Combinatorial Optimisation and Heuristics for Getting Around --PART 2--

Slides are (mostly) by: Phil Kilby
Speaker today is: Charles Gretton

Outline

PART 1

- Combinatorial Optimisation
- The vehicle routing problem (VRP)
  - VRP Variants
- Solving the Combinatorial Optimisation problems
  - Exact methods
  - Heuristic Construction
  - Auction
  - Local Search
  - Meta-heuristics

PART 2

- Large Neighbourhood Search
- A CP model for the VRP
- Propagation
- Large Neighbourhood Search revisited

Large Neighbourhood Search

- Originally developed by Paul Shaw (1997)
- I teach the “adaptive” version, originally Ropke & Pisinger (2007)
- A.k.a “Record-to-record” search

- Destroy part of the solution
  - Remove visits from the solution
- Re-create solution
  - Use favourite construct method to re-insert customers
- If the solution is better, keep it
- Repeat

Large Neighbourhood Search

Destroy part of the solution (*Select* method)

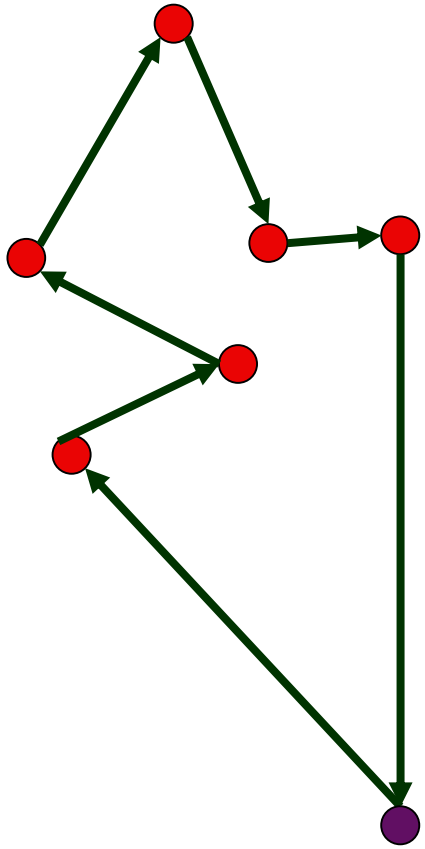
- Remove some visits
- Move them to the “unassigned” lists

Large Neighbourhood Search

Destroy part of the solution (*Select* method)

Examples

- Remove a sequence of visits

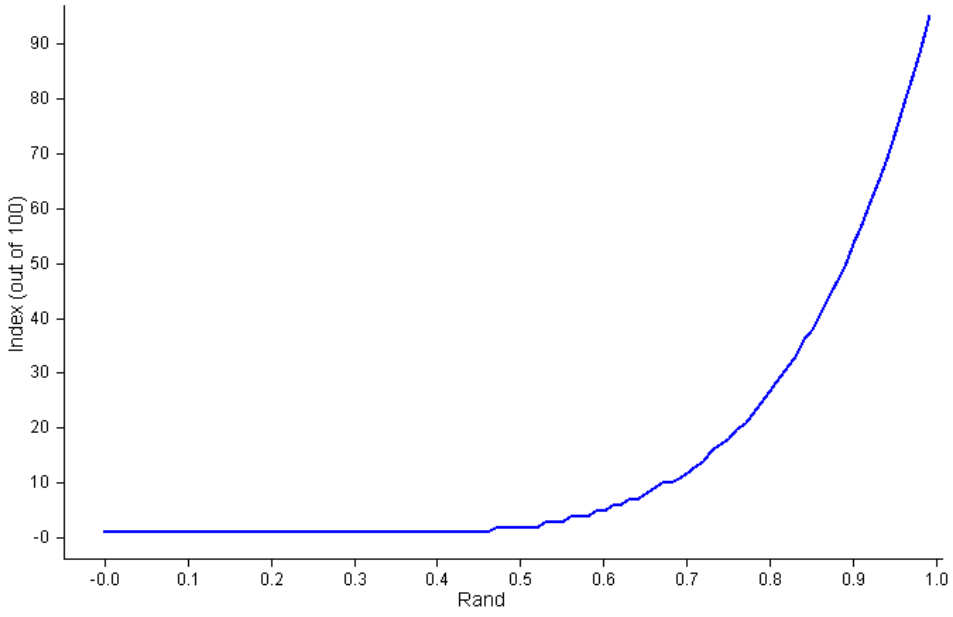


Large Neighbourhood Search

Destroy part of the solution (*Select* method)

Examples

- Choose longest (worst) arc in solution
  - Remove visits at each end
  - Remove nearby visits
- Actually, choose r^{th} worst
- $r = n * (\text{uniform}(0,1))^y$
- $y \sim 6$
  - $0.5^6 = 0.016$
  - $0.9^6 = 0.531$



Large Neighbourhood Search

Destroy part of the solution (*Select* method)

Examples

- Dump visits from k routes (k = 1, 2, 3)
  - Prefer routes that are close,
  - Better yet, overlapping

Large Neighbourhood Search

Destroy part of the solution (*Select* method)

Examples

- Choose first visit randomly
- Then, remove "related" visits
  - Based on distance, time compatibility, load

$$R_{ij} = \varphi C_{ij} + \chi(|a_i - a_j|) + \psi(|q_i - q_j|)$$

Large Neighbourhood Search

Destroy part of the solution (*Select* method)

Examples

- Dump visits from the smallest route
  - Good if saving vehicles
  - Sometimes fewer vehicles = reduced travel

Large Neighbourhood Search

Destroy part of the solution (*Select* method)

- Parameter: Max to dump
  - As a % of n?
  - As a fixed number e.g. 100 for large problems

- Actual number is uniform rand (5, *max*)

Large Neighbourhood Search

Re-create solution

- Systematic search
  - Smaller problem, easier to solve
  - Can be very effective

  - E.g.: CP Backtracking search
  - Constraint: objective must be less than current
  - (Implicitly) Look at all reconstructions

  - Backtrack as soon as a better sol is found
  - Backtrack anyway after *too many* failures

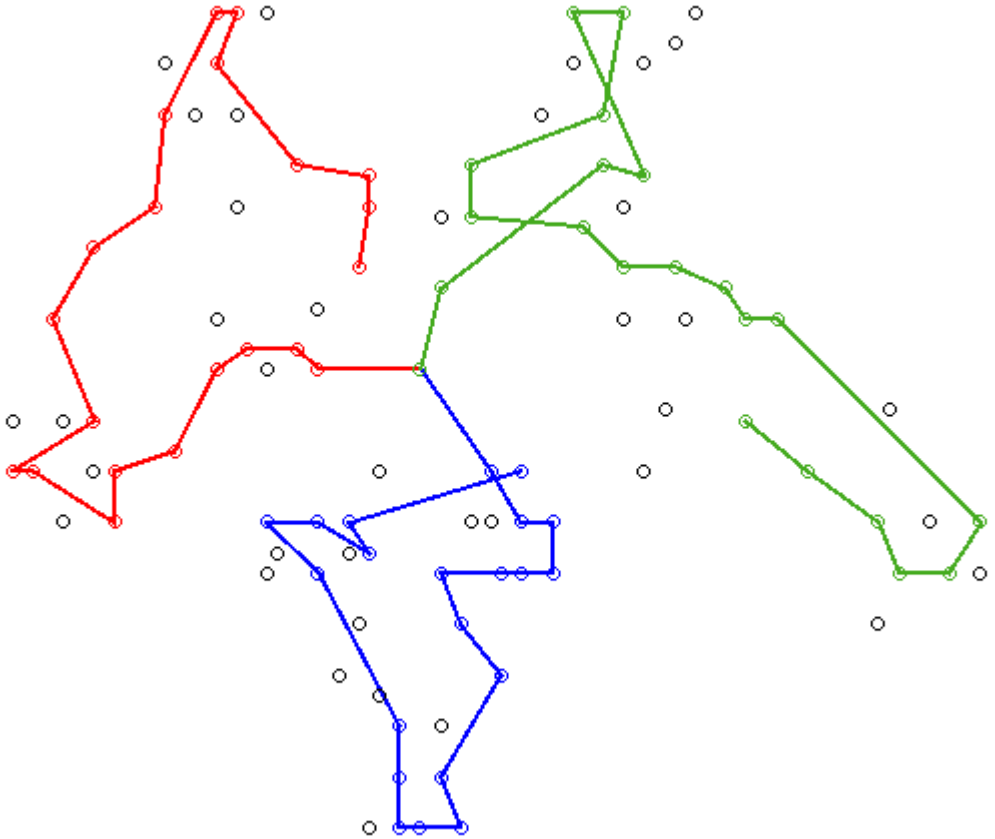
Large Neighbourhood Search

Re-create solution

- Use your favourite insert method

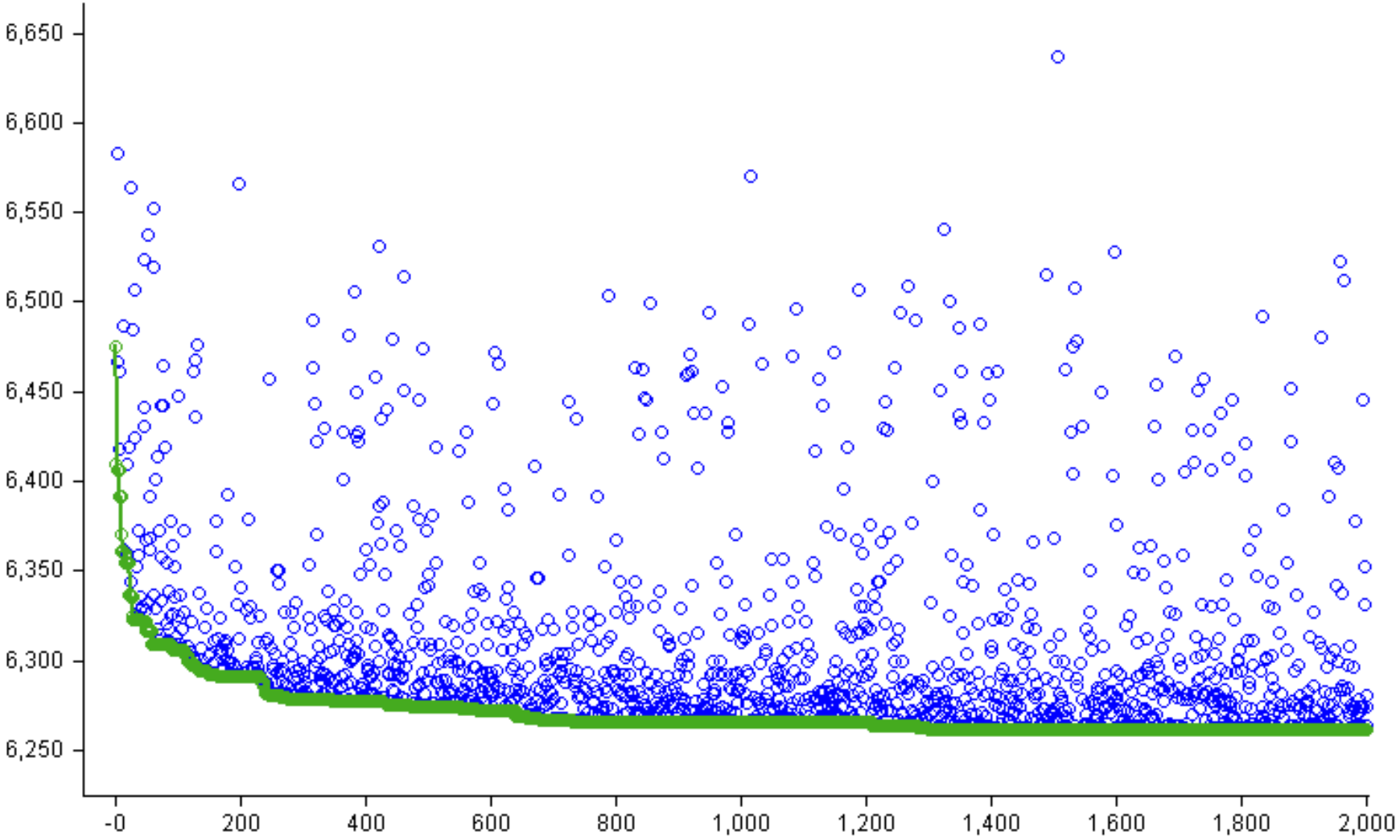
- Better still, use a portfolio
  - Ropke: Select amongst
    - Minimum Insert Cost
    - Regret
    - 3-regret
    - 4-regret

