

An improved path-scanning for CARP

1. Preliminaries

The capacitated arc routing problem (CARP) is a challenging combinatorial optimization problem with many real-world applications, e.g., salting route optimization and fleet management. Given a graph with some edges and arcs required to be served (called tasks) and a number of vehicles with limited capacity, a CARP is defined as seeking an optimal routing plan for the vehicles under the following conditions.[1]

- Each vehicle starts and ends at a predened vertex, namely depot.
- Each task is served by exactly one vehicle.
- The total demand of the tasks served by each vehicle does not exceed its capacity.

1.1. Software

This project is written in Python using IDE PyCharm. The libraries being used includes NumPy, Copy, random, sys, geopt, os, re

1.2. Algorithm

The algorithm being used in this project includes Dijkstra and path-scanning.

2. Methodology

Path-scanning is similar to greedy algorithm. First, put all the arcs in a set F , pick up an arc which is near the node at the end of the route every time. The new heuristic differs from the Belenguer et al. one in that it utilizes an ellipse rule when the vehicle load is near capacity. Intuition suggests that as a vehicle's load approaches its capacity, its route should stay closer to the depot to reduce its cost to return to the depot when full.[2]

2.1. Representation

This project contains some main data need to be maintain during the process:

- **information**:all the information of the data, including the capacity, vehicles' amount.
- **cost** :adjacent matrix to store the cost information of two joined nodes
- **demand**:adjacent matrix to store the demand information of two joined nodes

- **nodes_amount**:nodes amount
- **initial_route** :dictionary to store the routes
- **initial_load** :dictionary to store the load every time
- **all_distance**:dictionary to store the minimum cost of two joined nodes
- **total_demand**:total demand of every arcs
- **total_cost**:total cost of every arcs
- **all_cost**:total cost of the routes

2.2. Architecture

Here list all functions in the Python file CARP_solver.py with their usage.

- **open_file**: read the .dat file.
- **dijkstra**:dijkstra algorithm, return the minimum distance between arbitrary points on the map.
- **path_scanning**:main algorithm in this project, return detail routes.
- **command_line_reader**: read command line from cmd.

The last of the Python file, use *main()* to test all the codes. I also write a simple crossover function to improve the performance of the result.However it doesn't work.

- Duplicate 20 copies of parent-route set as route
- Random find a segment seg1 in the route[i][k1], and random decide whether to switch the direction
- Random find a segment seg2 in the route[i][k2], and random decide whether to switch the direction
- Insert seg1 into route[i][k2] and insert seg2 into route[i][k1]
- Calculate the minimum cost of these 20 routes, set the route as the next parent-route
- Run this program for 1000 times

2.3. Detail of Algorithm

Algorithm 1 path-scanning

Input: *information, nodes, cost, demand*

Output: the routes of the problem

```

1:  $k \leftarrow 0$ 
2: copy all required arcs which demand are larger than 0
   in a list free
3: while  $free \neq \emptyset$  do
4:    $k \leftarrow k + 1, R_k \leftarrow \emptyset, load(k), cost(k) \leftarrow 0, i \leftarrow 1$ 
5:   while  $free \neq \emptyset$  do
6:     for each  $u \in free$  do
7:       if  $rvc > N \times td/ned$  then
8:         if  $d_{i,beg(u)} < \bar{d}$  and  $load < capacity$  then
9:            $\bar{d} \leftarrow d_{i,beg(u)}$ 
10:           $\bar{u} \leftarrow u$ 
11:         else if  $d_{i,beg(u)} = \bar{d}$  and  $better(u, \bar{d})$  then
12:            $\bar{u} \leftarrow u$ 
13:         end if
14:         else if  $SP(v_i, v_p) + C_{pj} + SP(v_j, v_0) \leq tc \div$ 
15:            $ned + SP(v_j, v_0)$  then
16:           if  $d_{i,beg(u)} < \bar{d}$  and  $load < capacity$  then
17:              $\bar{d} \leftarrow d_{i,beg(u)}$ 
18:              $\bar{u} \leftarrow u$ 
19:           else if  $d_{i,beg(u)} = \bar{d}$  and  $better(u, \bar{d})$  then
20:              $\bar{u} \leftarrow u$ 
21:           end if
22:         end if
23:         end for
24:         add  $\bar{u}$  at the end of route  $R_k$ 
25:         remove arc  $\bar{u}$  from free
26:          $load(k) \leftarrow load(k) + q_{\bar{u}}$ 
27:          $cost(k) \leftarrow cost(k) + \bar{d} + c_{\bar{u}}$ 
28:          $i \leftarrow end_{\bar{u}}$ 
29:       end while
30:   end while
31: return points status  $\leftarrow is\_not\_alive$ 

