorithms.

So far, algorithms for exact solution of TSP[2][3][4] have been widely developed and discussed respectively, but the solution is still infeasible to reach when the number of cities grows large due to the exponential complexity[5].

At the same time approximate algorithms[6][7] are developed to obtain an acceptable solution in relatively short time, including many bioinspired algorithms[8][9][10]. Genetic Algorithm (GA) [11] is one of the best algorithms in many combinatorial optimization problems with huge research work thanks to its efficient strategy to reach global optimization, and has been shown to perform quite good on TSP as well[12].

The basic procedure of GA on TSP could be described as following: first start with a random initial population and apply local optimizing strategy for each of them, then select instance as "parents" to generate new off-springs by cross-over breeding. Apply local optimizing strategy again to those off-springs, and partially update population by off-springs after mutation. Repeat generations and population would converge to the global optimization. In the Genetic Algorithm for TSP, local search strategy is critical. A current popular local search method is 2-opt algorithm[13][5] with basic idea of performing exchanges of two edges to obtain a shorter tour. This greedy strategy is simple but usually the simplicity is obtained with the danger of failure or inefficiency in finding optimal solutions. The detail of this algorithm is discussed in Section 2.2.

Reinforcement Learning (RL) in Neural Network[14] is a goal-oriented algorithm that learns to attain some complex goal or maximize value over dimension by discovering which actions yield the most reward. The time consuming work for 2-opt algorithm could be accelerated if the experience from an ideal RL model is learned, and efficiency of the whole Genetic Algorithm is improved consequently. Such improvement would be the main focus of this project. Methodology would be discussed and practice on sample case are performed. The new algorithm is compared with previous work in the evaluation on distance of provided solution and time consumed.

The rest of this paper is organized as follows. Section 2 introduces a classical algorithm for TSP, namely the combination of 2-opt as local search and Genetic Algorithm for global search. The whole procedure and short coming are discussed in detail. Section 3 makes a description of the RL-2opt-GA algorithm, and illustrates how it may make an improvement. Section 4 shows how algorithms are realized and compared, including dataset and computation platform. Section 5 evaluates the result of experiments, and the whole project is summarized in Section 6.

2 2opt-GA algorithm for TSP

2.1 Genetic Algorithm

Genetic Algorithm is an important tool in solving both constrained and unconstrained optimization problems with inspiration of natural selection. The procedure of Genetic Algorithm in the context of TSP is illustrated in Figure 1. Some important definitions are stated as below.

- Tour: a tour that visits each city exactly one time and returns to the starting point in the end
- Population: a collection of tours

```
procedure TSP-GA
 begin
   initialize population P randomly
   foreach tour $t \in P$ do $2\text{-}opt(t)$
   repeat #repeat for generation
      repeat p times: #repeat for population
        randomly select $t_{p1}, t_{p2} \in P$
        $t_{child} := breed(t_{p1}, t_{p2})$
        $2\text{-}opt(t_{child})$
        With prob. of 0.01: $Mutation(t_{child})$
        replace one tour in P by $t_{child}$;
      end
   until converged;
 end
```

Figure 1: TSP-GA

- Parents: two tours that could be used to breed children tour
- Mutation: variation in tours' population, here is a two point swap at random position
- Elitism: keep best k tours into the next generation

When initializing population in GA, tours with diversity are desired. But connecting points at completely random leads to a lot of work for 2-opt. To balance this problem a half-greedy approach is utilized as illustrated in Figure 2. A starting city is selected randomly at the beginning. Select next city to connect from the 3 cities with the smallest distances to the starting city. Repeat this process until all cities are visited. Such half-greedy random selection technique will preserve the diversity and utility of tour sets.