Large Neighbourhood Search



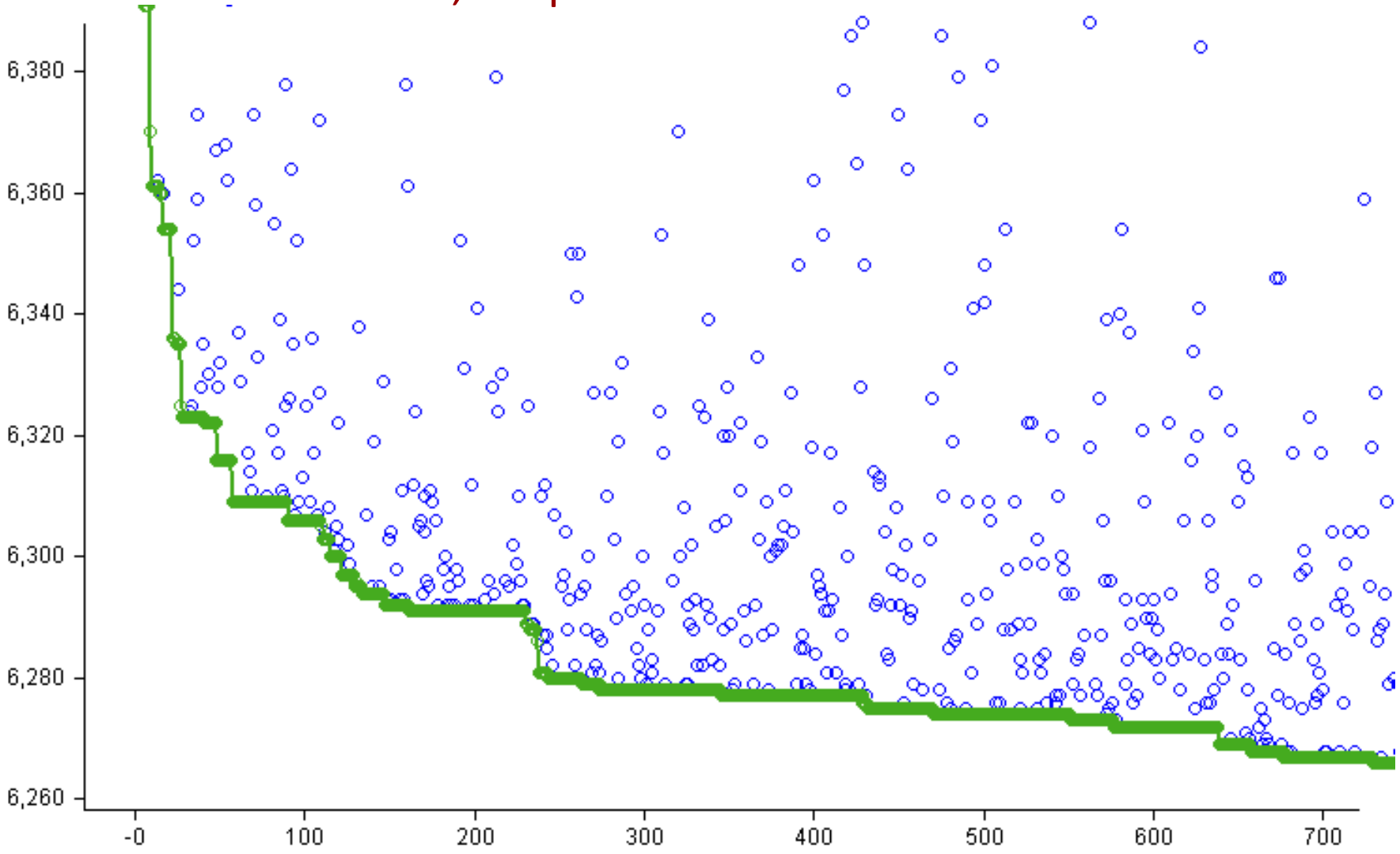
Large Neighbourhood Search

If the solution is better, keep it



Large Neighbourhood Search

If the solution is better, keep it



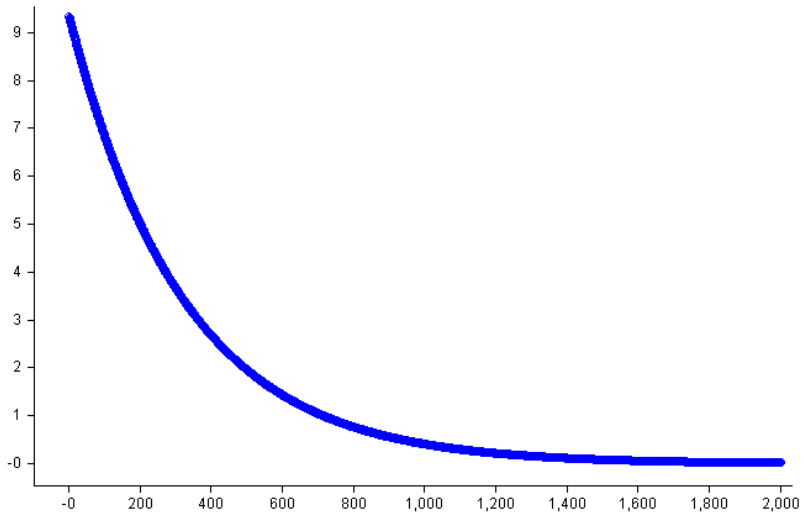
Large Neighbourhood Search

If the solution is better, keep it

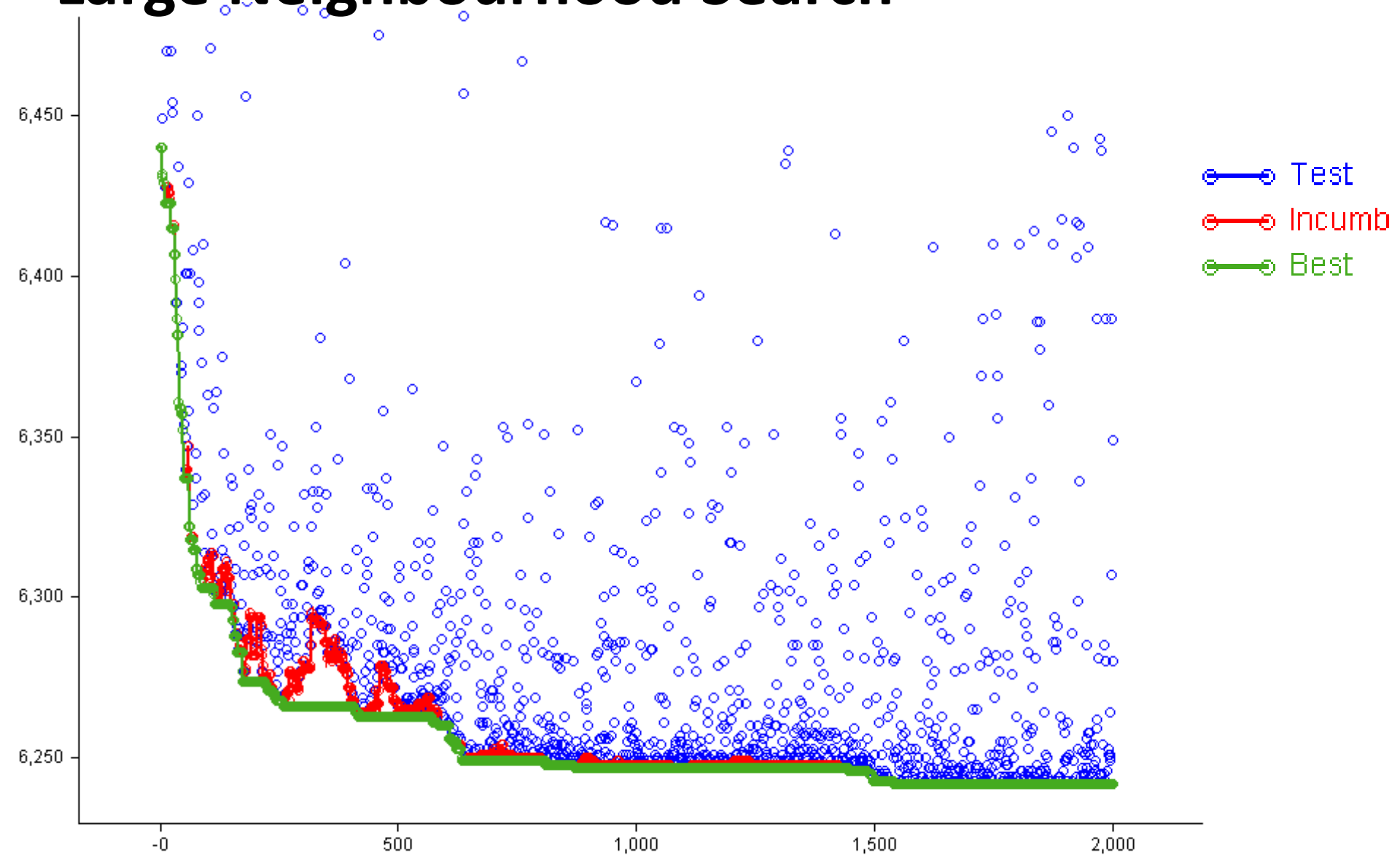
- Can use Hill-climbing
- Can use Simulated Annealing
- Can use Threshold Annealing
- …

Large Neighbourhood Search

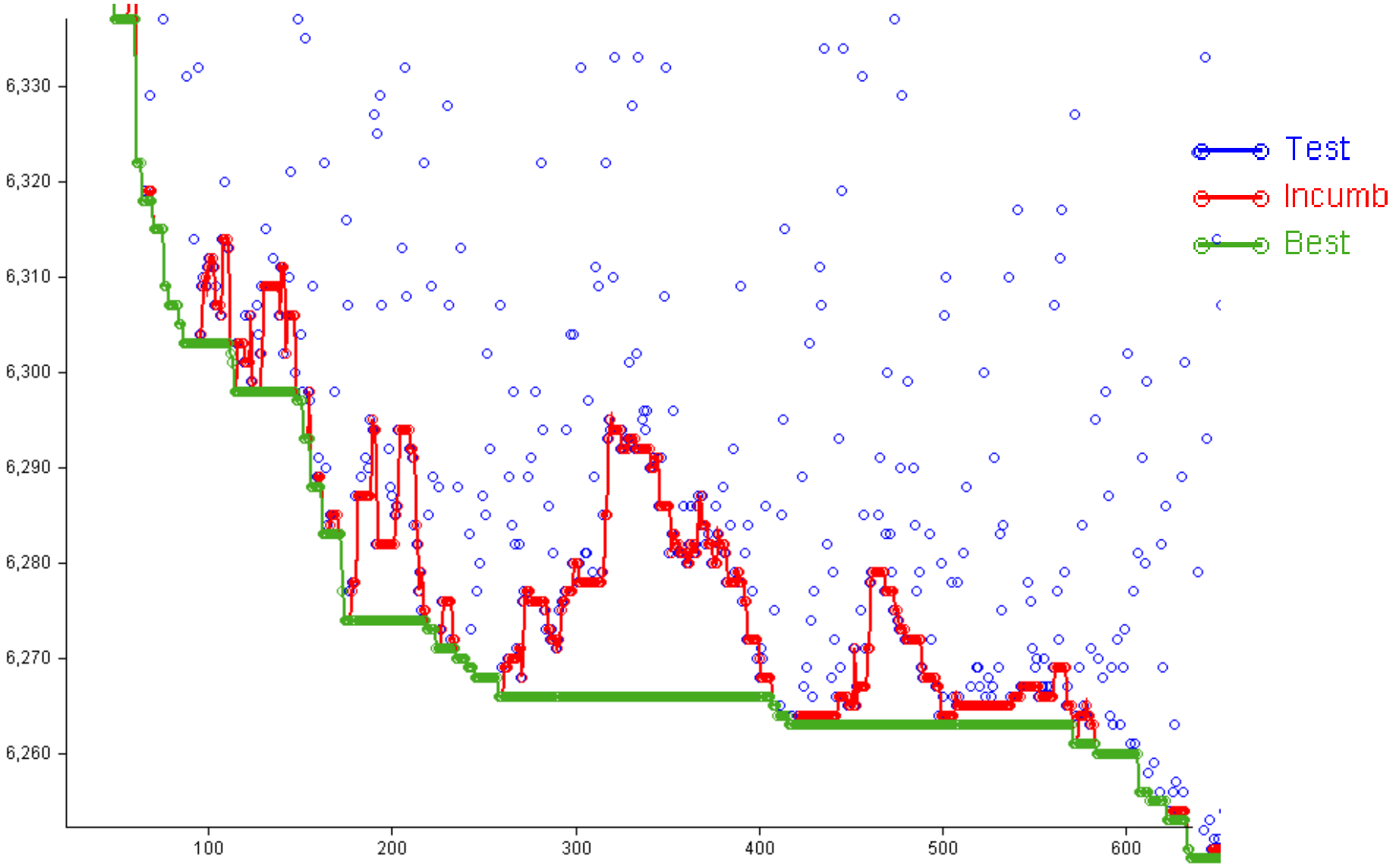
$$P(accept\ increase\ \Delta) = e^{-\Delta/T}$$



Large Neighbourhood Search



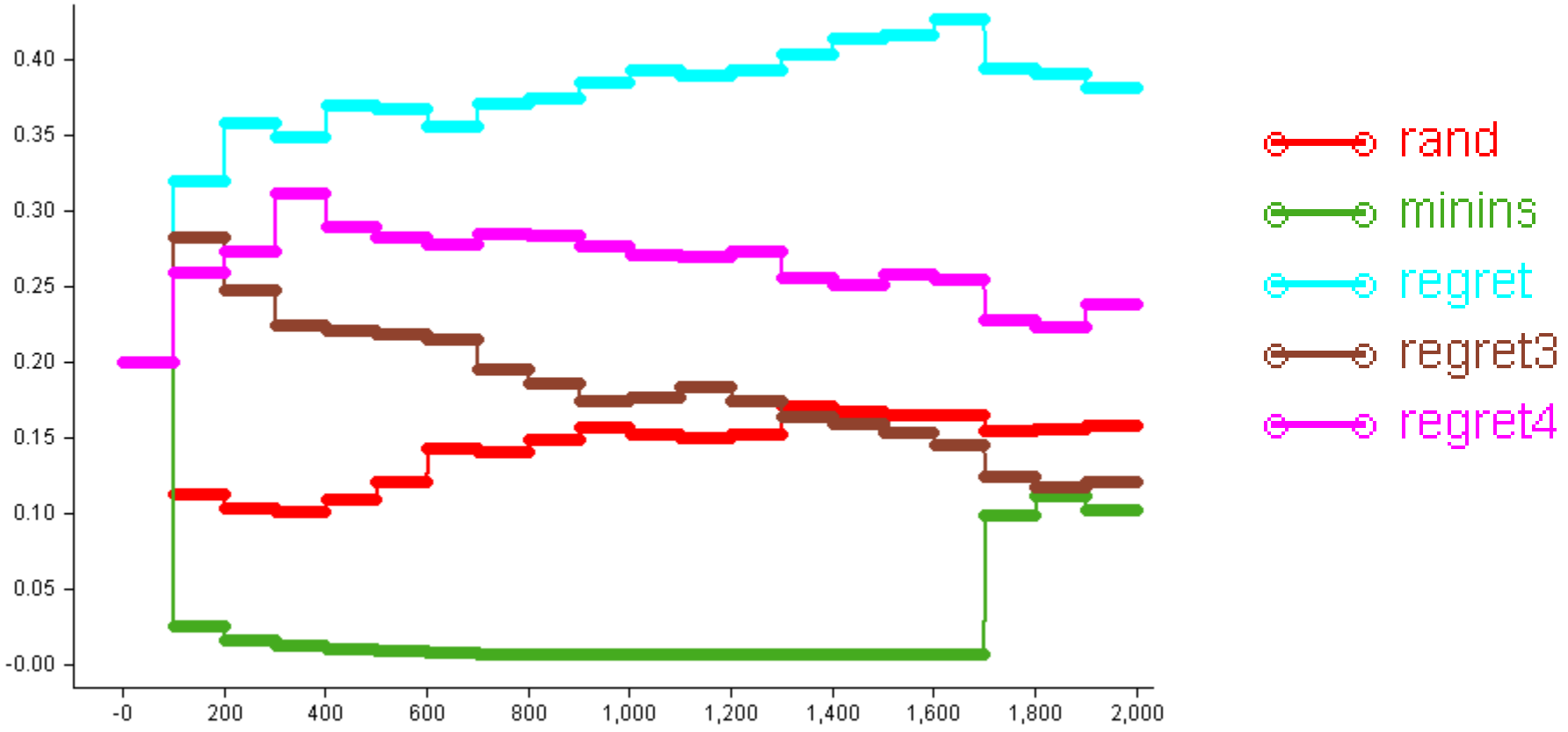
Large Neighbourhood Search



Large Neighbourhood Search

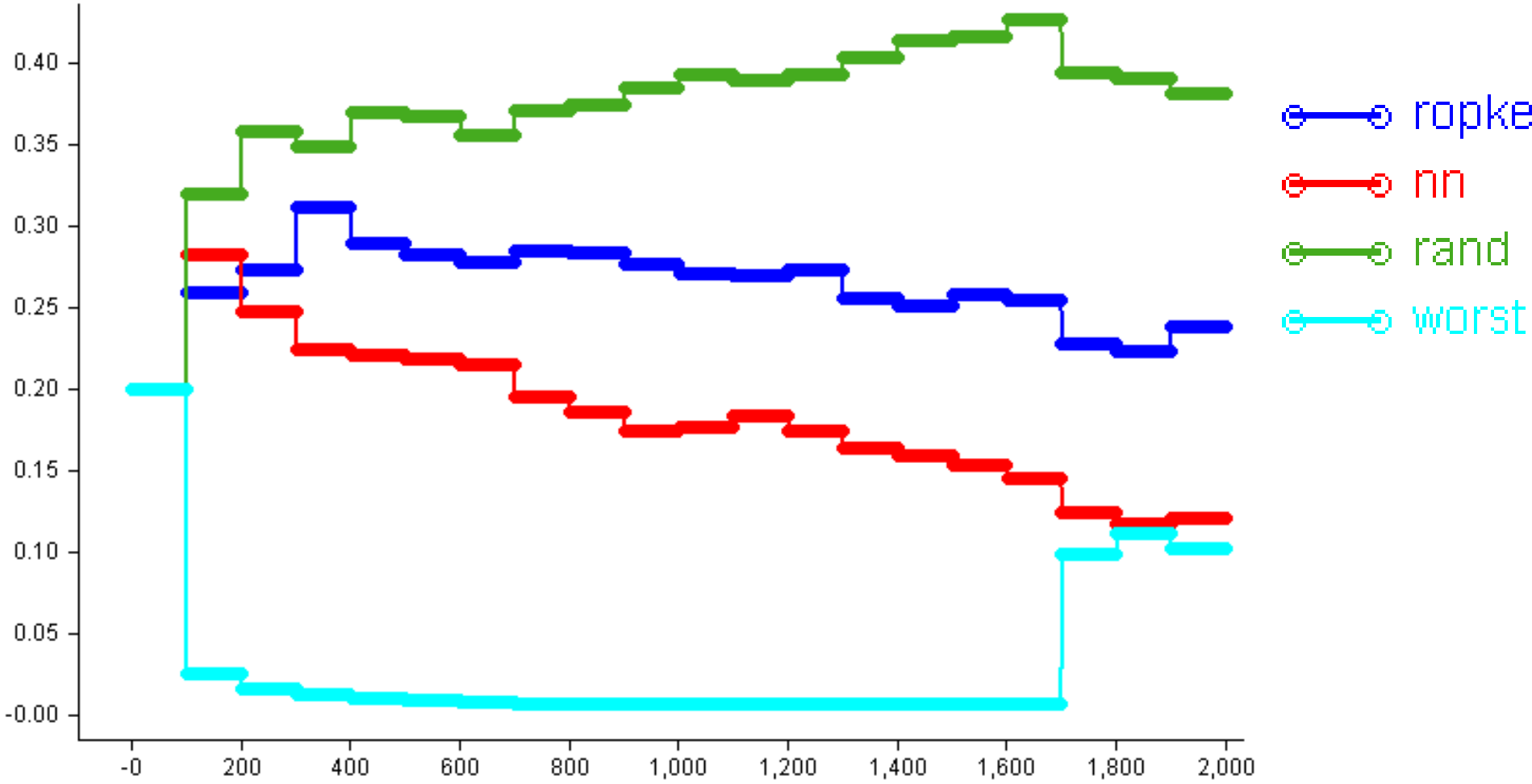
Adaptive

– Ropke adapts choice based on prior performance
  – "Good" methods are chosen more often



Large Neighbourhood Search

Adapting Select method (Selecting customers to remove)



Large Neighbourhood Search

Ropke & Pisinger (with additions) can solve a variety of problems

- VRP
- VRP + Time Windows
- Pickup and Delivery
- Multiple Depots
- Multiple Commodities
- Heterogeneous Fleet
- Compatibility Constraints
- Visual Attractiveness Soft Constraints:
  - http://users.cecs.anu.edu.au/~charlesg/
  - reveal.js/2014/VisuallyAttractive/
  - visually_attractive_routes.html#/

Solution Methods

What's wrong with that?