```

3. title

4. Empirical Verification

Empirical verification is compared with the given ans in the dat file:

. egl-e1-A

```

167 except (IOError as err):
168     print (str(err))
169     print "The argument should be every instance file -t termination -s random seed"
170     sys.exit(1)
171
172 #
173 #
174 #
175 #
176 #
177 #
178 #
179 #
180 #
181 #
182 #
183 #
184 #
185 #
186 #
187 #
188 #
189 #
190 #
191 #
192 #
193 #
194 #
195 #
196 #
197 #
198 #
199 #
200 #
201 #
202 #
203 #
204 #
205 #
206 #
207 #
208 #
209 #
210 #
211 #
212 #
213 #
214 #
215 #
216 #
217 #
218 #
219 #
220 #
221 #
222 #
223 #
224 #
225 #
226 #
227 #
228 #
229 #
230 #
231 #
232 #
233 #
234 #
235 #
236 #
237 #
238 #
239 #
240 #
241 #
242 #
243 #
244 #
245 #
246 #
247 #
248 #
249 #
250 #
251 #
252 #
253 #
254 #
255 #
256 #
257 #
258 #
259 #
260 #
261 #
262 #
263 #
264 #
265 #
266 #
267 #
268 #
269 #
270 #
271 #
272 #
273 #
274 #
275 #
276 #
277 #
278 #
279 #
280 #
281 #
282 #
283 #
284 #
285 #
286 #
287 #
288 #
289 #
290 #
291 #
292 #
293 #
294 #
295 #
296 #
297 #
298 #
299 #
300 #
301 #
302 #
303 #
304 #
305 #
306 #
307 #
308 #
309 #
310 #
311 #
312 #
313 #
314 #
315 #
316 #
317 #
318 #
319 #
320 #
321 #
322 #
323 #
324 #
325 #
326 #
327 #
328 #
329 #
330 #
331 #
332 #
333 #
334 #
335 #
336 #
337 #
338 #
339 #
340 #
341 #
342 #
343 #
344 #
345 #
346 #
347 #
348 #
349 #
350 #
351 #
352 #
353 #
354 #
355 #
356 #
357 #
358 #
359 #
360 #
361 #
362 #
363 #
364 #
365 #
366 #
367 #
368 #
369 #
370 #
371 #
372 #
373 #
374 #
375 #
376 #
377 #
378 #
379 #
380 #
381 #
382 #
383 #
384 #
385 #
386 #
387 #
388 #
389 #
390 #
391 #
392 #
393 #
394 #
395 #
396 #
397 #
398 #
399 #
400 #
401 #
402 #
403 #
404 #
405 #
406 #
407 #
408 #
409 #
410 #
411 #
412 #
413 #
414 #
415 #
416 #
417 #
418 #
419 #
420 #
421 #
422 #
423 #
424 #
425 #
426 #
427 #
428 #
429 #
430 #
431 #
432 #
433 #
434 #
435 #
436 #
437 #
438 #
439 #
440 #
441 #
442 #
443 #
444 #
445 #
446 #
447 #
448 #
449 #
450 #
451 #
452 #
453 #
454 #
455 #
456 #
457 #
458 #
459 #
460 #
461 #
462 #
463 #
464 #
465 #
466 #
467 #
468 #
469 #
470 #
471 #
472 #
473 #
474 #
475 #
476 #
477 #
478 #
479 #
480 #
481 #
482 #
483 #
484 #
485 #
486 #
487 #
488 #
489 #
490 #
491 #
492 #
493 #
494 #
495 #
496 #
497 #
498 #
499 #
500 #
501 #
502 #
503 #
504 #
505 #
506 #
507 #
508 #
509 #
510 #
511 #
512 #
513 #
514 #
515 #
516 #
517 #
518 #
519 #
520 #
521 #
522 #
523 #
524 #
525 #
526 #
527 #
528 #
529 #
530 #
531 #
532 #
533 #
534 #
535 #
536 #
537 #
538 #
539 #
540 #
541 #
542 #
543 #
544 #
545 #
546 #
547 #
548 #
549 #
550 #
