Linköping Institute of Technology
Department of Mathematics/Division of Optimization
Kaj Holmberg

2018–09–10

Lab Information

1 Information about SNOWPLAN

SNOWPLAN version 3 is code in python for solving a simplified snow removal problem. The problem to be solved is called a "k-postman problem", because it is a postman problem where all arcs are covered (at least once), but not by *one* postman but several. A "postman" corrresponds to a snow removal vehicle in this context.

SNOWPLAN works with a subproblem, which is a "rural postman problem" for each vehicle. These problems are solved using a code from Vineopt, *rupov*. The network is read in Vineopt format, and the main task of SNOWPLAN is to find a good division of arcs between different vehicles, i.e. an allocation of arcs to vehicles.

Two important vectors used by the code are `order`, which is the order in which the vehicles are considered, and `vehdo`, which states for each arc which vehicles should clean it.

The main program SNOWPLAN contains the following parts.

1. The network is read.
2. A starting order is constructed.
3. A starting allocation of arcs to vehicles is done.
4. The rural postman problem is solved for each vehicle.
5. Optional: Those arcs passed by a vehicle are considered to be cleaned by this vehicle, if they were not cleaned before, even if they were allocated to another vehicle.
6. Optional: The solution is improved by moving cycles from one tour to another one.
7. The order for vehicles is changed.
8. The allocation of arcs to vehicles is changed.
9. Steps 4 - 9 are repeated several times.
10. The allocation of vehicles to arcs is written on a file (as arc sets) in a format that Vineopt can read.

The total cost, which is the sum of the costs for each vehicle's tour plus the fixed costs for the vehicles, is computed at each iteration. Moreover the time for solution is computed, which is the maximal of the vehicles' times. The objective function value is then computed as a weighted sum of the cost and the time.

Simulated annealing is used to accept a move to a new point (for `vehdo`), i.e. better points are always accepted and worse points are accepted with a certain probability.

Steps 3,7 and 8 can be done in various ways, so one has to choose among the alternative. One also has to choose which of the optional steps 5 and 6 to do. These choices are governed by parameters set by the user.

Note that the method uses the *randomness*, so two runs with identical settings will not give exactly the same results.

1.1 Starting Snowplan

The program is started by opening a terminal window, setting the search paths by writing `module add ext/MAI` and `module add math/optlab` and then write `dine 6`.

You then get the interactive window with possibilities to read a network and parameters, and change parameters and solve the problem. One can also give names to network file and parameter file on the command line:

```
dine 6 network1 param1.par
```

reads the network *network1* and the parameter file *param1.par*.

```
dine 6 network1 param1.par 1
```

directly solves the problem. You will then see how the objective function iteratively changes. A blue line gives the original objective function value in each iteration, while green crosses gives values after the improvement phase, and the red dashed curve gives the best value at each iteration. A light blue line gives the lower limit.

```
dine 6 network1 param1.par 2
```

will solve the problem and then exit Snowplan. To see the results, you must study the output on the terminal.

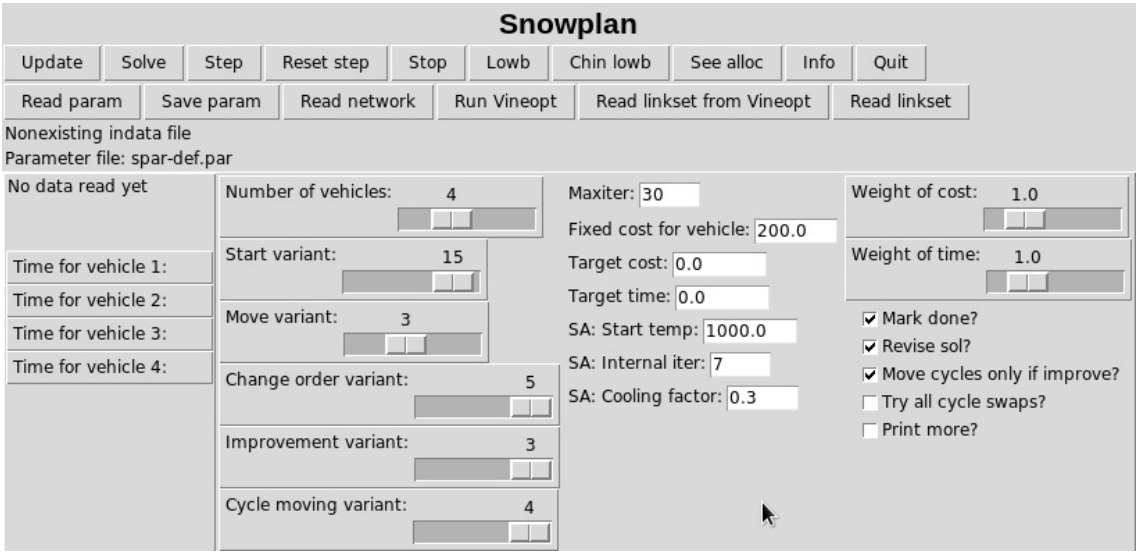
Every time a problem is solved, a line is added to the file *fkres.txt*, with the network name, parameter file, date, size of problem and the objective function value and solution time. This file is never deleted, so it is recommended to rename it when you have made all the runs that you want to compare. A similar file, *fpkres.txt*, created in the same way, but includes (compact) information about all parameters.