- New constraint → new code
  - Often right in the core
- New constraints interact
  - e.g. Multiple time windows mess up duration calculation
- Code is hard to understand, hard to maintain

Solution Methods

An alternative:

Constraint Programming

… after the break….

Constraint Programming

CP offers a language for representing problems

- Decision variables
- Constraints on variables

Also offers techniques for solving the problems

- Systematic search
- Heuristic Search

Constraint Programming for the VRP

Constraint Programming

Advantages:

- Separate problem statement from data
- Expressive language for formulating constraints
- Each constraint encapsulated
- Constraints interact naturally

Disadvantages:

- Difficulty in representation
- Can be slower

Constraint Programming

Two ways to use constraint programming

- Rule Checker – i.e. validating the correctness of a solution
- Properly – i.e. calculating a solution, or search artefact

Rule Checker:

- Use favourite method to create/improve a solution
- Check it with CP
  - Very inefficient.

A CP Model for the VRP

Model

A (rich) vehicle routing problem

- n customers (fixed in this model)
- m routes (fixed in this model)
- fixed locations
  - where things happen
  - one for each customer + one for (each?) depot
- c commodities (e.g. weight, volume, pallets)
  - Know demand from each customer for each commodity
- Know time between each location pair
- Know cost between each location pair
  - Both obey triangle inequality

Vocabulary

- A *solution* is made up of *routes* (one for each vehicle)
- A *route* is made up of a sequence of *visits*
- Some visits serve a customer (*customer visit*)

(Some tricks)

- Each route has a “start visit” and an “end visit”
- Start visit is first visit on a route – location is depot
- End visit is last visit on a route – location is depot
- Also have an additional route – the unassigned route
  - Where visits live that cannot be assigned
  - Usual constraints (time, capacity, …) not applied

Referencing

Customers

- Each customer has an index in $N = \{1..n\}$
- Customers are ‘named’ in CP by their index

Routes

- Each route has an index in $M = \{1..m\}$
- Unassigned route has index 0
- Routes are ‘named’ in CP by their index

Visits

- Customer visit index same as customer index
- Start visit for route k has index $n + k$; aka $start_k$
- End visit for route k has index $n + m + k$; aka end_k

Referencing

Sets

- $N = \{1 \ldots n\}$ – customers
- $M = \{1 \ldots m\}$ – routes
- $R = M \cup \{0\}$ – includes ‘unassigned’ route
- $S = \{n+1 \ldots n+m\}$ – start visits
- $E = \{n+m+1 \ldots n+2m\}$ – end visits
- $V = N \cup S \cup E$ – all visits
- $V^S = N \cup S$ – visits that have a sensible successor
- $V^E = N \cup E$ – visits that have a sensible predecessor

Data

We know (note uppercase)

- V_i The ‘value’ of customer i
- D_{ik} Demand by customer i for commodity k
- E_i Earliest time to start service at i
- L_i Latest time to start service at i
- Q_{jk} Capacity of vehicle j for commodity k
- T_{ij} Travel time from visit i to visit j
- C_{ij} Cost (w.r.t. objective) of travel from i to j

Basic Variables

Successor variables: s_i

- s_i gives direct successor of i, i.e. the index of the next visit on the route that visits i
- $s_i \in V^E$ for i in V^S $s_i = 0$ for i in E

Predecessor variables p_i

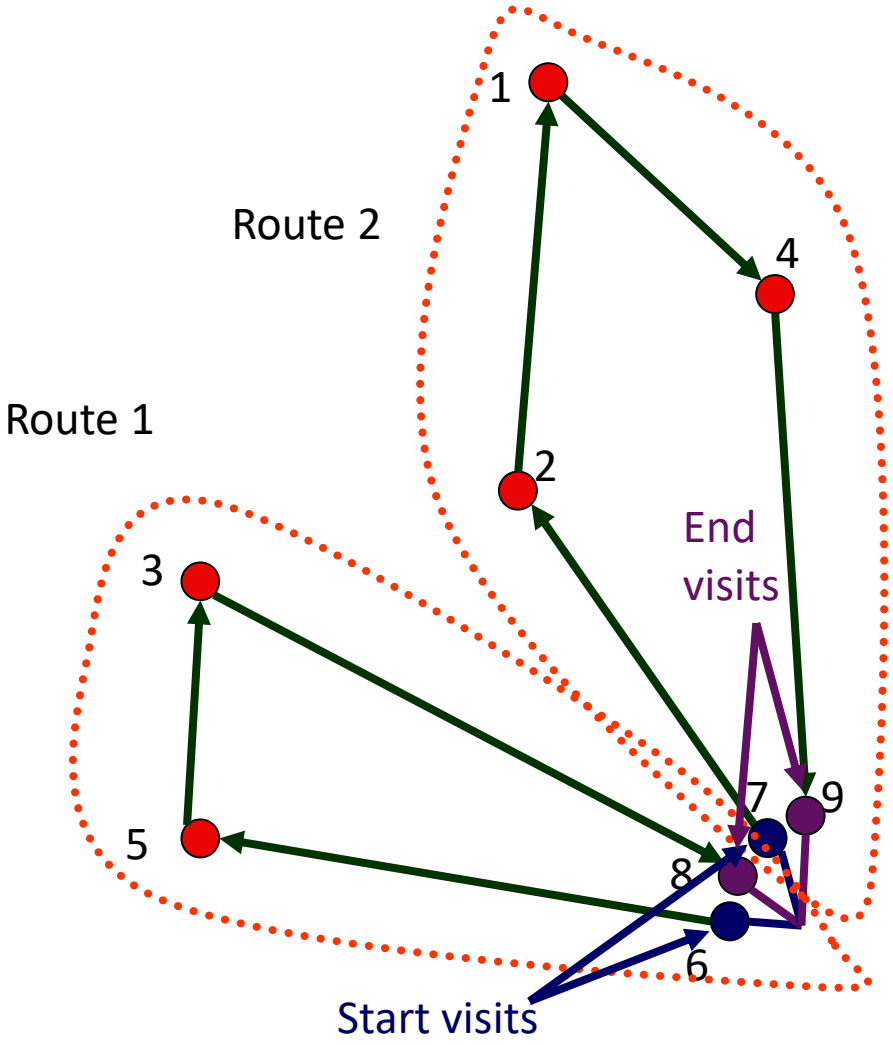
- p_i gives the index of the previous visit in the route
- $p_i \in V^S$ for i in V^E $p_i = 0$ for i in S
- Redundant – but empirical evidence for its use

Route variables r_i

- r_i gives the index of the route (vehicle) that visits i
- $r_i \in R$

Example

i	s_i	p_i	r_i
1	4	2	2
2	1	7	2
3	8	5	1
4	9	1	2
5	3	6	1
6	5	0	1
7	2	0	2
8	0	3	1
9	0	4	2



Other variables

Accumulation Variables

- q_{ik} Quantity of commodity k after visit i
- c_i Objective cost getting to i

For problems with time constraints

- a_i Arrival time at i
- t_i Start time at i (time service starts)
- d_i Departure time at i

- Actually, only t_i is required, but others allow for expressive constraints

What can we model?

- Basic VRP
- VRP with time windows
- Multi-depot
- Heterogeneous fleet
- Open VRP (vehicle not required to return to base)
  - Requires "*anywhere*" location
  - Route end visits located at *anywhere*
  - distance FROM i TO *anywhere* = 0
- Compatibility
  - Customers on different / same vehicle
  - Customers on/not on given vehicle
- Pickup and Delivery problems

What can we model?

- Variable load/unload times
  - by changing departure time relative to start time
- Dispatch time constraints
  - e.g. limited docks
  - s_i for i in S is load-start time
- Depot close time
  - Time window on end visits
- Fleet size and mix
  - Add lots of vehicles
  - Need to introduce a ‘fixed cost’ for a vehicle
  - C_{ij} is increased by fixed cost for all $i \in S$, all $j \in N$

What cannot we model

- cannot handle dynamic problems
  - Fixed domain for s, p, r vars
- cannot introduce new visits post-hoc
  - E.g. optional driver break must be allowed at start
- cannot handle multiple visits to same customer
  - 'Larger than truck-load' problems
  - If qty is fixed, can have multiple visits / cust
  - Heterogeneous fleet is a pain
- Can handle time- or vehicle-dependent travel times/costs with mods
- Can handle Soft Constraints with mods

Objective

Want to minimize

- sum of objective (c_{ij}) over used arcs, plus
- value of unassigned visits

minimize

Cost of dropped visits

Cost of getting to visit i

$$\sum_{i \in E} c_i + \sum_{i \mid r_i = 0} v_i$$

Dummy route

Of unassigned visits

Basic constraints

Path (S, E, { $s_i \mid i \in V$ })

AllDifferent ({ $p_i \mid i \in V^E$ })

Accumulate obj.

$$c_{s_i} = c_i + C_{i,s_i} \quad \forall i \in V^S$$

Accumulate time

$$a_{s_i} = d_i + T_{i,s_i} \quad \forall i \in V^S$$

Time windows

$$t_i \geq a_i \qquad \forall i \in V$$

$$t_i \leq L_i \qquad \forall i \in V$$

$$t_i \geq E_i \qquad \forall i \in V$$

$$t_i = 0 \qquad \forall i \in S$$

Constraints

Load

$$q_{s_i k} = q_{ik} + Q_{s_i k} \quad \forall i \in V^S, k \in C$$

$$q_{ik} \leq Q_{r_i k} \quad \forall i \in V, k \in C$$

$$q_{ik} \geq 0 \quad \forall i \in V, k \in C$$

$$q_{ik} = 0 \quad \forall i \in S, k \in C$$

Consistency

$$s_{p_i} = i \quad \forall i \in V^S$$

$$p_{s_i} = i \quad \forall i \in V^E$$

$$r_i = r_{s_i} \quad \forall i \in V^S$$

$$r_{n+k} = k \quad \forall k \in M$$

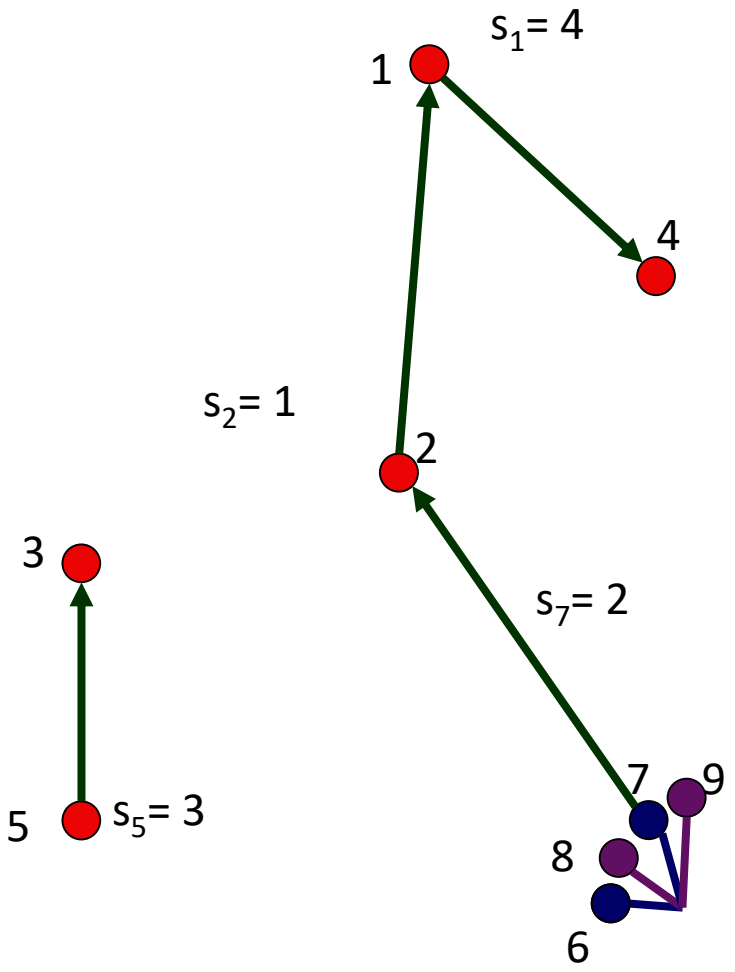
$$r_{n+m+k} = k \quad \forall k \in M$$

Solving

Pure CP

- Assign to ‘successor’ variables
- Form chains of visits
- Decision 1: Which visit to insert (== variable choice)
- Decision 2: Where to insert it (== value choice)

- ‘***Rooted chain***’ starts at Start
- ‘***Free chain***’ otherwise
- Can reason about *free chains* but rooted chains easier
- “`last(k)`” is last visit in chain starting at `k`



Propagation – Cycles

Subtour elimination

- Rooted chains are fine
- For free chains:
  - for any chain starting at j,
    - remove j from $S_{last(j)}$

- Some CP libraries have built-ins
  - Comet: ‘circuit’
  - ILOG: Path constraint

Propagation – Capacity (assume +ve qtys)

Rooted Chain

- For each visit i with p_i not bound
  - For each route k in domain of r_i
    - If spare space on route k won't allow visit i
      - Remove k from r_i
      - Remove i from $p_{end(k)}$
      - Remove *i* from $s_{last(start(k))}$

Assume +ve qtys

Free chains

- As above (pretty much)
- Before adding visit to chain and increasing chain to load L
  - v = Count routes with free space ≥ L
  - c = Count free chains with load ≥ L
  - if $c > v$, cannot form chain, backtrack

Propagation – Time

Time Constraints:

Rooted chains:

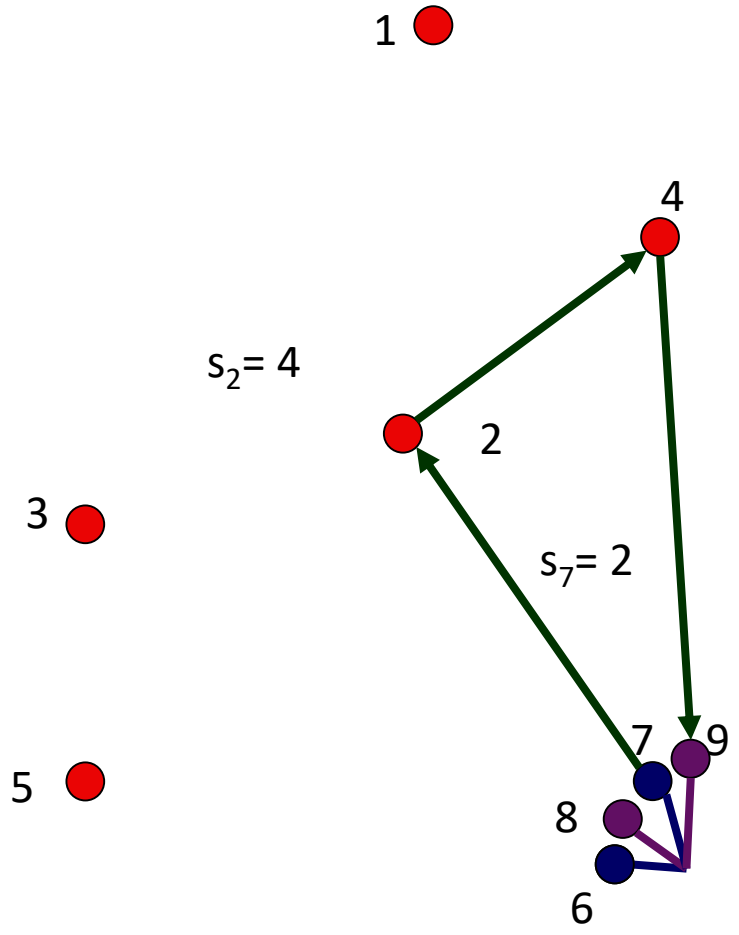
- For each route k
  - For each visit i in domain of $s_{last(k)}$
    - If vehicle cannot reach i before E_i
      - Remove k from r_i
      - Remove i from $s_{last(start(k))}$