In the breeding part of Generic Algorithm, Distance-Preserving Crossover (DPX) has been used. The idea of DPX is to keep child tour having same "distance", in terms of number of common edges, from its two parents. When given two parent tours t_{p_1} and t_{p_2}, the initial edges of their child t_{child} tour will be the common edges in t_{p_1} and t_{p_2}, and all edges that t_{p_1} and t_{p_2} do not agree with each other are removed. Reconnect components in t_{child} using greedy approach, with restriction that all reconnected edges

```
procedure Half-greedy Tour Selection
 begin
  input maps  #list of cities
  N = size(maps)
  initialize tour = {}
  length_of_tour = 0
  randomly select starting city $c \in maps$
  tour.append(c)
  while length_of_tour < N
    maps.remove(c)
    pool={argsort(dist(c,c′)[: 3]|c′ ∈ maps}
    #select next city to connect:
    $c_{next}$ =random.choice(pool,1)
    tour.append($c_{next}$)
    length_of_tour += 1
    $c = c_{next}$
 return tour
```

Figure 2: Half-greedy Tour Selection

must not repeat the deleted edges. See pseudo-code in Figure 3 for details.

```
procedure DPX
 begin
    input $t_{p_1}, t_{p_2} \in P$
    $e_1 := \{edge \in t_{p_1}\}$
    $e_2 := \{edge \in t_{p_2}\}$
    common-edges $:= e_1 \cap e_2$
    destroyed-edges $:= e_1 \cup e_2 - e_1 \cap e_2$
    $t_{child} = \{\}$
    for $e \in$ common-edges:
       $t_{child} = t_{child} + e$
    reconnect $t_{child}$ with greedy approach and
    new edge in $t_{child}$ must not come from
    destroyed-edges
 end
```

Figure 3: DPX Procedure

Mutation serves as random variant to keep population open to fresh exploring, and happens in low frequency (1%). To perform mutation on a tour, two cities in tour are randomly swapped to generate variation in the population.

2.2 2-opt Algorithm

The idea of the 2-opt algorithm[5] is to obtain a better tour through replacing any two edges by other two ones. Suppose there are n cities the cities salesmen should reach sequentially: $\{c_1, ..., c_n\}$. Given the swap position determined as c_f and c_l, the tour is $\{c_1,...c_{f-1},c_f,c_{f+1},...,c_{l-1},c_l,c_{l+1},...,c_n\}$, then edge $(c_f,\ c_{f-1})$ and $(c_l,\ c_{l+1})$ are replaced by (c_l,c_{f-1}) and $(c_f,\ c_{l+1})$. The new tour turns to be $\{c_1,...,c_{f-1},c_l,c_{last-1},...,c_{f+1},c_f,c_{l+1}...,c_n\}$. The 2-opt algorithm is shown in detail in Figure 4.

```
2-opt Algorithm
 begin
 Initialize tour $t$ and best-distance.
 Let count = 0.
 while (the tour is still improving):
    for $c_{first}$ in range(1,(len(tour)-2)):
       for $c_{last}$ in range($c_{first}$+1,len(tour)):
          Del ($c_{first}$, $c_{first-1}$) and ($c_{last}$, $c_{last+1}$)
          Build ($c_{last}$, $c_{first-1}$) and ($c_{first}$, $c_{last+1}$)
          Calculate new-distance
          if new-distance <best-distance:
             Update tour and best-distance
             count += 1
  end
```

Figure 4: 2-opt Algorithm

In this procedure each pair of cities should be checked with a total number of $O(n^2)$ times. Effort can be saved if somehow narrowing down the size of potential cities to be tried instead of trying all probable exchanges. Fortunately, some idea from Reinforcement Learning could help with this narrowing, plus with improvement coming from efficiency in moving trace. It would be discussed in detail in next section.

