A Powerful Route Minimization Heuristic for the Vehicle Routing Problem with Time Windows

Yuichi Nagata

Japan Advanced Institute of Science and Technology

Olli Bräysy

University of Jyväskylä, Finland

Introduction

- Vehicle routing problem with time window (VRPTW) is one of the most important and studied VRP variant.
  - primary objective: minimize the number of vehicles
  - secondary objective: minimize the total travel distance
- Recent trend of heuristic algorithms for the VRPTW is the two-stage approach (Bent and Hentenryck, 04).
- We propose an efficient route-minimization heuristic for the VRPTW.

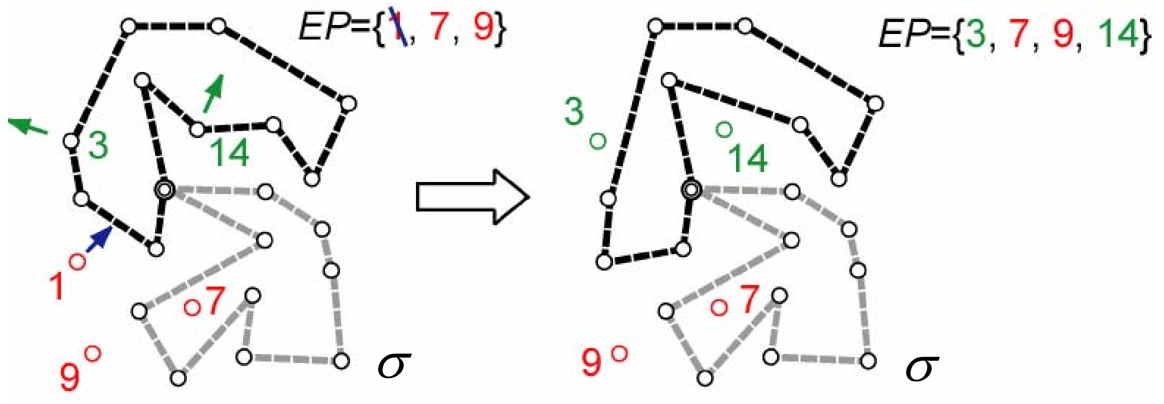
Outline of this talk

- General framework using ejection pool
- Our solution method
- Experimental results
- Conclusions

General framework of the *Ejection Pool* (1)

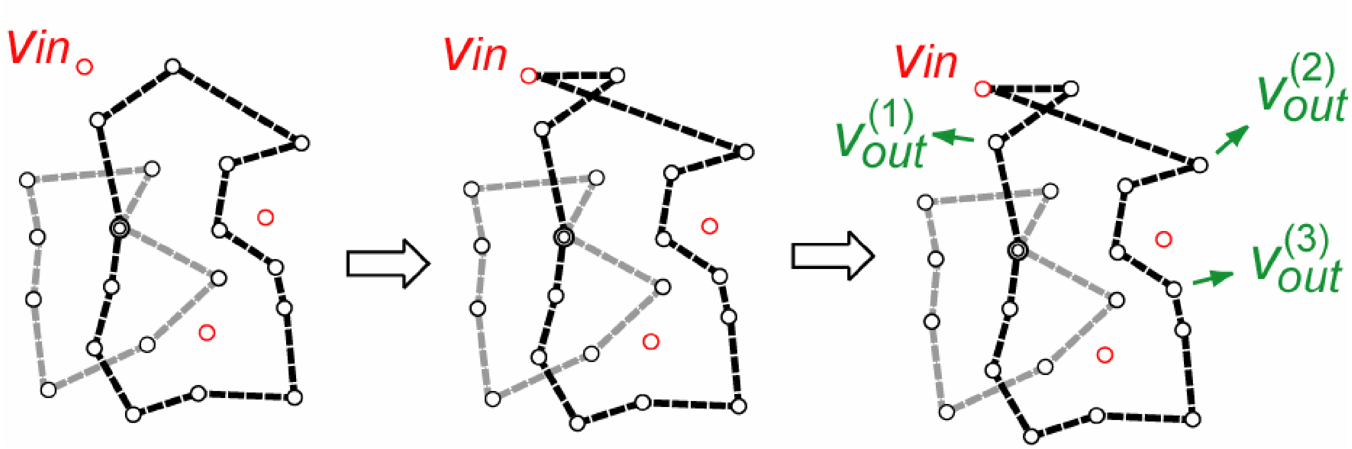
Route-minimization procedure using the EP (Lim and Zhang, 07)

```
Remove a route from σ and initialize EP with the removed
  customers;
while EP is not empty do
      Select v_in from EP and remove it from EP
      if v_in can be inserted into σ then
          Insert v_in into σ ;
      else
          Insert v_in into σ  and eject customers from the resulting route;
          Add ejected customers to EP ;
      end if
end while
```