Free chains:

- Form “implied time window” from chain
- Rest is as above

A gotchya: Chronological backtracking

Assign a successor

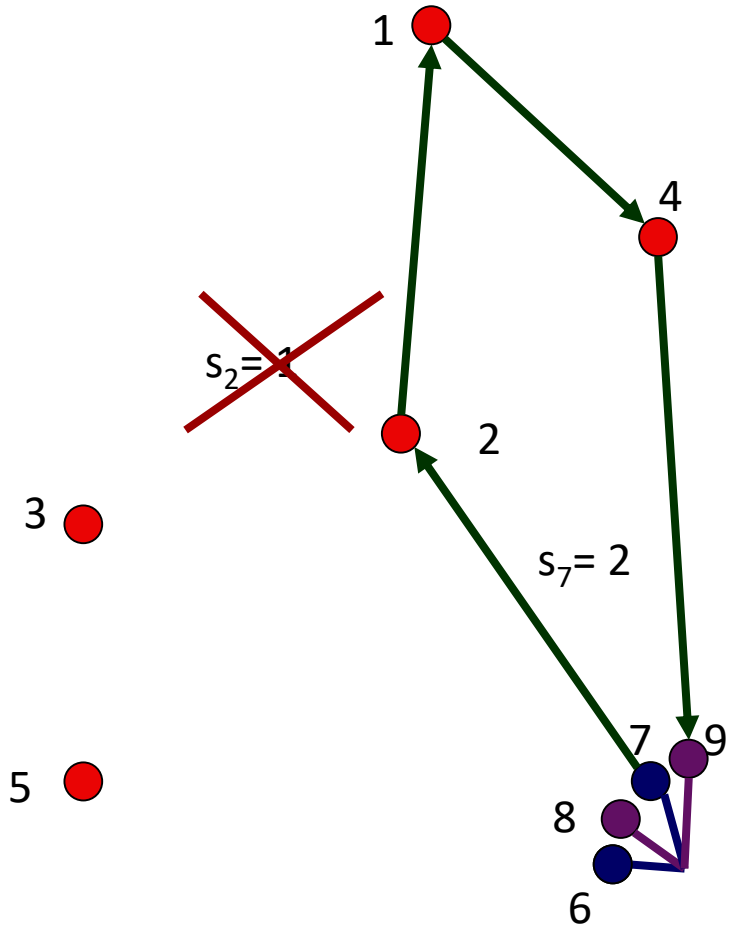


A gotchya: Chronological backtracking

Assign a successor

Cannot re-assign

- Domains can only shrink



Gotchya Resolution

Assign a successor

s_1 = [2,3,4,5,8,9]

s_2 = [1,3,4,5]

s_3 = [1,2,4,5,8,9]

s_4 = [1,3,5,9]

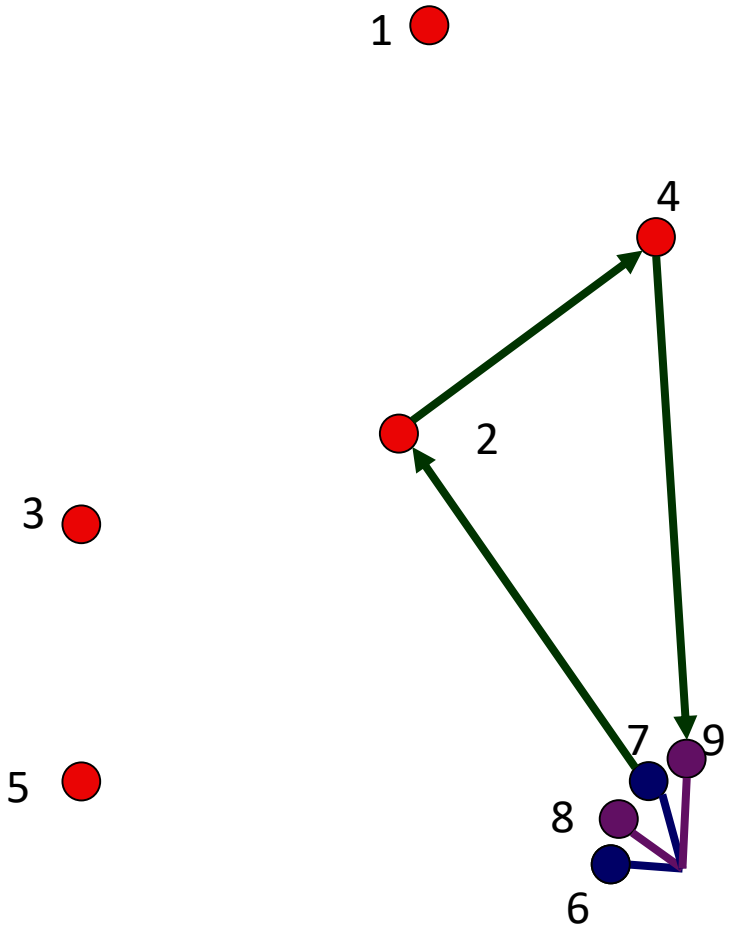
s_5 = [1,2,3,4,8,9]

s_6 = [1,3,5,8]

s_7 = [1,2,3,5]

s_8 = [0]

s_9 = [0]



Gotchya Resolution

Assign a successor

s_1 = [3,4,5]

s_2 = [1,3,5]

s_3 = [1,2,4,5,8,9]

s_4 = [3,5,9]

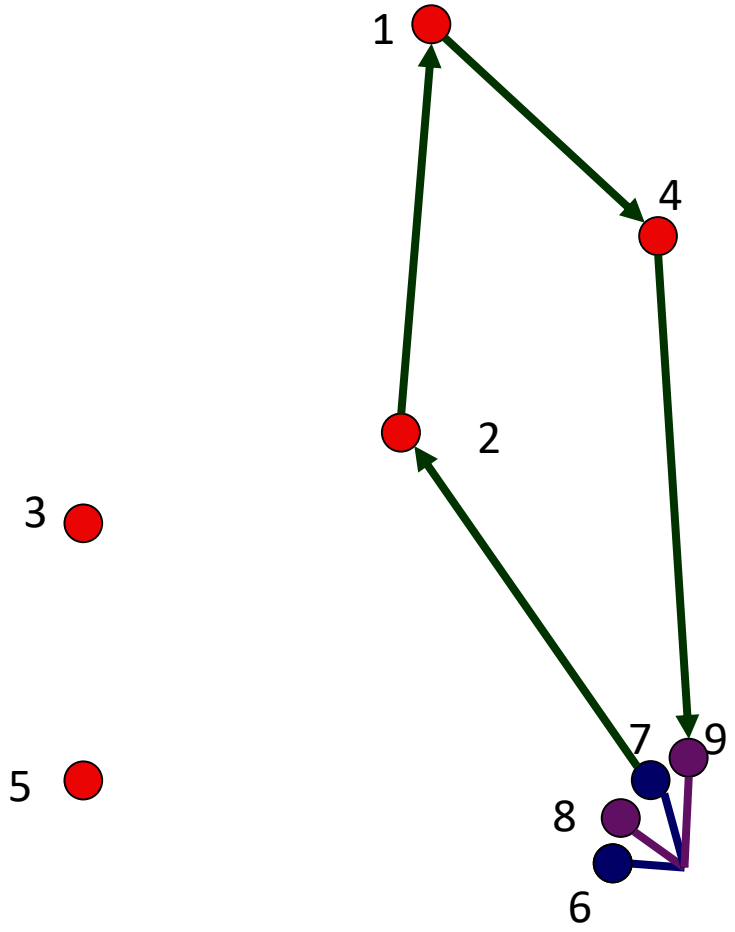
s_5 = [1,2,3,4,8,9]

s_6 = [3,5,8]

s_7 = [2,3,5]

s_8 = [0]

s_9 = [0]



Gotchya Resolution

Assume: $T_{i,j} = 1$ for all i, j

TWs for 1-5 are are [1-3]

a_1 = [1,2,3]

a_2 = [1,2]

a_3 = [1,2,3]

a_4 = [2,3]

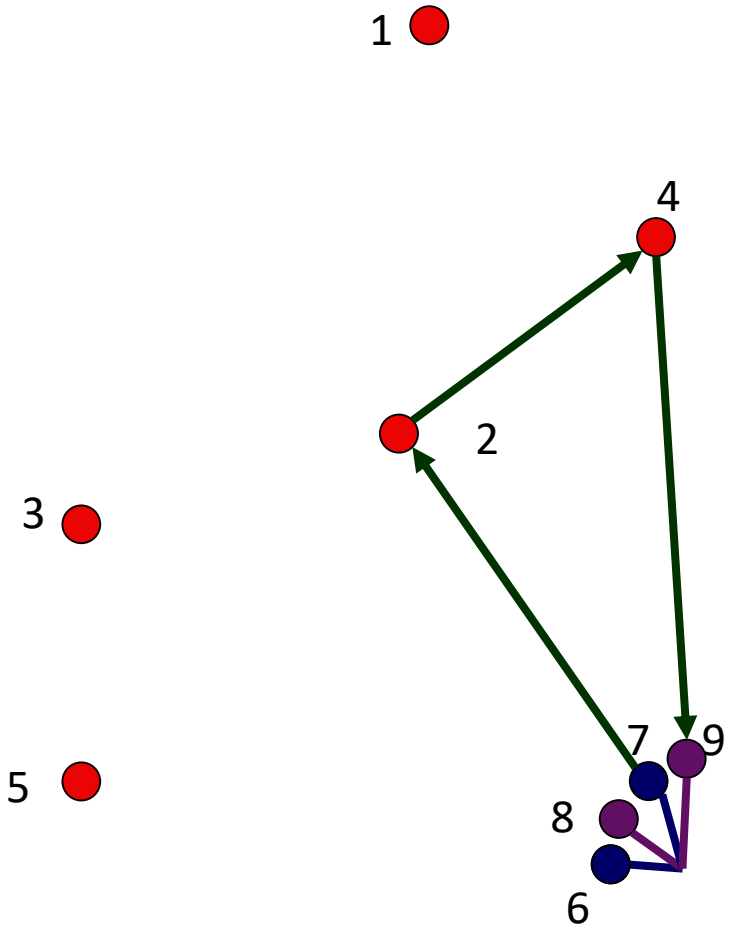
a_5 = [1,2,3]

a_6 = [0]

a_7 = [0]

a_8 = [0,1,2,3,4]

a_9 = [3,4]



Gotchya Resolution

Assume: $T_{i,j} = 1$ for all i, j

TWs for 1-5 are are [1-3]

a_1 = [2]

a_2 = [1]

a_3 = [1,2,3]

a_4 = [3]

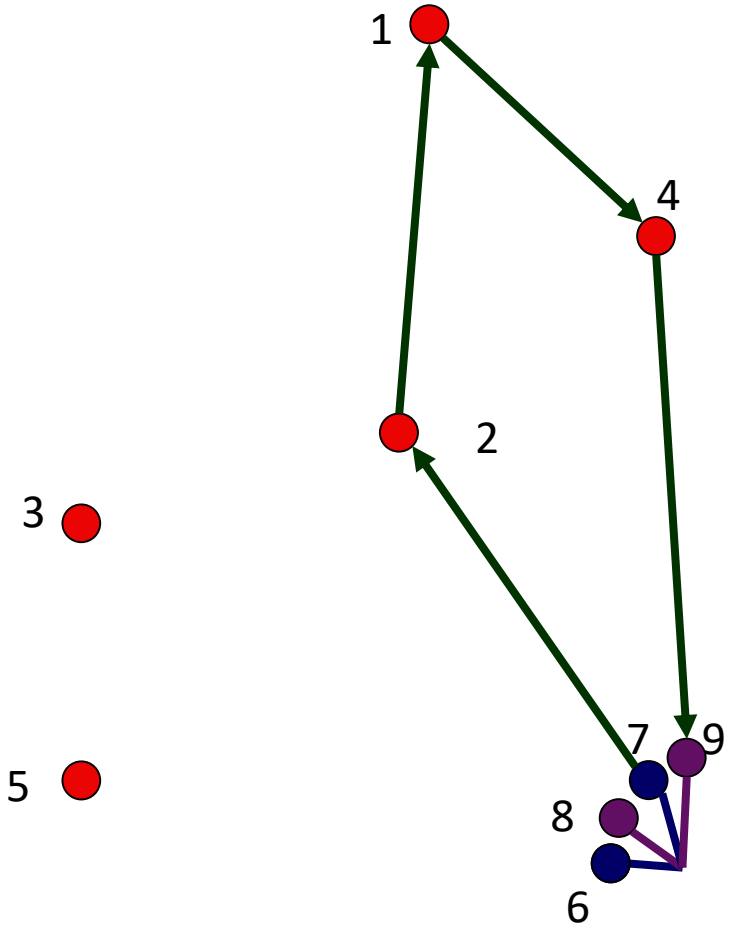
a_5 = [1,2,3]

a_6 = [0]

a_7 = [0]

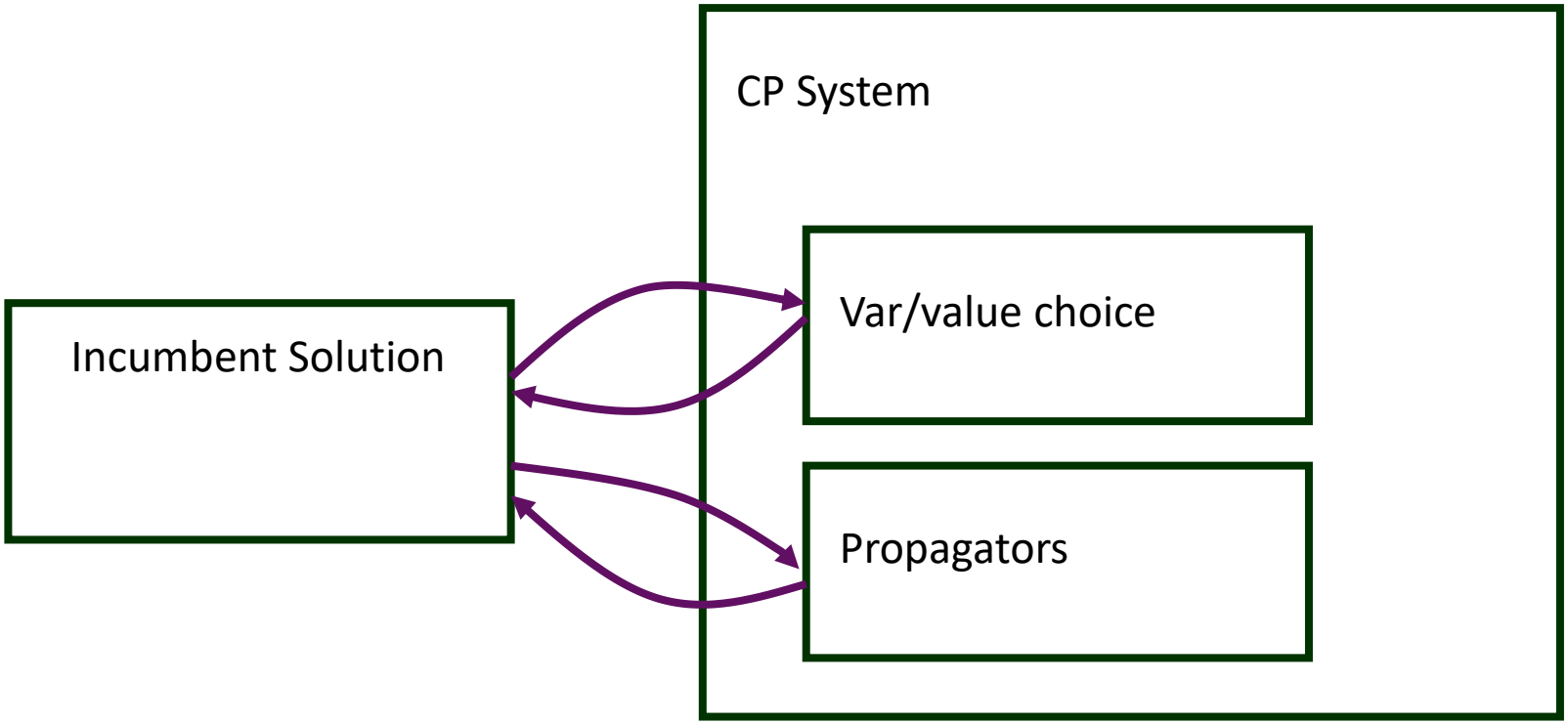
a_8 = [0,1,2,3,4]

a_9 = [4]