Also created are multiple result files in the data directory (# stands for a number that increased each time):
network1-param1_#.png: A picture of the best allocation in png-format.
network1-param1_#.ps: A picture of the best allocation in postscript-format.
network1-param1_#.val: Lower bounds, and for each iteration the objective function value before and after improvement, as well as the best value so far.
(Please check and remove these files sometimes.)

1.2 Parameters

The program uses several parameters that control the solution method, and these can be loaded from a file. An example of the proposed values are given in the file *spar-def.par*. If one finds better values of the parameters, saved them on another file (with the extension *.par*).

You can also set the parameters in the program's interface; see the picture below.



When you have changed any parameter, you should click on the button "Update" to the settings to be saved. If you forgot what the different values stand for, you can click on the button "Info". You also get informative output in the terminal when changing any parameter.

In order to calculate a lower bound, click on "Lowb" or "Chin lowb".

When you click on "Solve" the problem is solved. If you want to do one iteration at a time, you should instead click on "Step".

If you click on "Run Vineopt", Vineopt is started, and the current allocation (linksets) is loaded and displayed. You can then change the allocation, and save it via the menu "Sets", "Save Link sets for Snowplan". Then exit Vineopt, and click "Read linkset from Vineopt" in Snowplan, to make Snowplan read the allocation made in Vineopt. One can then proceed from that allocation (if startvar = 0). One can also set startvar = 2, which automatically loads the allocation from Vineopt when clicking "Solve".

The button "See alloc" shows the allocation in a colored graph. (This feature uses an external program, and may fail if program is not installed.)

1.3 Program function

SNOWPLAN first finds a tour for the first vehicle which covers all arcs which are allocated for vehicle 1. If the parameter *makedone*=1, all arcs passed by vehicle 1 are marked as

done, i.e. the allocation for these arcs is not folloed. This means that the following vehicles have less work to do. If one changes the order in which tours for vehicles are decided, a different solution could be generated.

Therefore the vector `order` can be changed to get different solutions. There are several different approaches to (randomly) change `order`.

In the same way, there are several approaches to change the vector `vehdo`, in order to change the allocation of arcs to vehicles.

The following parameters can be given the following values in the parameter file. If you do not give any value, the parameters get default values, given below in brackets.

Parameter	Description
nov	Number of vehicles, ≥ 1 (4)
fixvehcost	Fixed cost for a vehicle, ≥ 0 (200)
wcost	Weight of cost in the objective function, ≥ 0 (1.0)
wtime	Weight of time in the objective function, ≥ 0 (1.0)
maxiter	Number of main iterations, ≥ 1 (30)
logfil	Controls output from rupov (to the screen or to the file), 0,1 (1)
pri	Amount of output (0 less, 1 more), 0,1 (0)
markdone	Mark the passed arcs as completed 0,1 (1)
startvar	Variant for starting allocation, $0 - 16$, (15)
movevar	Variant for changing allocation, $0 - 7$ (3)
moalloc	Number of steps for `vehdo` to be moved, ≥ 0 (1)
impvar	Variant for the improvement heuristic, $0 - 3$ (3)
dorevsol	Change the allocation to follow the solution, 0,1 (1)
chordvar	Variant for changing order, $1 - 5$ (5)
moord	Number of steps for `order` to be moved, ≥ 0 (1)
cymvar	Variant for moving cycles, $1 - 4$ (4)
cymimp	Move cycles only if this provides improvement, 0,1 (0)
targetcost	Terminate when the cost is below this, ≥ 0 (0.0)
targettime	Terminate when the time is below this, ≥ 0 (0.0)
starttemp	Starting temperature in simulated annealing, ≥ 0 (1000)
intiter	Number of iterations with the same temperature in simulated annealing, ≥ 1 (7)
rcold	Cooling factor in simulated annealing, > 0, < 1 (0.3)

The file *spar-def.par* contains the default values for each of these parameters. Some parameters influence the methods behavior a lot, others less. It is difficult to find values which are the best for all problems. One can try change some of the values to make the method work good for some problems. Some values can generate good solutions, but cause the method to run for a long time for large problems. (Observe that very small problems may have too few degrees of freedom in order for the calibration of parameters to give a good result.) For large/difficult problems, it is tempting to run only a few iterations, but this kind of method will not give a good solution in that case. Difficult problems will always take a long time to solve, so one needs to be patient.