551 #
552 #
553 #
554 #
555 #
556 #
557 #
558 #
559 #
560 #
561 #
562 #
563 #
564 #
565 #
566 #
567 #
568 #
569 #
570 #
571 #
572 #
573 #
574 #
575 #
576 #
577 #
578 #
579 #
580 #
581 #
582 #
583 #
584 #
585 #
586 #
587 #
588 #
589 #
590 #
591 #
592 #
593 #
594 #
595 #
596 #
597 #
598 #
599 #
600 #
601 #
602 #
603 #
604 #
605 #
606 #
607 #
608 #
609 #
610 #
611 #
612 #
613 #
614 #
615 #
616 #
617 #
618 #
619 #
620 #
621 #
622 #
623 #
624 #
625 #
626 #
627 #
628 #
629 #
630 #
631 #
632 #
633 #
634 #
635 #
636 #
637 #
638 #
639 #
640 #
641 #
642 #
643 #
644 #
645 #
646 #
647 #
648 #
649 #
650 #
651 #
652 #
653 #
654 #
655 #
656 #
657 #
658 #
659 #
660 #
661 #
662 #
663 #
664 #
665 #
666 #
667 #
668 #
669 #
670 #
671 #
672 #
673 #
674 #
675 #
676 #
677 #
678 #
679 #
680 #
681 #
682 #
683 #
684 #
685 #
686 #
687 #
688 #
689 #
690 #
691 #
692 #
693 #
694 #
695 #
696 #
697 #
698 #
699 #
700 #
701 #
702 #
703 #
704 #
705 #
706 #
707 #
708 #
709 #
710 #
711 #
712 #
713 #
714 #
715 #
716 #
717 #
718 #
719 #
720 #
721 #
722 #
723 #
724 #
725 #
726 #
727 #
728 #
729 #
730 #
731 #
732 #
733 #
734 #
735 #
736 #
737 #
738 #
739 #
740 #
741 #
742 #
743 #
744 #
745 #
746 #
747 #
748 #
749 #
750 #
751 #
752 #
753 #
754 #
755 #
756 #
757 #
758 #
759 #
760 #
761 #
762 #
763 #
764 #
765 #
766 #
767 #
768 #
769 #
770 #
771 #
772 #
773 #
774 #
775 #
776 #
777 #
778 #
779 #
780 #
781 #
782 #
783 #
784 #
785 #
786 #
787 #
788 #
789 #
790 #
791 #
792 #
793 #
794 #
795 #
796 #
797 #
798 #
799 #
800 #
801 #
802 #
803 #
804 #
805 #
806 #
807 #
808 #
809 #
810 #
811 #
812 #
813 #
814 #
815 #
816 #
817 #
818 #
819 #
820 #
821 #
822 #
823 #
824 #
825 #
826 #
827 #
828 #
829 #
830 #
831 #
832 #
833 #
834 #
835 #
836 #
837 #
838 #
839 #
840 #
841 #
842 #
843 #
844 #
845 #
846 #
847 #
848 #
849 #
850 #
851 #
852 #
853 #
854 #
855 #
856 #
857 #
858 #
859 #
860 #
861 #
862 #
863 #
864 #
865 #
866 #
867 #
868 #
869 #
870 #
871 #
872 #
873 #
874 #
875 #
876 #
877 #
878 #
879 #
880 #
881 #
882 #
883 #
884 #
885 #
886 #
887 #
888 #
889 #
890 #
891 #
892 #
893 #
894 #
895 #
896 #
897 #
898 #
899 #
900 #
901 #
902 #
903 #
904 #
905 #
906 #
907 #
908 #
909 #
910 #
911 #
912 #
913 #
914 #
915 #
916 #
917 #
918 #
919 #
920 #
921 #
922 #
923 #
924 #
925 #
926 #
927 #
928 #
929 #
930 #
931 #
932 #
933 #
934 #
935 #
936 #
937 #
938 #
939 #
940 #
941 #
942 #
943 #
944 #
945 #
946 #
947 #
948 #
949 #
950 #
951 #
952 #
953 #
954 #
955 #
956 #
957 #
958 #
959 #
960 #
961 #
962 #
963 #
964 #
965 #
966 #
967 #
968 #
969 #
970 #
971 #
972 #
973 #
974 #
975 #
976 #
977 #
978 #
9
```