Propagation – an alternative

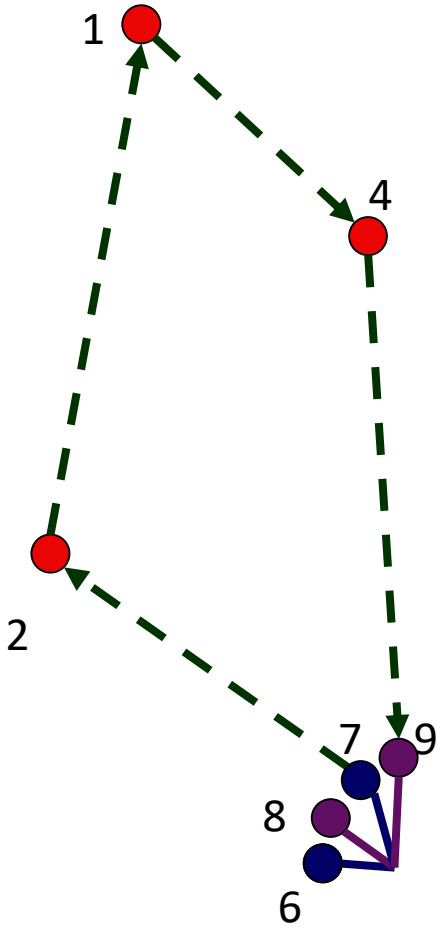
- Alternative relies on knowledge of “incumbent” solution
- Use shared data structure
- Can use full insertion (insert into middle of ‘chain’)



Propagation

Insertion ‘in the middle’

- Propagators know incumbent solution
- Essentially propagate partial order
- Only propagate non-binding implications
- e.g. don’t “set” $s_1 \; s_2$ here:
- But 1 is removed from s_4 , s_6
- Can calculate earliest start, latest start to bound start time
- Can calculate feasible inserts, and update s vars appropriately
- Strong propagation for capacity, time constraints



Insertion

- Allows for maximum propagation
- Allows constraints to influence solution progressively

- e.g. Blood delivery constraints
  - Delivery within 20 minutes of pickup
  - As soon as one is fixed implication flows to other

- e.g. Driver break
  - Extra request (*a-priori*) with time constraints that relate to other breaks
  - Special propagator that removes requests that cannot be inserted in a route without violating rules

Example

5 R(equests); 10 D(demands) @ each request; E(arliest) arrival time
L(atest) departure time; S(ervice) time, Q is vehicle capacity

	R1	R2	R3	R4	R5	R6	R7	R8	R9
$D(emand)_i$	10	10	10	10	10	0	0	0	0
$E(arliest)_i$	20	20	20	20	20	0	0	0	0
$L(atest)_i$	150	65	150	150	150	200	200	200	200
$S(ervice)_i$	10	10	10	10	10	0	0	0	0

+ Request Compatibility
  Different Routes: R2, R4

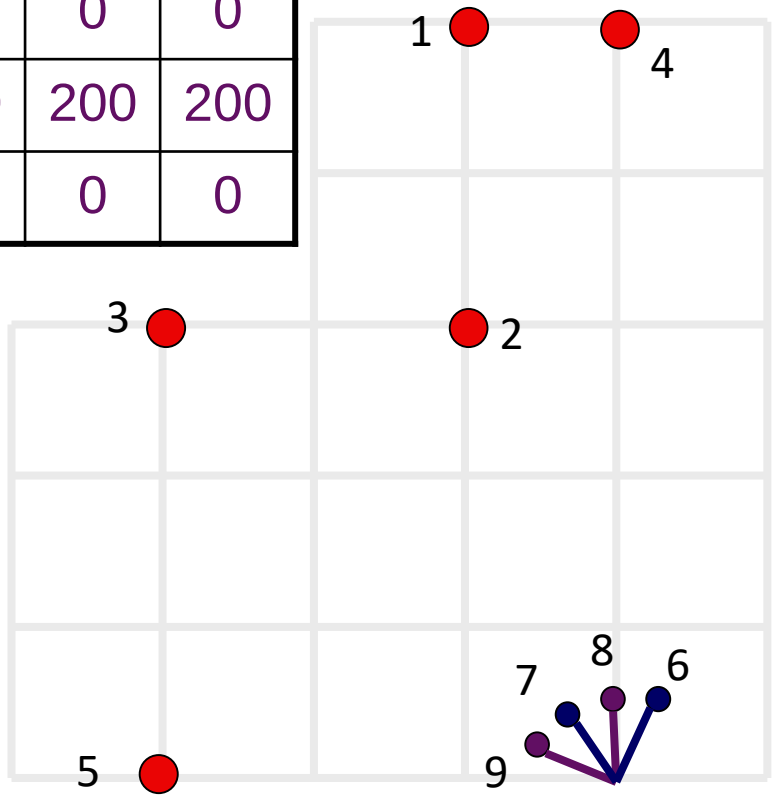
	V1	V2
$Q(uantity)_k$	30	30

Example

5 R(equests); 10 D(demands) @ each request; E(arliest) arrival time
L(atest) departure time; S(ervice) time, Q is vehicle capacity

	R1	R2	R3	R4	R5	R6	R7	R8	R9
D_i	10	10	10	10	10	0	0	0	0
E_i	20	20	20	20	20	0	0	0	0
L_i	150	65	150	150	150	200	200	200	200
S_i	10	10	10	10	10	0	0	0	0

	V1	V2
Q_k	30	30

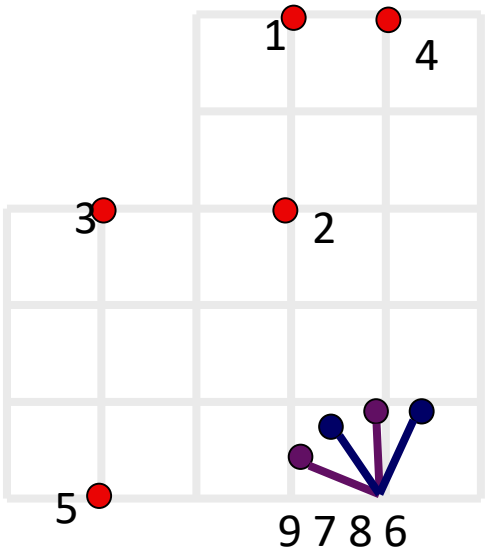


+ Request Compatibility

  Different Routes: R2, R4

Initial

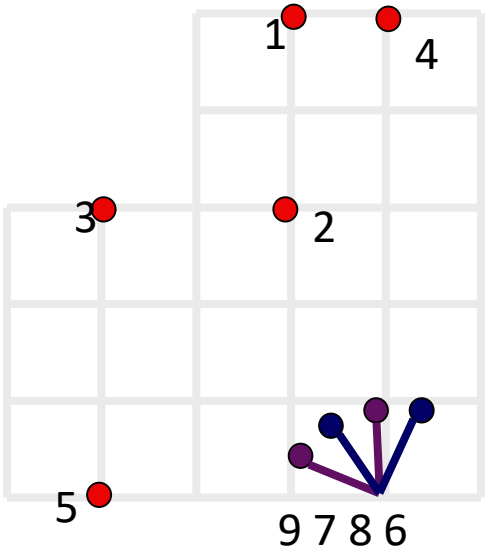
	R1	R2	R3	R4	R5	R6	R7	R8	R9
s_i	0,2-5, 8,9	0,1, 3-5,8,9	0-2 4,5,8,9	0-3,5, 8,9	0-4, 8,9	1-5	1-5	0	0
r_i	0,1,2	0,1,2	0,1,2	0,1,2	0,1,2	1	1	2	2
t_i	20-150	20-65	20-150	20-150	20-150	0	0	0-200	0-200
q_i	0-30	0-30	0-30	0-30	0-30	0	0	0-30	0-30



Initial

	R1	R2	R3	R4	R5	R6	R7	R8	R9
s_i	0,2-5, 8,9	0,1, 3-5,8,9	0-2 4,5,8,9	0-3,5, 8,9	0-4, 8,9	1-5	1-5	0	0
r_i	0,1,2	0,1,2	0,1,2	0,1,2	0,1,2	1	1	2	2
t_i	51-150	32-65	42-150	50-150	30-150	0	0	0-200	0-200
q_i	0-30	0-30	0-30	0-30	0-30	0	0	0-30	0-30

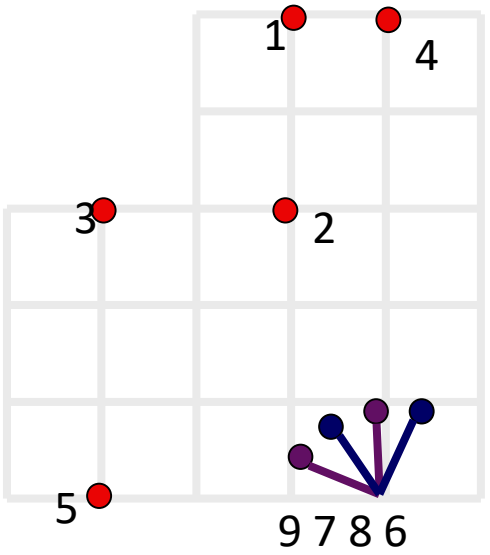
Initial propagations (arrive time)



Initial

	R1	R2	R3	R4	R5	R6	R7	R8	R9
s_i	0, 3-5,8,9	0,1, 3-5,8,9	0-2 4,5,8,9	0,1 3,5,8,9	0,1 3,4,8,9	1-5	1-5	0	0
r_i	0,1,2	0,1,2	0,1,2	0,1,2	0,1,2	1	1	2	2
t_i	51-150	32-65	42-150	50-150	30-150	0	0	0-200	0-200
q_i	0-30	0-30	0-30	0-30	0-30	0	0	0-30	0-30

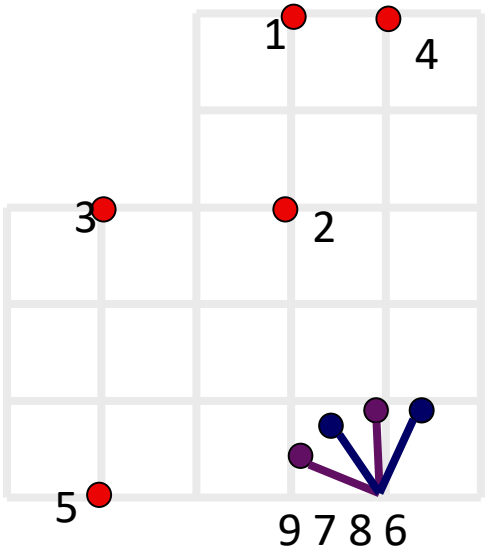
Initial propagations (time windows)



Initial

	R1	R2	R3	R4	R5	R6	R7	R8	R9
s_i	0, 3-5,8,9	0,1, 3,5,8,9	0-2 4,5,8,9	0,1 3,5,8,9	0,1 3,4,8,9	1-5	1-5	0	0
r_i	0,1,2	0,1,2	0,1,2	0,1,2	0,1,2	1	1	2	2
t_i	51-150	32-65	42-150	50-150	30-150	0	0	0-200	0-200
q_i	0-30	0-30	0-30	0-30	0-30	0	0	0-30	0-30

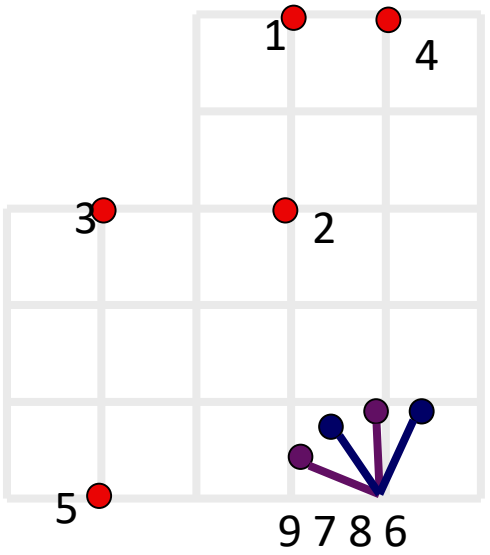
Initial propagations (compatibility constraint)



Initial

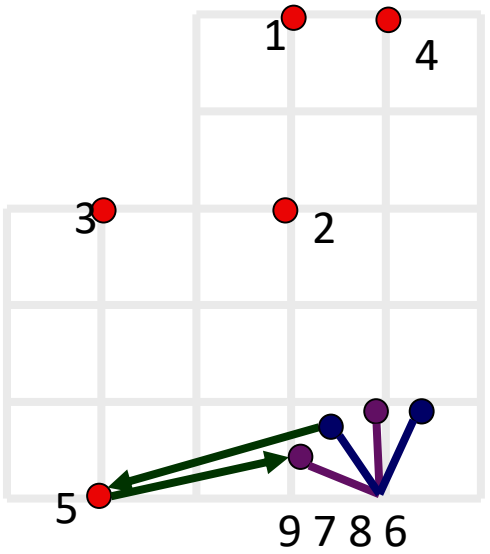
	R1	R2	R3	R4	R5	R6	R7	R8	R9
s_i	0, 3-5,8,9	0,1, 3,5,8,9	0-2 4,5,8,9	0,1 3,5,8,9	0,1 3,4,8,9	1-5	1-5	0	0
r_i	0,1,2	0,1,2	0,1,2	0,1,2	0,1,2	1	1	2	2
t_i	51-150	32-65	42-150	50-150	30-150	0	0	0-200	0-200
q_i	0-30	0-30	0-30	0-30	0-30	0	0	0-30	0-30

Fix-point.