After each run the linksets are saved on the file *linksets#.set*, where # stands for a number that increased every time a file is saved.

1.3.1 startalloc

This function creates an allocation for vehicles, i.e. the vector *vehdo*, where *vehdo(j)* = the vehicle to clean arc j.

It is controlled by the parameter *startvar*:
startvar = 0: Start from the current solution.
startvar = 1: Create a random allocation.
startvar = 2: Read a linkset that has been saved in Vineopt and use as a starting allocation.
startvar = 3: Let vehicle 1 to do everything.
startvar = 4: Sort by x for a starting node, begin with the first vehicle, select the nearest arcs, then proceed to the next vehicle.
startvar = 5: Like in 4, but sort by x for an ending node.
startvar = 6: Like in 4, but sort by x for a middle point.
startvar = 7: Like in 4, but sort by y for a starting node.
startvar = 8: Like in 4, but sort by y for an ending node.
startvar = 9: Like in 4, but sort by y for a middle point.
startvar = 10: Like in 4, but sort by the distance from the origin to a starting node.
startvar = 11: Like in 10, but sort by the distance from the origin to an ending node.
startvar = 12: Like in 10, but sort by the distance from the origin to a middle point.
startvar = 13: Apply the code *kmeans* which creates a number of clusters (the same as the number of vehicles), where the closest nodes to every cluster's middle point are put into the cluster. Arcs with both the nodes in one cluster are allocated to the corresponding vehicle. If the nodes belong to different clusters, the arc is allocated to the cluster with the highest number.
startvar = 14: Like in 13, but if the nodes belong to different clusters, the arc is allocated to the cluster with the lowest number.
startvar = 15: Like in 13, but if the nodes belong to different clusters, the arc is allocated randomly to one of this clusters.
startvar = 16: Like in 13, but if the nodes belong to different clusters, the arc is allocated so that the distance to the center of the cluster is minimized.

If one uses *startvar* = 0, one can make multiple runs one after another to continue to improve the solution. Running three times with 30 iterations gives almost the same effect as one run with 90 iterations. The difference is that the temperature in simulated annealing is reset after each run.

1.3.2 changealloc

This function changes the allocation of arcs to vehicles (the vector *vehdo*).

It is controlled by the parameters *movevar* and *moalloc*:
movevar = 0: Do not change the allocation.
movevar = 1: Swap two random links.
movevar = 2: Change *moalloc* random links.
movevar = 3: Move a link from a vehicle with the largest number of links to a vehicle with the lowest number of links.
movevar = 4: Move an arc between two random vehicles that have both nodes in common.
movevar = 5: Swap between two random vehicles that have both nodes in common.

movevar = 6: Move an arc, when one node is in common.
movevar = 7: Swap an arc, when one node is in common.

1.3.3 changeorder

This function changes the order. It is controlled by parameters *chordvar* and *moord*:
chordvar = 1: Swap two random elements in the vector `order`.
chordvar = 2: Create a completely new order.
chordvar = 3: Shift the vector `order` by *moord* steps to the right.
chordvar = 4: Shift the vector `order` by *moord* steps to the left.
chordvar = 5: Swap the one has the most with the one that has the least.

1.3.4 improvesol

This function improves the solution as follows. Find a cycle in a tour, check if the cycle contains a node in another tour, move the cycle to that tour.

It is controlled by the parameter *impvar*:
impvar = 0: Do not move cycles.
impvar = 1: Move cycles, one random try.
impvar = 2: Move cycles, terminate when there is no improvement.
impvar = 3: Move cycles, try all combinations.

1.3.5 cyclemove

Executed if *impvar* > 0. Invoked from *improvesol*. Performs the movement of the cycle.

It is controlled by the parameter *cymvar*:
cymvar = 0: Do nothing.
cymvar = 1: Try to move a random cycle between two random vehicles.
cymvar = 2: Try to move a cycle from a long tour to a short one.
cymvar = 3: Try to move a cycle between two given vehicles (if *impvar* = 3).
cymvar = 3: Try to move a cycle from a costly tour to a cheap one.

1.4 Contact

If the program should not work as it should in any way, the user is asked to send information so that the error can be recreated (and corrected) to kaj.holmberg@liu.se. Please send the parameter file with the settings that makes the program fail, and details of the problem that has been run.