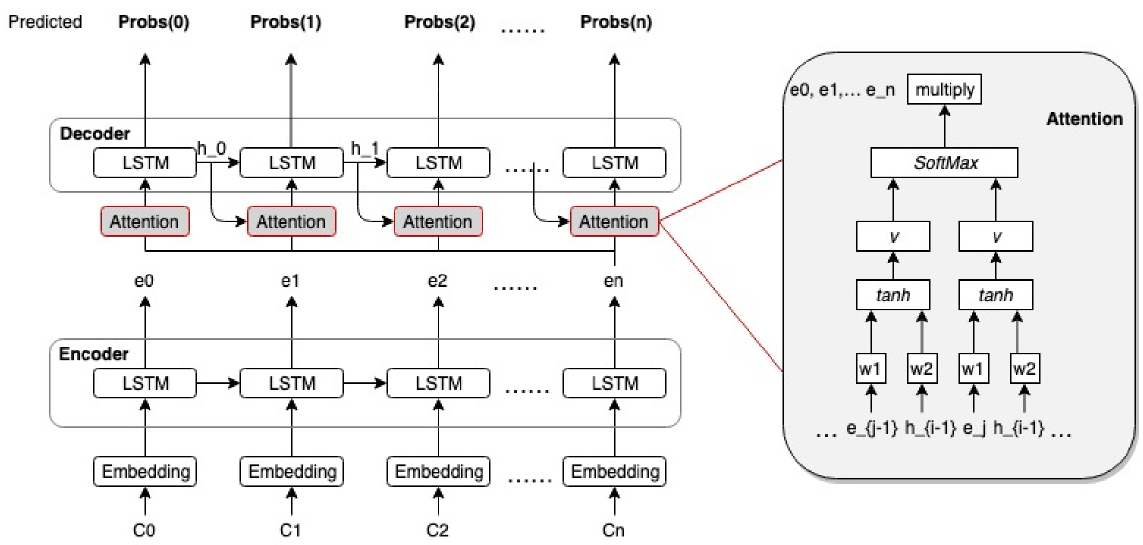

Figure 5: Pointer Network with Bahdanau Attention[15]

3 RL-2opt-GA

In terms of TSP, swap based local search wants to minimize the distance of tour by making greedy improvement each step. In this non-convex situation it may move in a zigzag way with relatively slow speed to reach the local optimal. Instead of greedy strategy, Reinforcement Learning is trained to make a decision in prospect of future reward. In this project a learning result from Reinforcement Learning, namely the transition operator based on probabilities is applied to 2-opt step, as each swap operation in 2-opt process are formed according to those probabilities. Besides narrowing down the pool to be checked out, guide from Reinforcement Learning prefers long term reward and will lead a way reaching local optimal in less steps. Such a combination will provide a more efficient and effective way for tour selection and generation.

3.1 Pointer Network and Reinforcement Learning

As a modification of Recurrent Neural Network[16], Pointer Network[17] uses attention as a pointer to give anoutput by selecting a member from the input sequence. The structure of Pointer Network (PN) utilizing Bahdanau Attention is shown in Figure 5.

The parameters of a Pointer Network can be optimized using the well-known REINFORCE algorithm[18]. See Figure 6 for detail.

Although an independent use of Reinforcement Learning could provide a solution by itself, it requires a large scale of data and time for calculations. A new procedure RL-2opt-GA, as proposed in below section, is a combined algorithm of Reinforcement Learning, 2-opt and Generic Algorithm. In this algorithm PN is half-trained based on relatively small data load to save effort, while it is still able to generate a well-performing optimal tour.

After optimizing the Pointer Network with REINFORCE algorithm, the $n \times n$ matrix of probabilities D will be returned. Notice that in PN the probabilities come out as a sequence corresponding to actions at each step, but the collected $D[i, j]$ just represents the probability of connecting city j as the next when city i has just reached without considering any other edges. In other word, given different starting point, first step of each sequence is collected.

```
REINFORCE Algorithm
 training set = T
 batch size = B
 repeat
   $s_i \sim sample(T)$ for $i \in \{1, \ldots, B\}$
   $\pi_i \sim sample\left(p_\theta\left(\cdot|s_i\right)\right)$for $i \in \{1, \ldots, B\}$
   $g_\theta \leftarrow \frac{1}{B}\sum_{i=1}^{B}\left(L\left(\pi_i|s_i\right) - b_i\right)\nabla_\theta \log p_\theta\left(\pi_i|s_i\right)$
   $\theta \leftarrow ADAM\left(\theta, g_\theta\right)$
   until convergence
 return $p_\theta$
```

Figure 6: REINFORCE Algorithm (Source: [18])