Choose R5 after R7 (start V2)

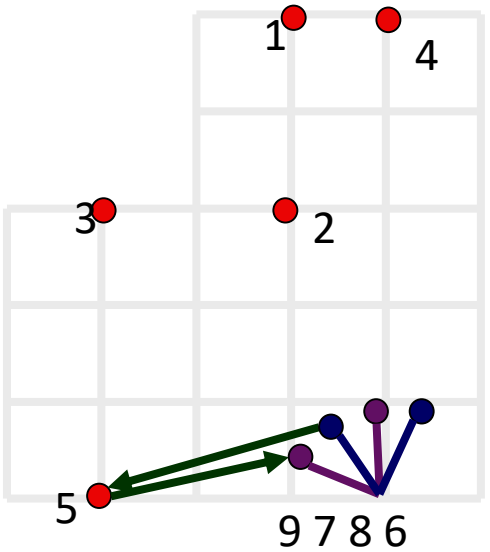
	R1	R2	R3	R4	R5	R6	R7	R8	R9
s_i	0, 3-5,8,9	0,1, 3,5,8,9	0-2 4,5,8,9	0,1 3,5,8,9	0,1 3,4,8,9	1-5	1-5	0	0
r_i	0,1,2	0,1,2	0,1,2	0,1,2	0,1,2	1	1	2	2
t_i	51-150	32-65	42-150	50-150	30-150	0	0	0-200	0-200
q_i	0-30	0-30	0-30	0-30	0-30	0	0	0-30	0-30



Choose R5 after R7 (start V2)

	R1	R2	R3	R4	R5	R6	R7	R8	R9
s_i	0, 3-5,8,9	0,1, 3,5,8,9	0-2 4,5,8,9	0,1 3,5,8,9	0,1 3,4,8,9	1-4	1-5	0	0
r_i	0,1,2	0,1,2	0,1,2	0,1,2	0,1,2	1	1	2	2
t_i	51-150	32-65	42-150	50-150	30-150	0	0	0-200	0-200
q_i	0-30	0-30	0-30	0-30	0-30	0	0	0-30	0-30

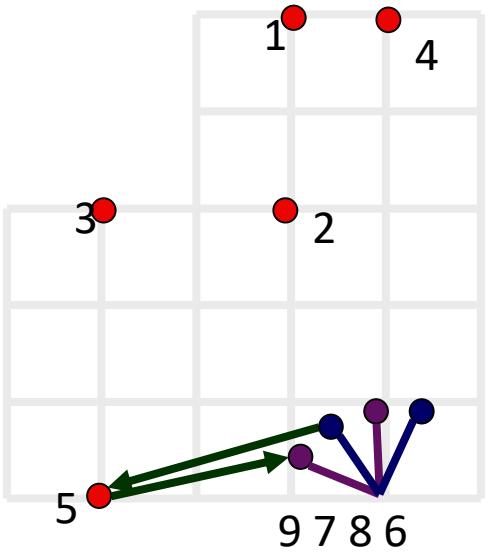
Propagate successor implications



Choose R5 after R7 (start V2)

	R1	R2	R3	R4	R5	R6	R7	R8	R9
s_i	0, 3-5,8,9	0,1, 3,5,8,9	0-2 4,5,8,9	0,1 3,5,8,9	0,1 3,4,9	1-4	1-5	0	0
r_i	0,1,2	0,1,2	0,1,2	0,1,2	0,1,2	1	1	2	2
t_i	51-150	32-65	42-150	50-150	30-150	0	0	0-200	60-200
q_i	0-30	0-30	0-30	0-30	10-30	0	0	0-30	10-30

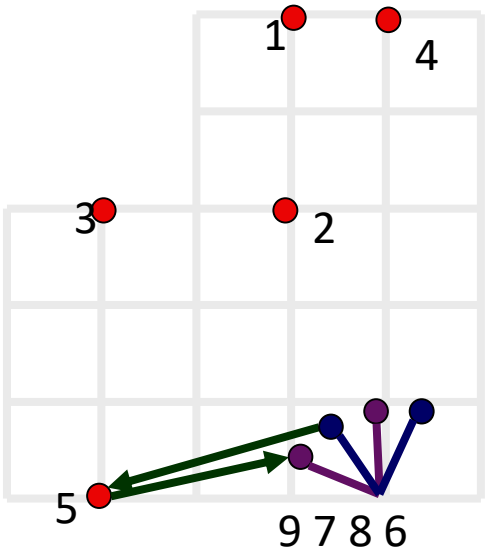
Propagate changes to time and load



Choose R5 after R7 (start V2)

	R1	R2	R3	R4	R5	R6	R7	R8	R9
s_i	0, 3-5,8,9	0,1, 3,5,8,9	0-2 4,5,8,9	0,1 3,5,8,9	0,1 3,4,9	1-4	1-5	0	0
r_i	0,1,2	0,1,2	0,1,2	0,1,2	~~0,1~~,2	1	1	2	2
t_i	51-150	32-65	42-150	50-150	30-150	0	0	0-200	60-200
q_i	0-30	0-30	0-30	0-30	10-30	0	0	0-30	10-30

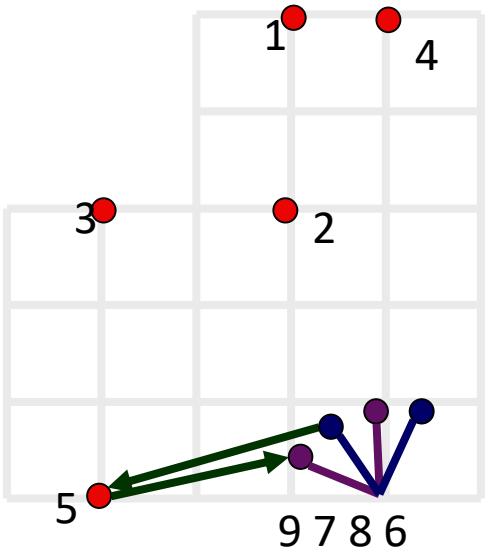
Bind route var



Choose R5 after R7 (start V2)

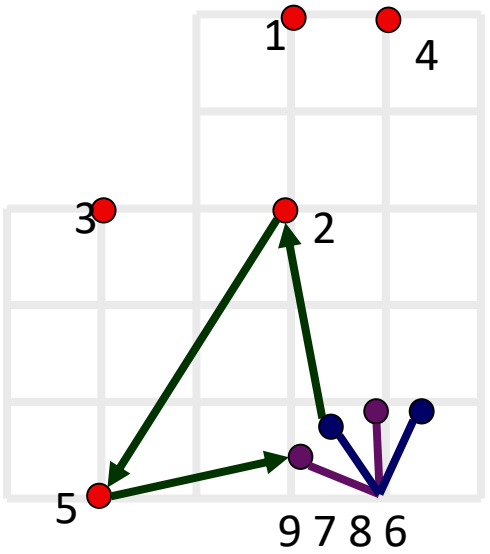
	R1	R2	R3	R4	R5	R6	R7	R8	R9
s_i	0, 3-5,8,9	0,1, 3,5,8,9	0-2 4,5,8,9	0,1 3,5,8,9	0,1 3,4,9	1-4	1-5	0	0
r_i	0,1,2	0,1,2	0,1,2	0,1,2	2	1	1	2	2
t_i	51-150	32-65	42-150	50-150	30-150	0	0	0-200	60-200
q_i	0-30	0-30	0-30	0-30	10-30	0	0	0-30	10-30

Fix-point



Choose R2 after R7 (start V2)

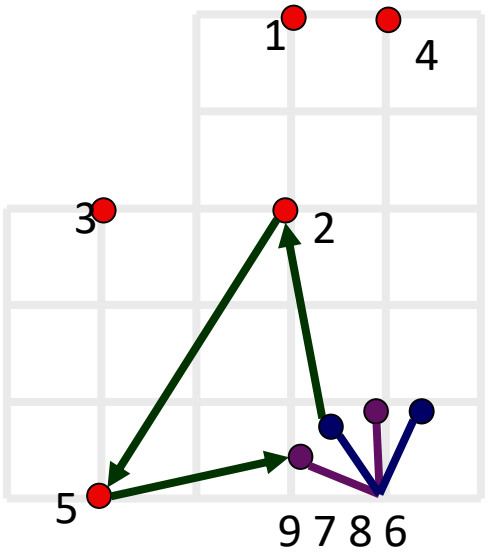
	R1	R2	R3	R4	R5	R6	R7	R8	R9
s_i	0, 3-5,8,9	0,1, 3,5,8,9	0-2 4,5,8,9	0,1 3,5,8,9	0,1 3,4,9	1-4	1-5	0	0
r_i	0,1,2	0,1,2	0,1,2	0,1,2	2	1	1	2	2
t_i	51-150	32-65	42-150	50-150	30-150	0	0	0-200	60-200
q_i	0-30	0-30	0-30	0-30	10-30	0	0	0-30	10-30



Choose R2 after R7 (start V2)

	R1	R2	R3	R4	R5	R6	R7	R8	R9
s_i	0, 3-5,8,9	1,3,5	0-2 4,5,8,9	0,1 3,5,8,9	1,3,4, 9	1,3,4	1-4	0	0
r_i	0,1,2	0,1,2	0,1,2	0,1,2	2	1	1	2	2
t_i	51-150	32-65	42-150	50-150	30-150	0	0	0-200	60-200
q_i	0-30	0-30	0-30	0-30	10-30	0	0	0-30	10-30

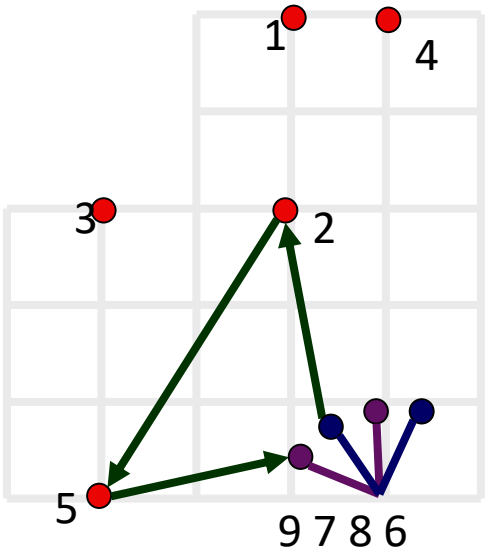
Propagate successor implications



Choose R2 after R7 (start V2)

	R1	R2	R3	R4	R5	R6	R7	R8	R9
s_i	0, 3-5,8,9	1,3,5	0-2 4,5,8,9	0,1 3,5,8,9	1,3,4, 9	1,3,4	1-4	0	0
r_i	0,1,2	0,1,2	0,1,2	0,1,2	2	1	1	2	2
t_i	51-150	32-65	42-150	50-150	78-150	0	0	0-200	118-200
q_i	0-30	10-30	0-30	0-30	20-30	0	0	0-30	20-30

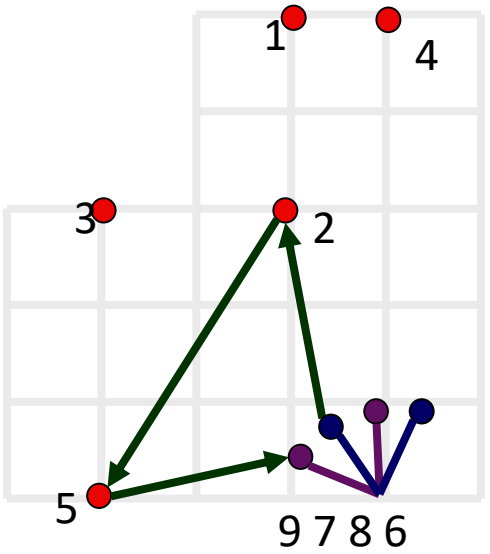
Update load and time



Choose R2 after R7 (start V2)

	R1	R2	R3	R4	R5	R6	R7	R8	R9
s_i	0, 3-5,8,9	1,3,5	0-2 4,5,8,9	0,1 3,5,8,9	1,3,4, 9	1,3,4	1-4	0	0
r_i	0,1,2	~~0,1~~,2	0,1,2	0,1,2	2	1	1	2	2
t_i	51-150	32-65	42-150	50-150	78-150	0	0	0-200	118-200
q_i	0-30	10-30	0-30	0-30	20-30	0	0	0-30	20-30

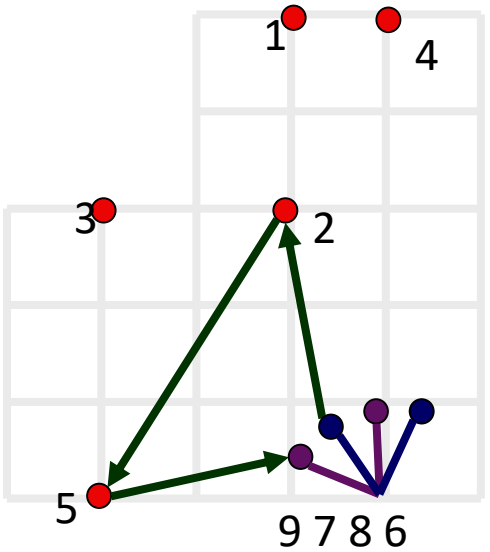
Bind route var



Choose R2 after R7 (start V2)

	R1	R2	R3	R4	R5	R6	R7	R8	R9
s_i	0, 3-5,8,9	1,3,5	0-2 4,5,8,9	0,1,3, 8	1,3,~~4~~ 9	1,3,4	1-3	0	0
r_i	0,1,2	2	0,1,2	0,1,~~2~~	2	1	1	2	2
t_i	51-150	32-65	42-150	50-150	78-150	0	0	0-200	118-200
q_i	0-30	10-30	0-30	0-30	20-30	0	0	0-30	20-30

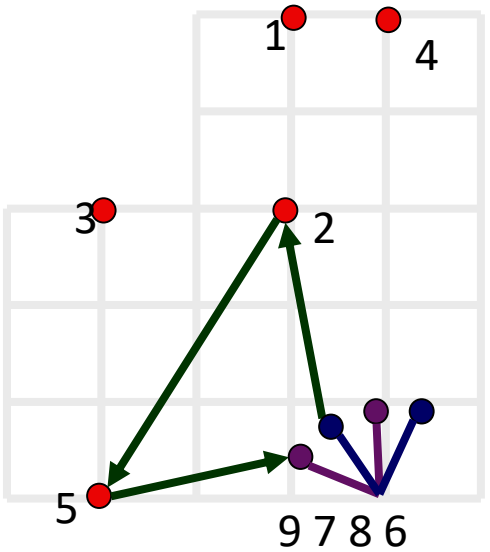
Propagate request compatibility constraint



Choose R2 after R7 (start V2)

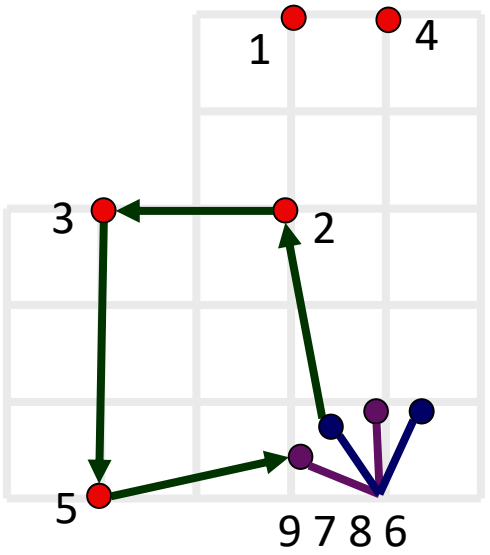
	R1	R2	R3	R4	R5	R6	R7	R8	R9
s_i	0, 3-5,8,9	1,3,5	0-2 4,5,8,9	0,1,3, 8	1,3,9	1,3,4	1-3	0	0
r_i	0,1,2	2	0,1,2	0,1	2	1	1	2	2
t_i	51-150	32-65	42-150	50-150	78-150	0	0	0-200	118-200
q_i	0-30	10-30	0-30	0-30	20-30	0	0	0-30	20-30

Fixpoint



Choose R3 after R2

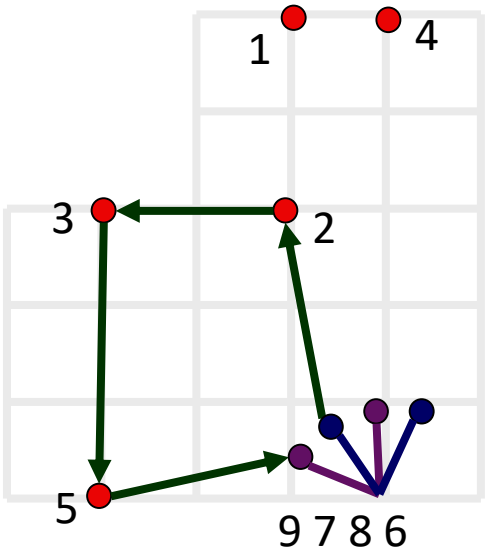
	R1	R2	R3	R4	R5	R6	R7	R8	R9
s_i	0, 3-5,8,9	1,3,5	0-2 4,5,8,9	0,1,3, 8	1,3,9	1,3,4	1-3	0	0
r_i	0,1,2	2	0,1,2	0,1	2	1	1	2	2
t_i	51-150	32-65	42-150	50-150	78-150	0	0	0-200	118-200
q_i	0-30	10-30	0-30	0-30	20-30	0	0	0-30	20-30



Choose R3 after R2

	R1	R2	R3	R4	R5	R6	R7	R8	R9
s_i	0, 3-5,8,9	1,3,5	1,4,5	0,1,3, 8	1,3,9	1,3,4	1,2	0	0
r_i	0,1,2	2	0,1,2	0,1	2	1	1	2	2
t_i	51-150	32-65	42-150	50-150	78-150	0	0	0-200	118-200
q_i	0-30	10-30	0-30	0-30	20-30	0	0	0-30	20-30