General framework of the *Ejection Pool* (2)

- Insert-ejection move
  - For customer v_{in} to be inserted, all edges can be insertion positions.
  - For each insertion of v_{in}, there are a lot of customer combinations, $v_{out}^{(1)}, \ldots, v_{out}^{(k)}$, to be ejected



- Which insertion-ejection move is better?
  - The number of ejecting customers (k) should be small.
  - Ejecting more than two customers may benefit the subsequent insertion.

Our idea

- A concept of the guided local search (GLS) is employed to determine the insertion-ejection move.
- Guided local search (GLS) (Voudouris and Tsang, 95)
  - Penalizing solution features that are frequently appeared in local minima during the local search.
  - A modified objective function including the penalties are used to help the local search escape from local minima and to diverse the search.
- In our solution method, customers in the EP are solution features.

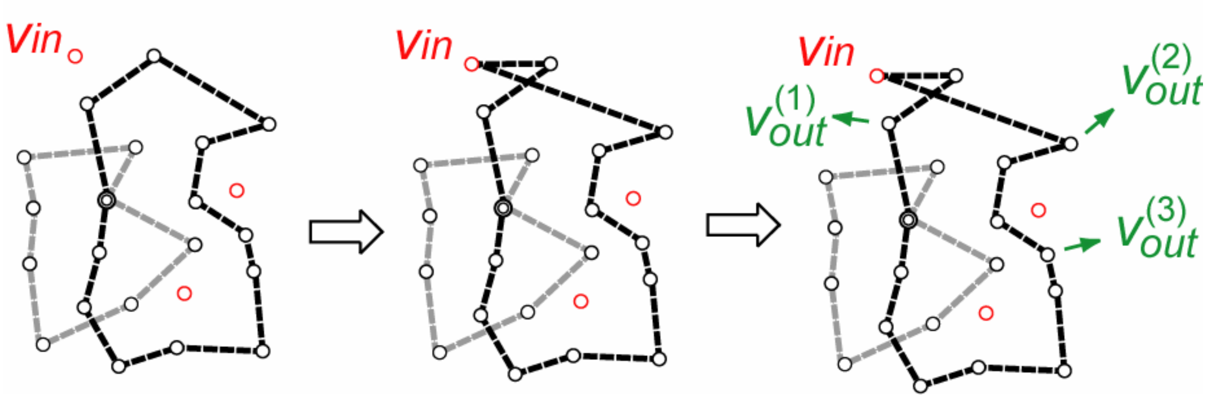
Main framework

Procedure Delete_Route (σ)

Randomly remove a route from σ and initialize EP ;
Initialize all penalty counters: $p[v] = 1$ (v =1, …, N) ;
while EP is not empty do
    Select and remove v_{in} from EP (LIFO queue)
    if v_{in} can be inserted into σ then
        Insert v_{in} into σ ;
    else
        Set: $p[v_{in}] = p[v_{in}] + 1$;
        Execute the insertion-ejection move on σ such that $P_{sum} = p[v_{out}^{(1)}] +, \ldots, + p[v_{out}^{(k)}]$ is minimized ;
        Add $v_{out}^{(1)}, \ldots, v_{out}^{(k)}$ to EP ;
        $\sigma :=$ Perturb (σ) ;
    end if
end while