3.2 A Reinforcement Learning Inspired Local Search

As the decision matrix D comes from a half-trained Pointer Network, no doubt that its suggestion is far away from the best. But for a 2opt-GA model, it is enough to make some improvement, as all suggestions are tested and the useful one with good output would be found. This good "character" learned form D would be saved and breed off to the whole population, and eventually shows in optimal tour. In terms of efficiency, 2-opt performs a work with complexity $O(n^2)$ as mentioned previously. With the help of decision matrix D, for each city i, only k cities with largest probabilities in D[i,:] need to be checked, which requires $O(n)$ times. The 2-opt algorithm based on decision matrix is shown in detail in Figure 7.

```
2-opt Algorithm Based on Decision Matrix
 begin
 Initialize tour $t$ and best-distance
 Input decision matrix $D$
 Let count = 0
 while (the tour is still improving):
   for $c_{first}$ in range(1,(len(tour)-2)):
     $D_{c_{first}} = \{d_1, ..., d_K\}$
     $C$ = the corresponding cities
     for $c_{last}$ in $C$:
       Del ($c_{first}$, $c_{first-1}$) and ($c_{last}$, $c_{last+1}$)
       Build ($c_{last}$, $c_{first-1}$) and ($c_{first}$, $c_{last+1}$)
       Calculate new-distance
       if new-distance <best-distance:
         Update tour and best-distance
         count + = 1
   end
```

Figure 7: 2-opt Algorithm with Decision Matrix

4 Datasets and Experiments

Python and PyTorch is used to realize computing work in this project, on the cloud server `www.sciserver.org`. The dataset, instances of 50 cities on a map, are randomly generated by sampling uniform distributed point from the range $[0, 1] * [0, 1]$, and cost between cities is measured by Euclidean distance.

To compare the classical 2opt-GA algorithm and RL-2opt-GA with decision matrix, distances of optimal solution and running time are reported for 30 randomly generated maps. Both algorithm runs genetic evolution with total population of 30, maximum 30 iterations.

5 Evaluation

Among these optimal solutions, distance of the tour will be viewed as main standard to evaluate results. The shortest distance will be treated as standard level, and all other solutions will be divided by this standard level to measure how much worse they are compared with standard level. Average will be another important measurement to evaluate outputs. Instead of arithmetic mean, geometric mean is measured as it is more stable when affected by extreme values. All n optimal solutions will be multiplied together and then take n^{th} root on its product.

Table 1 shows the comparing of classical 2opt-GA and RL-2opt-GA, in terms of running time in arithmetic mean and optimal solutions in geometric mean during iteration. Update of each iteration in terms of scaled distance of tour is shown in Figure 10. The distance of optimal solution from two algorithm is almost the same, and RL-2opt-GA finishes the task in shorter time, which indicates that information from

	Time (sd)	Distance			
		Iteration 0	Iteration 10	Iteration 20	Iteration 30
RL-2opt-GA with D	499.9 (9.195)	5.78444	5.72206	5.71061	5.70458
RL-2opt-GA	579.1 (10.962)	5.78523	5.71853	5.70684	5.70307

Table 1: Result Comparison

```
procedure RL-optimized PN
 #Begin RL
 Initial maps = cities(50)
 Initial tours = train_tours(10000)
 begin forward:
    probs = PN(maps, attention)
    decision = Route(probs, start_city)
    eval = evaluation(decision)
 #Gradient Descent optimization for PN
  PN_opt = RL(eval)
  D = PN_opt(maps)
 return decision matrix D
```

Figure 8: RL-optimized PN

```
 begin GA
 initialize population P
 foreach tour $t \in P$ do $LocalSearch(t)$
    repeat #repeat for generation
        repeat p times: #repeat for population
          randomly select $t_{p1}, t_{p2} \in P$
          $t_c := breed_{DPX}(t_{p1}, t_{p2})$
          $2\text{-}opt(t_c, D)$
          $Mutation(t_c)$
          replace one tour in P by $t_c$
       end
    until converged
 end
```

Figure 9: RL-2opt-PN

decision matrix do provide instructions for long term improvement, and move ahead more directly to a local optimal.

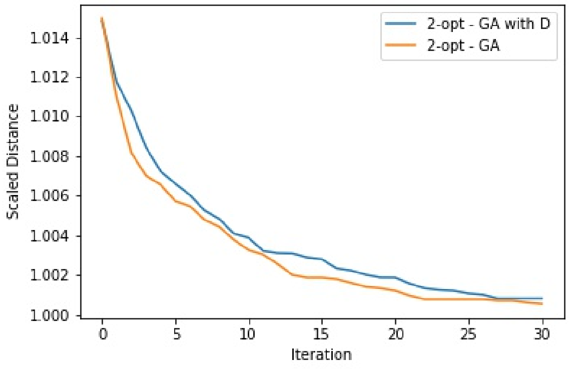

Figure 10: cost of optimal solutions during iteration

6 Summary

The implement of Deep Reinforcement Learning provides information about decision matrix to improve solutions from 2-opt algorithm. Thanks to prospective knowledge contained in decision matrix, the outputs obtained by modified algorithm are better than original 2-opt algorithm.