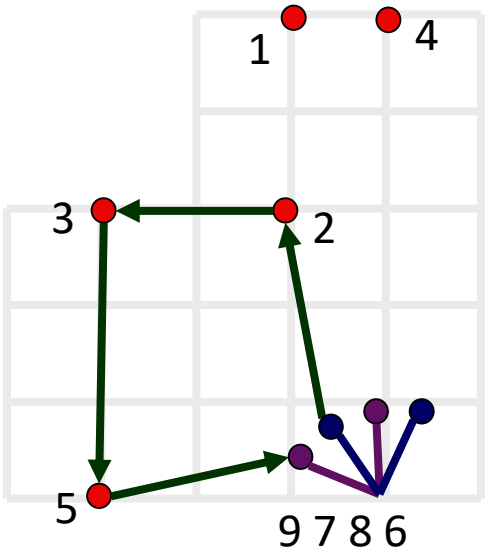
Propagate successor implications



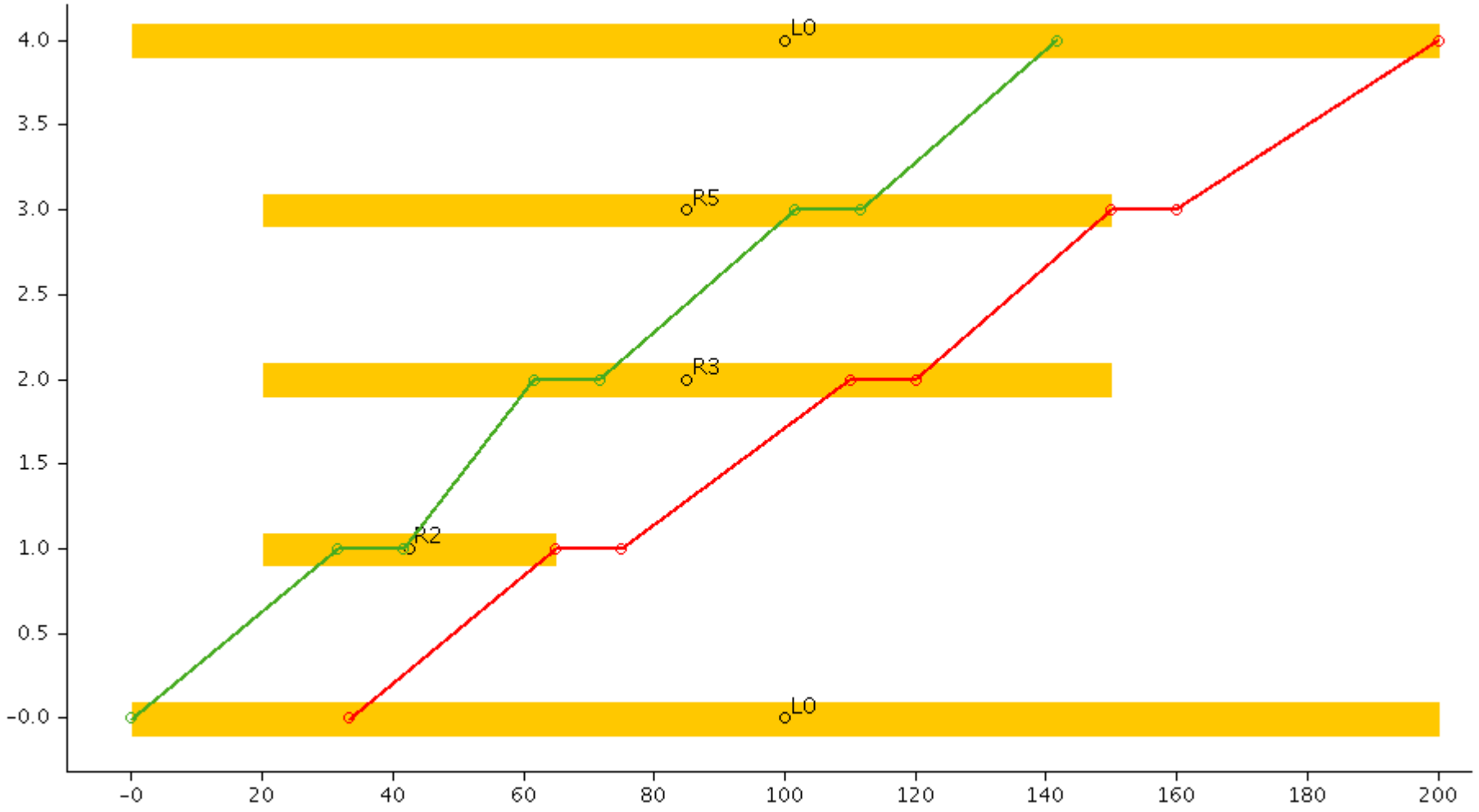
Choose R3 after R2

	R1	R2	R3	R4	R5	R6	R7	R8	R9
s_i	0, 3-5,8,9	1,3	1,4,5	0,1,3, 8	1,9	1,4	1,2	0	0
r_i	0,1,2	2	0,1,2	0,1	2	1	1	2	2
t_i	51-150	32-65	62-110	50-150	102-150	0	0	0-200	142-200
q_i	0-30	10-30	20-30	0-30	30	0	0	0-30	30

Update time and load



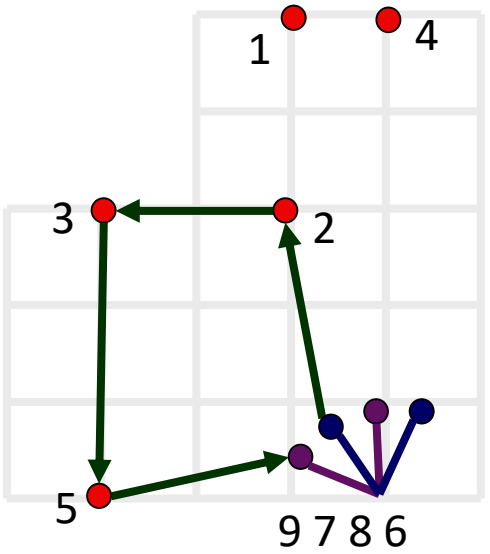
V1 schedule


4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
-0.0
-0
20
40
60
80
100
120
140
160
180
200
L0
R5
R3
R2
L0

Choose R3 after R2

	R1	R2	R3	R4	R5	R6	R7	R8	R9
s_i	0, 3-5,8,9	1,3	1,4,5	0,1,3, 8	1,9	1,4	1,2	0	0
r_i	0,1,2	2	~~0,1,2~~	0,1	2	1	1	2	2
t_i	51-150	32-65	62-110	50-150	102-150	0	0	0-200	142-200
q_i	0-30	10-30	20-30	0-30	30	0	0	0-30	30

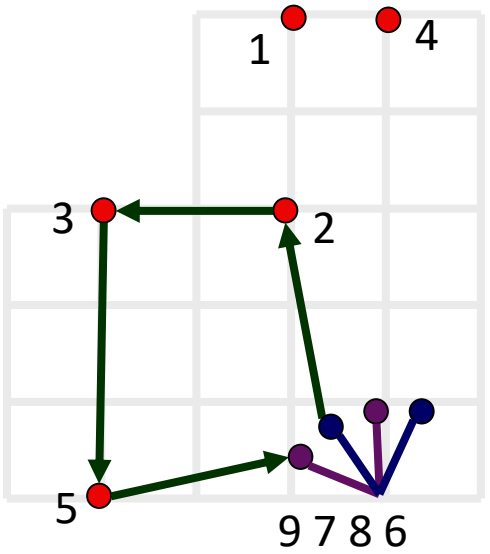
Bind route var



Choose R3 after R2

	R1	R2	R3	R4	R5	R6	R7	R8	R9
s_i	0, 3-5,8,9	1,3	~~1,4,~~5	0,1,~~3~~ 8	1,9	1,4	1,2	0	0
r_i	0,1,2	2	2	0,1	2	1	1	2	2
t_i	51-150	32-65	62-110	50-150	102-150	0	0	0-200	142-200
q_i	0-30	10-30	20-30	0-30	30	0	0	0-30	30

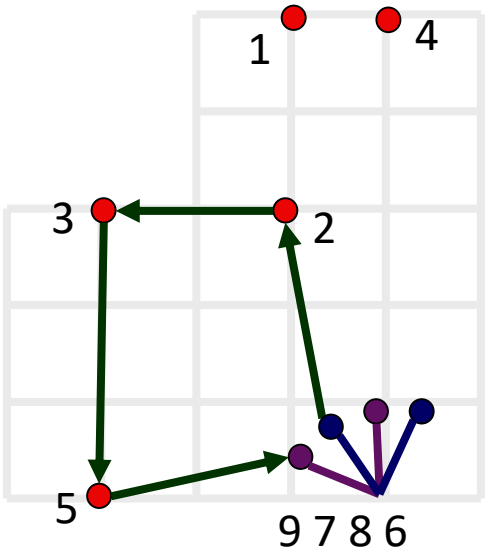
Propagate request incompatibility constraint



Choose R3 after R2

	R1	R2	R3	R4	R5	R6	R7	R8	R9
s_i	0,4,8	~~1~~,3	5	0,1,8	~~1~~9	1,4	~~1~~2	0	0
r_i	0,1,~~2~~	2	2	0,1	2	1	1	2	2
t_i	51-150	32-65	62-110	50-150	102-150	0	0	0-200	142-200
q_i	0-30	10-30	20-30	0-30	30	0	0	0-30	30

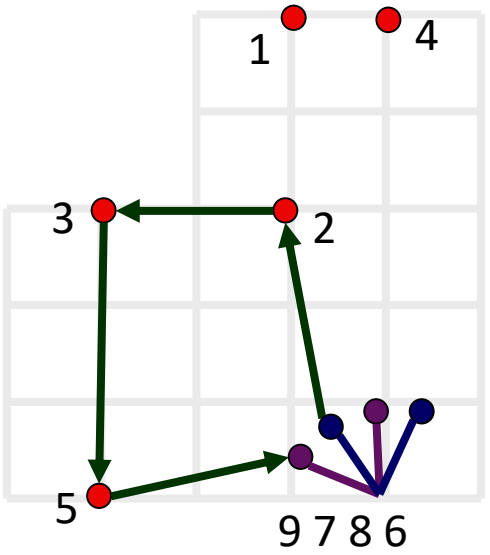
Propagate effects of full load



Choose R3 after R2

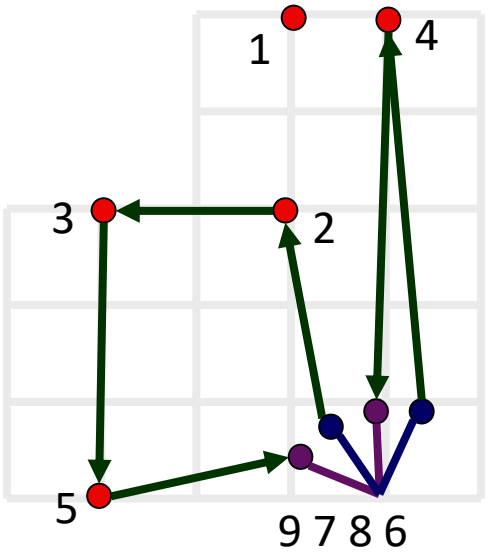
	R1	R2	R3	R4	R5	R6	R7	R8	R9
s_i	0,4,8	3	5	0,1,8	9	1,4	2	0	0
r_i	0,1	2	2	0,1	2	1	1	2	2
t_i	51-150	32-65	62-110	50-150	102-150	0	0	0-200	142-200
q_i	0-30	10-30	20-30	0-30	30	0	0	0-30	30

Fix-point



Choose R4 after R6 (start V1)

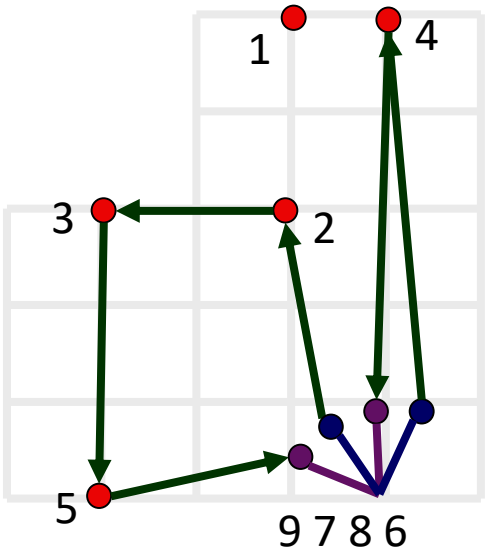
	R1	R2	R3	R4	R5	R6	R7	R8	R9
s_i	0,4,8	3	5	0,1,8	9	1,4	2	0	0
r_i	0,1	2	2	0,1	2	1	1	2	2
t_i	51-150	32-65	62-110	50-150	102-150	0	0	0-200	142-200
q_i	0-30	10-30	20-30	0-30	30	0	0	0-30	30



Choose R4 after R6 (start V1)

	R1	R2	R3	R4	R5	R6	R7	R8	R9
s_i	0,4,8	3	5	~~0~~,1,8	9	1,4	2	0	0
r_i	0,1	2	2	0,1	2	1	1	2	2
t_i	51-150	32-65	62-110	50-150	102-150	0	0	0-200	142-200
q_i	0-30	10-30	20-30	0-30	30	0	0	0-30	30

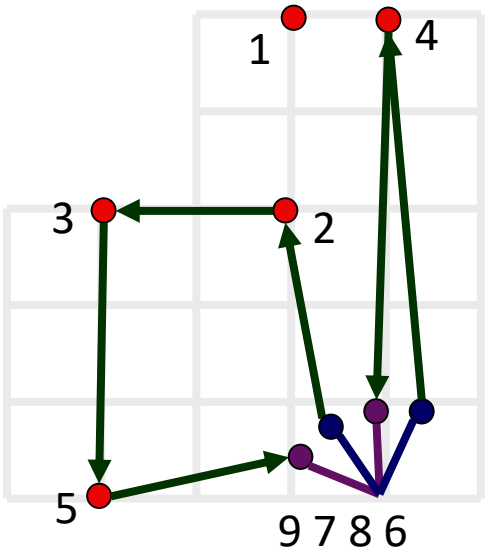
Propagate successor implications



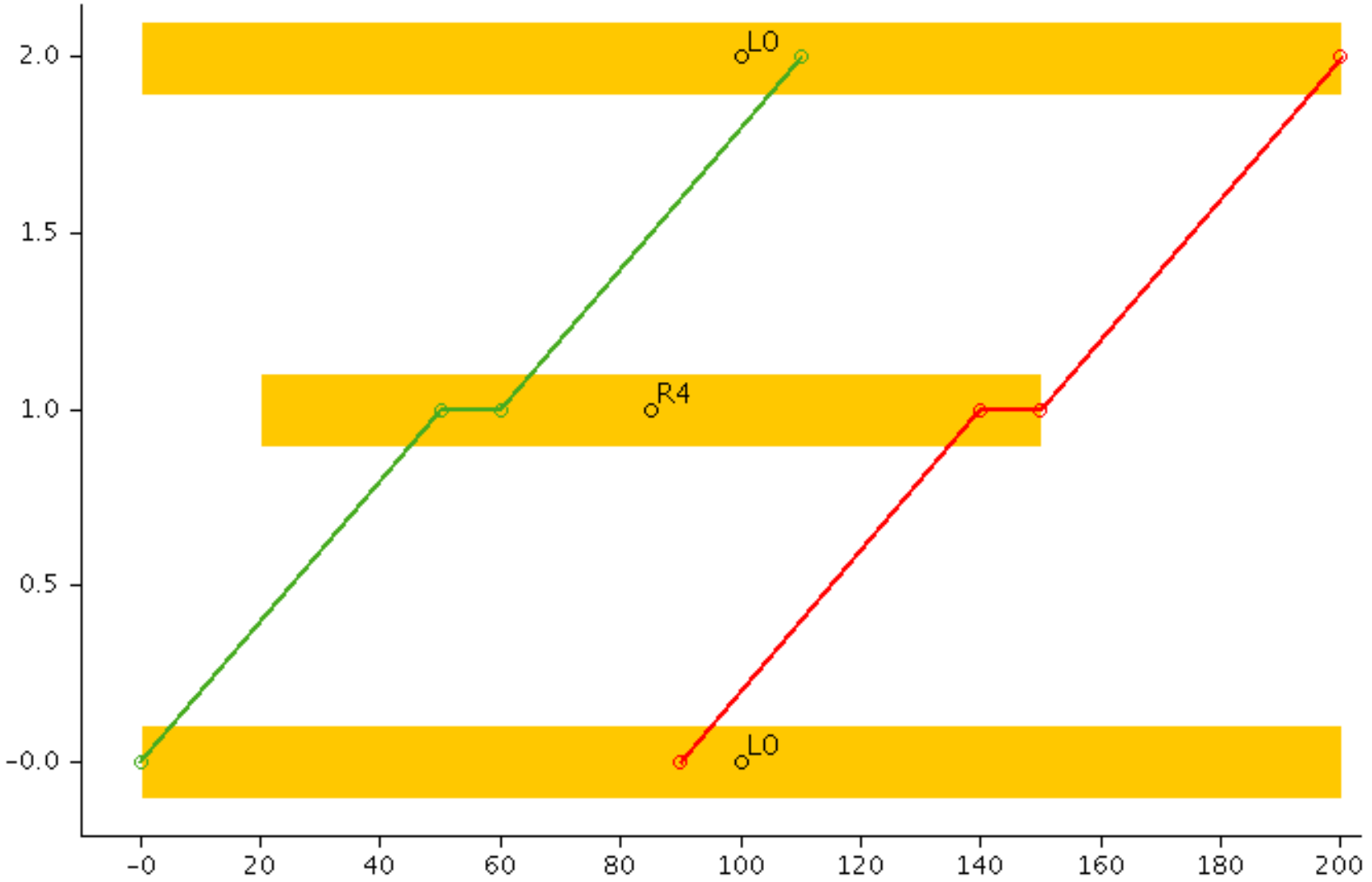
Choose R4 after R6 (start V1)

