CS492 Search-Based Software Engineering

- Traveling Sales Problem with Genetic Algorithm -

20163287 박지혁

1. Manual

Please Compile with following command:

```
g++ tsp.cpp -o tsp
```

And then, please excute the solver `tsp` with following arguments:

Population (-p): the number of population. The default value is 50.
Fitness evaluations (-f): the number of fitness evaluations. The default value is 100.
Verbose(-v): You can see the intermediate distances of populations when you turn on this option. Moreover, the distances and the optimal path are automatically saved into “out.db” and “solution.csv” respectively.
Update verbose(-uv): The solver also supports the `update` instruction. It may takes long time for some inputs with largs cities. So, if you switch on the update verbose option, the distances of intermediate tours will be shown.
DB input(-db): You can start with a database file(.db) generated by the verbose option. So, you can stop and continue later with database files.
Filename: Otherwise, the solver considers the inputs as the filename to solve.

2. Genetic Algorithm

1) Generate the initial population:

I define the `upgrade` function that unwinds the pair of line segments of a given tour if these 2 segments cross each other. I have help from the Stack Overflow site to implement the `get_line_intersection` function [1] to check given line segments cross each other.

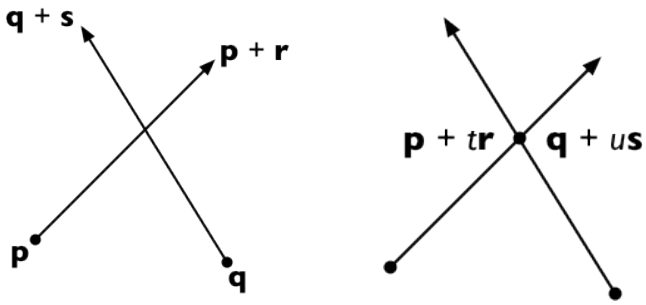

Figure 1. Cross product of 2 vectors for checking whether cross each other.

This algorithm uses the concept of cross product of 2 vectors to know the given point in whether side of the line segment. (**Figure 1**)

Based on this function, the `upgrade` function checks all the pair of line segments for a given tour. And then, it unwinds them by using the following `unwind` function:

```
void unwind(int *tour, int left, int right) {
      int left_before = BEFORE(left);
      int right_before = BEFORE(right);
      int gap = (right - left + size) % size;
      int i;
      for (i = 0; i < gap / 2; i++){
            swap(tour, (left+i)%size,
                  ((right-i-1)+size) % size);
      }
}
```

For the initial population, the solver generates `population` random permutations and applies the above `upgrade` function to unwind all the pairs of line segments for each random tour.

I used the whole distance of a tour as the fitness value of the tour. So, above process makes better fitness values because the whole distance of a reordering tour with unwinding cross line segments is always better then before. And it is much more effective than other hill climbing approach such as the swapping the adjacent cities.

2) Crossover Phase:

The solver has the crossover algorithm for 2 given tours: `crossover`. First, it picks the random initial start index `delta` and the size of the taking part from the father's tour: `cut`. I used the Gaussian distribution for the size `cut` based on the sum of 30 uniform distributions because I want to

give high probability for the half of the number of cities (e.g. E[`cut`] = 140 if # of cities = 280). Based on these values, the new child of them takes the part from father tour from `delta` to `(delta + cut)` and remaining part will be filled based on the order of mother tour from index 0.

3) Mutate Phase:

The solver also do mutate to generated children from the previous population. The `mutate` function picks the random initial start index `delta` and the range of mutation `cut` similar with crossover phase. After that, it reorders the part from `delta` to `(delta + cut)` by using `2 * cut` swaps between 2 random picks from the range. This process will be executed with 30% probability. So, 70% of children may be not mutated.

4) Generate Next Population:

The solver generates next population by using above helper functions. First, the `next_generation` function creates `(4 * population - 10)` children with random father and mother tours. It picks the parents tours based on the roulette mechanism with probability 10%. So, the better one is more picked almost exponentially and if no one is selected, it takes random one with uniform distribution. After that, it mutates the newly created child and unwinds pairs of it with the `upgrade` function. Finally, it leaves the 10 best tours in the previous population and the remaining population will be filled best ones from newly created children and those tours will be sorted based on fitness values.

3. Evaluation

I applied the solver to various TSP problems. **Table 1** shows the evaluation result of them.

TSP name	berlin52	a280	pcb442	vm1084	rl11849
# of cities	52	280	442	1,084	11,849
optimal	7542	2,579	50,778	239,297	923,288
population	50	50	50	50	20
fitness eval.	30	5,000	1,000	500	15
my solution	7,542	2,801	56,649	340,364	1,822,621
ratio	1.000	1.086	1.116	1.422	1.974

Table1 The result of evaluations for each TSP problems from TSPLIB [2]

〈berlin52〉
It becomes best solution very fast because he number of cities is very small. I use the randomness, so, it is not always converged to optimal solution. However, after 30 fitness evaluation, almost evaluations become the best.

〈a280〉
I use same population but much more fitness evaluation for this problem than the berlin52 because it is not converges to optimal solution. Nevertheless, my solution after 5000 fitness evaluation is quite similar with optimal solution (1.086 times longer than optimal solution).

〈pcb442 / vm1084〉
In these experiment, the time spending for the replacement of generations become much slower than before. So, I should stop the evaluation of them after 1,000 and 500 steps for each.

〈rl11849〉
It is quite bigger than other problems. So, I should decrease the number of population, but also stop in 50 steps. Finally, the ratio is 1.974.

4. Conclusion

I implement the solver for TSP problem based on genetic algorithm. I applied the various techniques to the solver. It is quite good solver to smaller size of TSP problems. However, after the 400 cities, it becomes worse, but it gives the solution having at most 2 times longer distance than optimal solution up to almost 10,000 cities.

A. Appendix

[1] http://stackoverflow.com/questions/563198/how-do-you-detect-where-two-line-segments-intersect

[2] http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

B. Additional Work with Heuristic Idea

I tried to change the initial population based on heuristic idea. First, I choose a random initial index and find the closest city from there and pick next city in the same way. It is very efficient and gives much better solution for the problem rl11849: 1,042,855. However, it is fail to apply the above genetic algorithm to this initial population because it's never going better after `crossover` and `upgrade` processes.