There are still some improving space for this project. Firstly, no information of the tour before 2-opt is involved in decision matrix currently. If such information could be taken into consideration, local search may be more efficient. Secondly, Deep Reinforcement Learning is trained on 10,000 graphs, which is far away from well-educated. Though quick training is desired for efficiency's sake, more rigorous tuning should be performed to find out the optimal training size. Thirdly, the size of cities is too small to tell a difference between algorithms, even no significant improvement from a simple local optimal to global optimal (solution of a pure local search is only 1.4% worse than solution after global search). Theoretically, with the increasing of city size, the problem becomes more and more complicated and greedy algorithm would be stuck. Due to the limitation of computation resource, experiment on large scale dataset is not performed here.

References

[1] Michael R Gary and David S Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. 1979.

[2] Michel L Balinski and Richard E Quandt. "On an integer program for a delivery problem". In: *Operations Research* 12.2 (1964), pp. 300–304.

[3] Samuel Eilon et al. "Distribution management-mathematical modelling and practical analysis". In: *IEEE Transactions on Systems, Man, and Cybernetics* 6 (1974), pp. 589–589.

[4] Gilbert Laporte, Hélène Mercure, and Yves Nobert. "An exact algorithm for the asymmetrical capacitated vehicle routing problem". In: *Networks* 16.1 (1986), pp. 33–46.

[5] Keld Helsgaun. "An effective implementation of the Lin–Kernighan traveling salesman heuristic". In: *European Journal of Operational Research* 126.1 (2000), pp. 106–130.

[6] Geoff Clarke and John W Wright. "Scheduling of vehicles from a central depot to a number of delivery points". In: *Operations research* 12.4 (1964), pp. 568–581.

[7] Billy E Gillett and Leland R Miller. "A heuristic algorithm for the vehicle-dispatch problem". In: *Operations research* 22.2 (1974), pp. 340–349.

[8] Natalio Krasnogor, Jim Smith, et al. "A Memetic Algorithm With Self-Adaptive Local Search: TSP as a case study." In: *GECCO*. 2000, pp. 987–994.

[9] Licheng Jiao and Lei Wang. "A novel genetic algorithm based on immunity". In: *IEEE Transactions on Systems, Man, and Cybernetics-part A: systems and humans* 30.5 (2000), pp. 552–561.

[10] Noraini Mohd Razali, John Geraghty, et al. "Genetic algorithm performance with different selection strategies in solving TSP". In: *Proceedings of the world congress on engineering*. Vol. 2. 1. International Association of Engineers Hong Kong. 2011, pp. 1–6.

[11] Darrell Whitley. "A genetic algorithm tutorial". In: *Statistics and computing* 4.2 (1994), pp. 65–85.

[12] Peter Merz and Bernd Freisleben. "Genetic local search for the TSP: New results". In: *Proceedings of 1997 Ieee International Conference on Evolutionary Computation (Icec'97)*. IEEE. 1997, pp. 159–164.

[13] Shen Lin and Brian W Kernighan. "An effective heuristic algorithm for the traveling-salesman problem". In: *Operations research* 21.2 (1973), pp. 498–516.

[14] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[15] Dzmitry Bahdanau et al. "End-to-end attention-based large vocabulary speech recognition". In: *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2016, pp. 4945–4949.

[16] Herbert Jaeger. *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the" echo state network" approach*. Vol. 5. GMD-Forschungszentrum Informationstechnik Bonn, 2002.

[17] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. "Pointer networks". In: *Advances in Neural Information Processing Systems*. 2015, pp. 2692–2700.

[18] Irwan Bello et al. "Neural combinatorial optimization with reinforcement learning". In: *arXiv preprint arXiv:1611.09940* (2016).