. gdb1

[illegible]

. gdb10

```

292
293 # ufa.dat
294 # ufa.dat
295 if __name__ == "__main__":
296     # file name termination, seed = command_line_reader(sys.argv[1])
297     # gpus = f'["C:/Users/THINPAD/PycharmProjects/CARP/data/gpus.dat"]'
298     # randoms = seed
299     cost1, route1 = path_scanning()
300     for r in range(0, len(route1)):
301         print route1[r]
302     print cost1
303     result = "a = 0"
304     for r in range(0, len(route1)):
305         result = result + "a="
306
307
308 # user
309 c:\Users\THINPAD\AppData\python.exe C:/Users/THINPAD/PycharmProjects/CARP/CARP_solver.py
310 [0, 51, 0, 21, 11, 2, 2, 3, 0, 8, 6, 10, 9]
311 [0, 1, 11, 1, 6], (6, 5), (5, 7), (7, 2), (2, 4), 0]
312 [0, 1, 11, 1, 6], (6, 5), (5, 7), (7, 2), (2, 4), 0]
313 [0, 1, 16, 10, 9], (4, 11), (11, 12), (12, 10), 0]
314 [0, 1, 4, 7, 0]
315 a=a=a
316 = 0, (1, 5), (5, 2), (1, 2), (2, 3), (3, 6), (6, 12), 0, 0, (1, 11), (11, 6), (6, 5), (5, 7), (7, 2), (2, 4), 0, 0, (4, 3), (3, 9), (9, 8), (1, 8), 0, 0, (1, 10), (10, 9), (4, 7), 0]
317
318 Process finished with exit code 0

```

. val1A

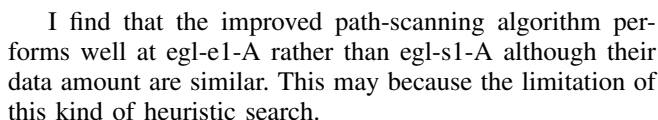
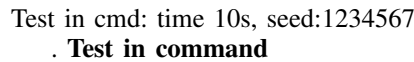
[illegible]

. val4A

[illegible]

. val7A

[illegible]



I would like to thank Ni, Li, Zheng for exchanged some thoughts with me on some vital algorithms although I did not successfully finished this project. But I have gained a lot through paper reading and algorithm implementations. Maybe I have to manage my time in rest projects. It is very regrettable of not having enough time to finish my optimize part.

- [1] Y. Mei, K. Tang and X. Yao, "Decomposition-Based Memetic Algorithm for Multiobjective Capacitated Arc Routing Problem", *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 2, pp. 151-165, 2011.
- [2] L. Santos, J. Coutinho-Rodrigues and J. Current, "An improved heuristic for the capacitated arc routing problem", *Computers & Operations Research*, vol. 36, no. 9, pp. 2632-2637, 2009.
- [3] "Dijkstra's algorithm", *En.wikipedia.org*, 2017. [Online]. Available: <https://en.wikipedia.org/wiki/Dijkstra>