Finding the best insertion-ejection move

- How to find the insertion-ejection move that minimizes $P_{sum} = p[v_{out}^{(1)}] +, \dots, + p[v_{out}^{(k)}]$.
- There are enormous numbers of insertion-ejection moves.
  - v_{in} is given.
  - All insertion positions for v_{in} are tested.
  - For each insertion, there are a lot of customer combinations to be ejected. (k is limited up to k_{max} (=5).)



- For each insertion, most of the ejection combinations can be ignored (the detail is omitted).

Main framework (again)

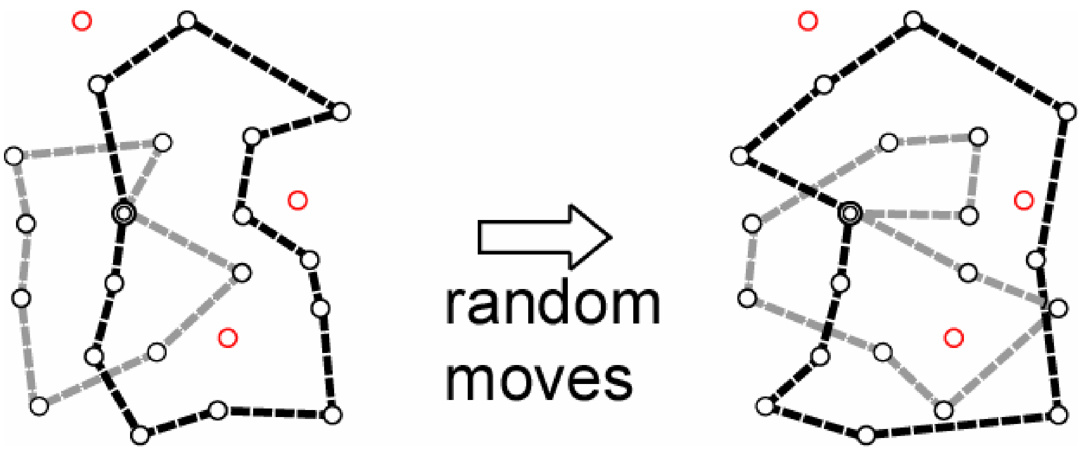
Procedure Delete_Route (σ)

Randomly remove a route from σ and initialize *EP* ;
Initialize all penalty counters: $p[v] = 1$ ($v = 1, \ldots, N$) ;
while *EP* is not empty do
    Select and remove v_{in} from *EP* (LIFO strategy)
    if v_{in} can be inserted into σ then
        Insert v_{in} into σ ;
    else
        Set: $p[v_{in}] = p[v_{in}] + 1$;
        Execute the insertion-ejection move on σ such that $P_{sum} = p[v_{out}^{(1)}] +, \ldots, + p[v_{out}^{(k)}]$ is minimized ;
        Add $v_{out}^{(1)}, \ldots, v_{out}^{(k)}$ to *EP* ;
        σ := Perturb (σ) ;
    end if
end while

Perturb procedure

Procedure Perturb (σ): Outline

- Random local search moves are executed inside σ for I_{rand} (=1000) times.
- Each move is randomly selected from 2-opt*, relocation and exchange moves.
- σ must be feasible after each move.



Improving the main framework

while *EP* is not empty **do**

    Select and remove v_{in} from *EP* (LIFO queue)

    if v_{in} can be inserted into σ (by the simple insertion) **then**

        Insert v_{in} into σ ;   // simple insertion for v_{in}

    else

        σ := Squeeze (v_{in}, σ) ;   // more powerful insertion for v_{in}

    endif

    if (v_{in} is not inserted) **then**

        Set: $p[v_{in}] = p[v_{in}] + 1$;

        Execute the insertion-ejection move on σ …… ;

        Add $v_{out}^{(1)}$, . . ., $v_{out}^{(k)}$ to *EP* ;

        σ := Perturb (σ) ;

    end if

end while

Squeeze procedure

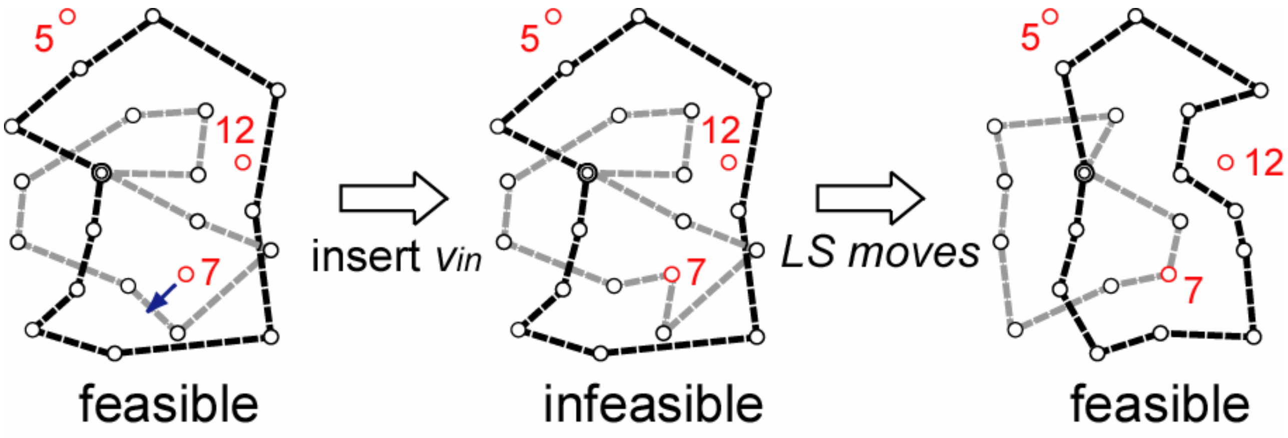
Procedure Squeeze (v_{in}, σ) : Outline

- Insert v_{in} into σ by allowing the violation of the constraints.
- Local search based repair procedure restores the feasibility.

  - 2-opt, relocation, exchange moves are applied inside σ
  - A solution is evaluated by a penalty function to guide it toward feasible solutions.
  - A standard hill climbing.

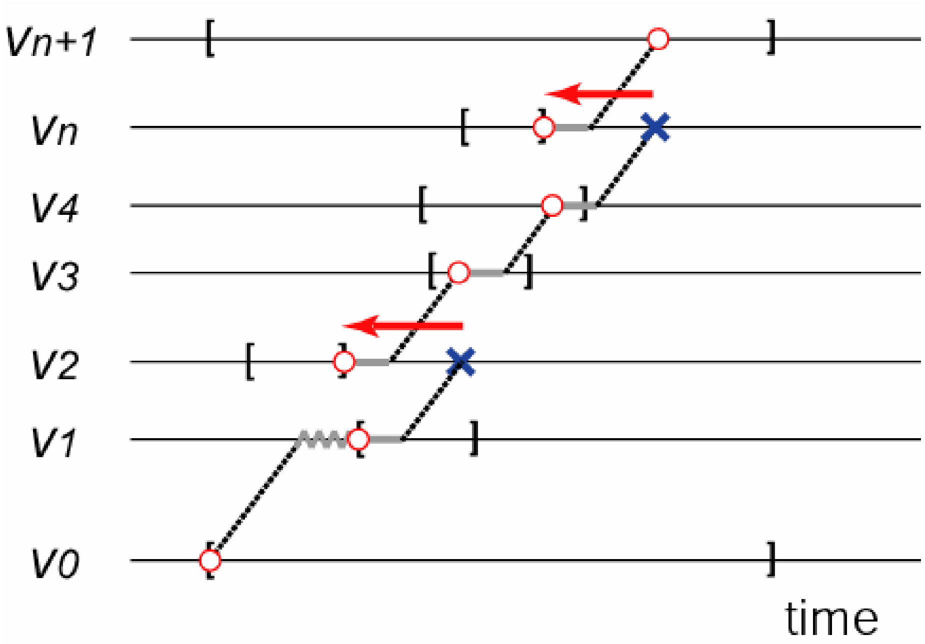
- Penalty function: $F_p(\sigma) = F_c(\sigma) + \alpha \cdot F_{tw}(\sigma)$

  - penalty terms for the capacity and time window constraints



Penalty term: $F_{tw}(\sigma)$ (Nagata, 07)

- Time window penalty for a route: TW_r
  - A sequence of a route: $r = \langle v_0, v_1, \ldots\ldots, v_n, v_{n+1} \rangle$
  - TW_r = sum of the length of the red arrows
- Time window penalty for σ
  - $F_{tw}(\sigma) = \sum_{r=1}^{m} TW_r$



- $\Delta F_{tw}(\sigma)$ by a local search move from 2-opt, relocation and exchange is calculated in $O(1)$ time

Experiments

Experimental settings

- Algorithm-G: Squeeze procedure is not used.
- Algorithm-GS: Squeeze procedure is used.

Benchmarks

- Gehring and Homberger's benchmarks
  - Instance sets of 200, 400, 600, 800 and 1000-customer
  - Each set consists of 60 instances.

Comparisons

- The best-known solutions taken from (Pisinger and Ropke, 07), (Ibaraki et. al., to appear), (Lim and Zhang, 07), (Gagnon et. al., 07), and SINTEF website (ignore several wrong solutions).

Results

- CNV: The cumulative number of vehicles in each problem size instances
- Best CNV: CNV in the best-known solutions.
- Our result: The difference in the CNV from the Best CNV

	Best	Algorithm-G			Algorithm-GS			
N	CNV	1min	10min	60min	1min	10min	60min	*N*/200 h
200	694	0	0	0	0	0	0	0
400	1382	+3	+2	+2	+3	0	- 2	- 2
600	2068	+7	- 1	- 1	+1	- 1	- 3	- 3
800	2739	+15	- 1	- 3	+2	- 2	- 4	- 5
1000	3425	+12	0	- 2	+2	- 5	- 6	- 8
total	10308	+37	0	- 4	+8	- 8	-15	-18
# Fail to reach best-known		37	6	4	10	2	0	0
# Find new best		0	6	8	2	10	15	18

New best-know solutions

- The 18 new best-known solutions

Instances	N	Best-known	New Best	Time (min)
C2_4_8	400	12	11	60
RC2_4_5	400	9	8	10
C1_6_6	600	60	59	60
C1_6_7	600	58	57	60
RC2_6_5	600	12	11	10
C1_8_2	800	73	72	1
C1_8_6	800	80	79	10
C1_8_8	800	74	73	240
C2_8_6	800	24	23	60
RC2_8_1	800	19	18	1
C110_6	1000	100	99	10
C110_7	1000	98	97	60
C110_8	1000	93	92	300
C210_3	1000	29	28	10
C210_6	1000	30	29	10
C210_7	1000	30	29	10
C210_8	1000	29	28	300
C21010	1000	29	28	10

Conclusions

- A powerful route minimization heuristic for the VRPTW is presented.
- The idea of the main framework is simple.
  - The concept of GLS is combined with a general framework of the *EP*.
- The main framework is further improved by the Squeeze procedure.
- The results of these methods are promising.
- The main framework can be generalized and applied to other combinational optimization problems (ongoing work).