	R1	R2	R3	R4	R5	R6	R7	R8	R9
s_i	0,4,8	3	5	1,8	9	1,4	2	0	0
r_i	0,1	2	2	0,1	2	1	1	2	2
t_i	51-150	32-65	62-110	50-140	102-150	0	0	110-200	142-200
q_i	0-30	10-30	20-30	10-30	30	0	0	10-30	30

Update time and load



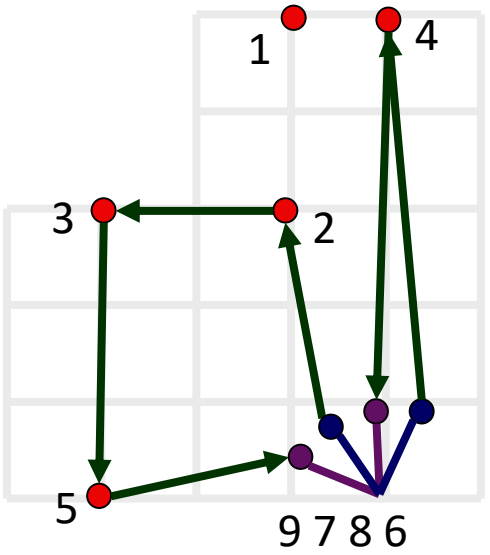
Choose R4 after R6 (start V1)



Choose R4 after R6 (start V1)

	R1	R2	R3	R4	R5	R6	R7	R8	R9
s_i	0,4,8	3	5	1,8	9	1,4	2	0	0
r_i	0,1	2	2	~~0~~1	2	1	1	2	2
t_i	51-150	32-65	62-110	50-140	102-150	0	0	110-200	142-200
q_i	0-30	10-30	20-30	10-30	30	0	0	10-30	30

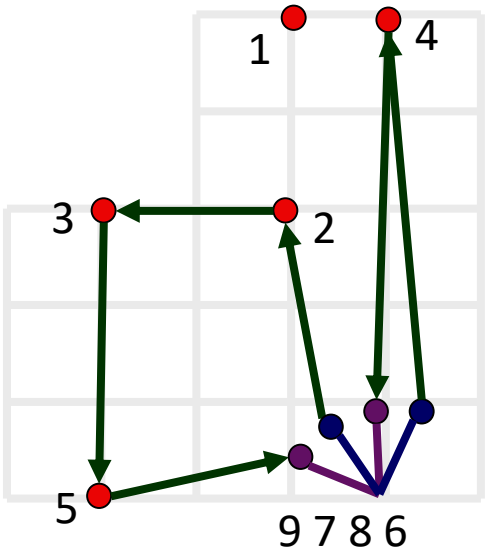
Bind route var



Choose R4 after R6 (start V1)

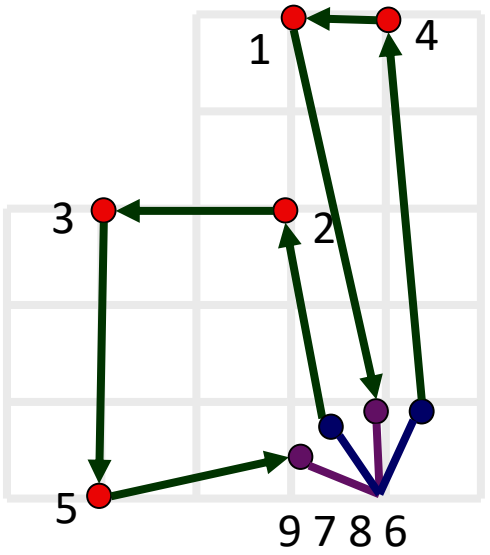
	R1	R2	R3	R4	R5	R6	R7	R8	R9
s_i	0,4,8	3	5	1,8	9	1,4	2	0	0
r_i	0,1	2	2	1	2	1	1	2	2
t_i	51-150	32-65	62-110	50-140	102-150	0	0	110-200	142-200
q_i	0-30	10-30	20-30	10-30	30	0	0	10-30	30

Fix-point



Choose R1 after R4

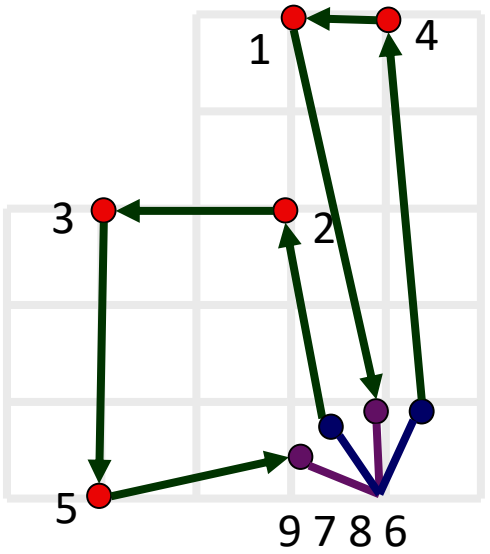
	R1	R2	R3	R4	R5	R6	R7	R8	R9
s_i	0,4,8	3	5	1,8	9	1,4	2	0	0
r_i	0,1	2	2	1	2	1	1	2	2
t_i	51-150	32-65	62-110	50-140	102-150	0	0	110-200	142-200
q_i	0-30	10-30	20-30	10-30	30	0	0	10-30	30



Choose R1 after R4

	R1	R2	R3	R4	R5	R6	R7	R8	R9
s_i	~~0,4,8~~	3	5	~~1,8~~	9	~~1,4~~	2	0	0
r_i	0,1	2	2	1	2	1	1	2	2
t_i	51-150	32-65	62-110	50-140	102-150	0	0	110-200	142-200
q_i	0-30	10-30	20-30	10-30	30	0	0	10-30	30

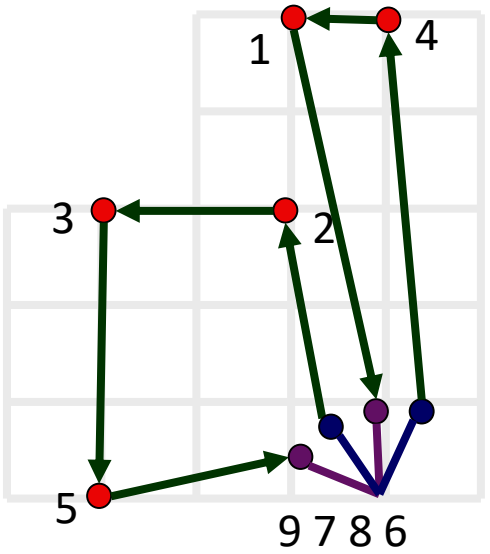
Propagate successor implications



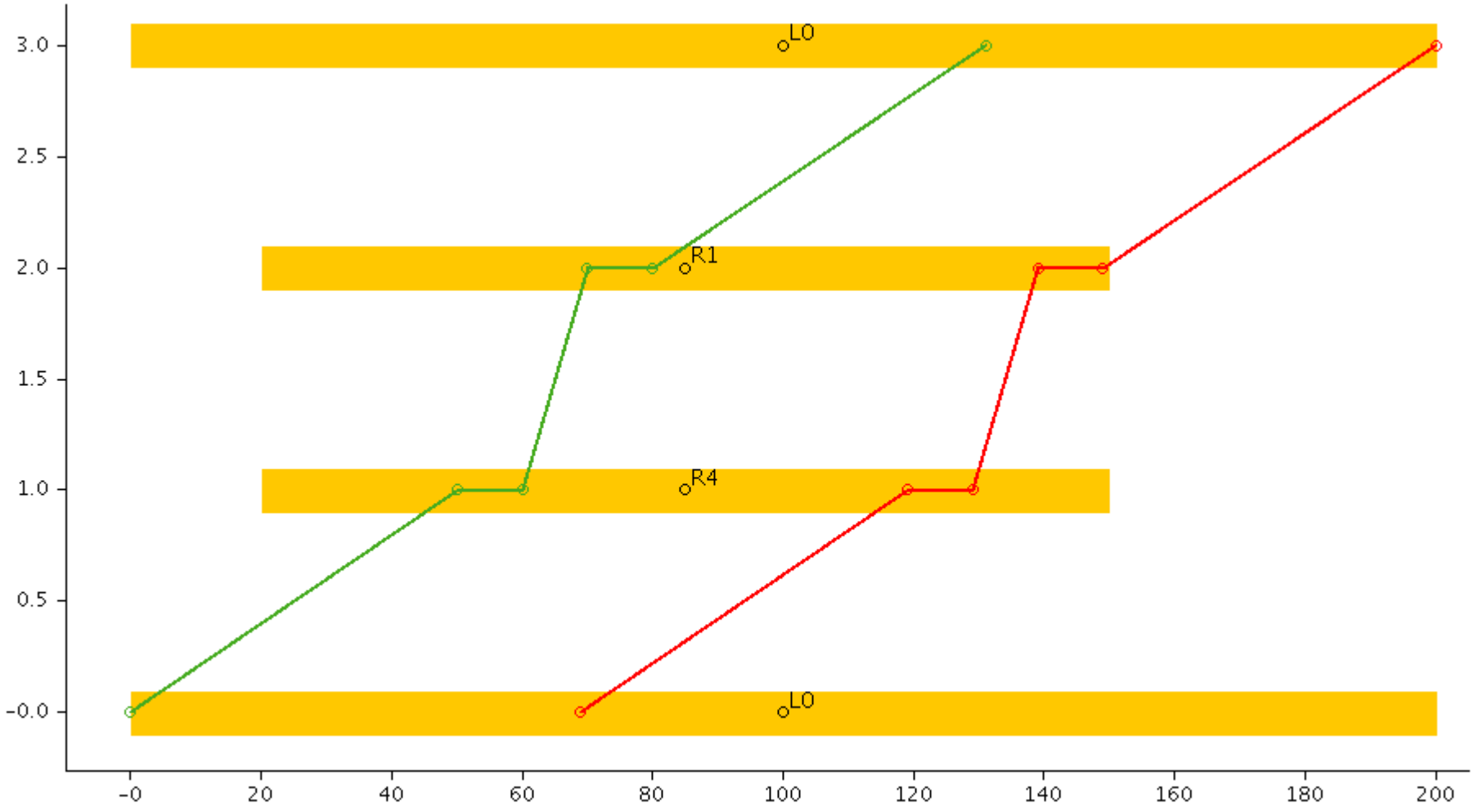
Choose R1 after R4

	R1	R2	R3	R4	R5	R6	R7	R8	R9
s_i	8	3	5	1	9	4	2	0	0
r_i	~~0~~ 1	2	2	1	2	1	1	2	2
t_i	70-139	32-65	62-110	50-119	102-150	0	0	131-200	142-200
q_i	20-30	10-30	20-30	10-30	30	0	0	20-30	30

Update load and time, and route var



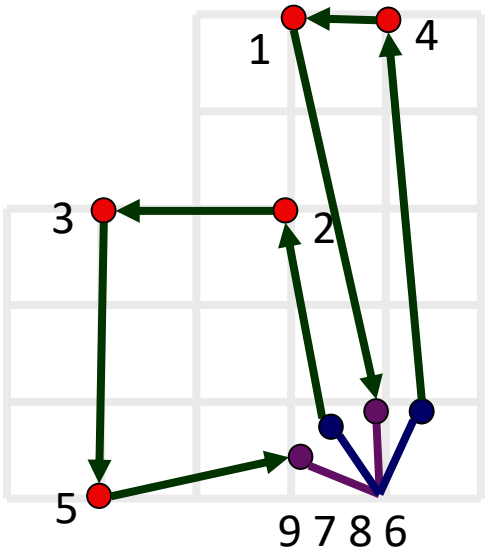
Choose R1 after R4



Choose R1 after R4

	R1	R2	R3	R4	R5	R6	R7	R8	R9
s_i	8	3	5	1	9	4	2	0	0
r_i	1	2	2	1	2	1	1	2	2
t_i	70-139	32-65	62-110	50-119	102-150	0	0	131-200	142-200
q_i	20-30	10-30	20-30	10-30	30	0	0	20-30	30

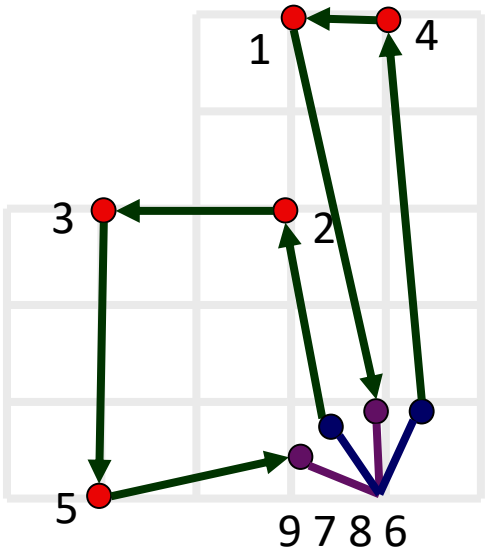
Fix-point



Choose R1 after R4

	R1	R2	R3	R4	R5	R6	R7	R8	R9
s_i	8	3	5	1	9	4	2	0	0
r_i	1	2	2	1	2	1	1	2	2
t_i	70	32	62	50	102	0	0	131	142
q_i	20	10	20	10	30	0	0	20	30

FINISHED! Bind remaining vars to min val



Local Search

- Can apply std local search methods VRPs
  - k-opt
  - Or-opt
  - exchange
  - …
- Step from solution to solution, so
- CP is only used as rule-checker
  - Little use of propagation

- But LNS is repeated insertion
  - Makes maximum use of propagation

Large Neighbourhood Search revisited

Large Neighbourhood Search

Destroy & Re-create

- Destroy part of the solution
  - Remove visits from the solution
- Re-create solution
  - Use insert method to re-insert customers
  - Different insert methods lead to new (better?) solutions
- If the solution is better, keep it
- Repeat

Large Neighbourhood Search

Destroy part of the solution (*Select* method)

In CP terms, this means:

- Relax some variable assignments

In CP-VRP terms, this means

- Relax some *routeOf* and *successor* assignments

Large Neighbourhood Search

Re-create solution

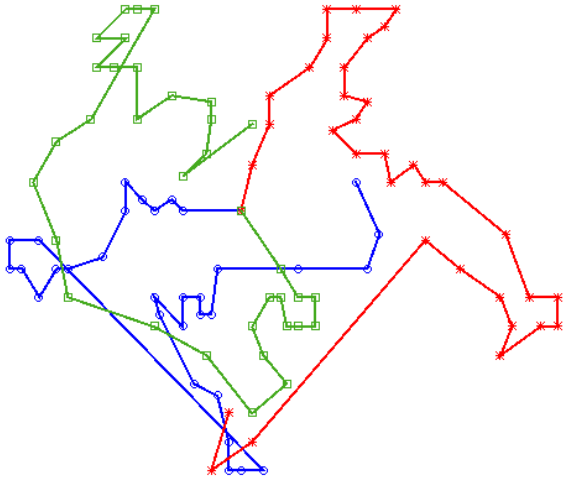
- Use insert methods
- Uses full power of CP propagations

Advanced techniques – Re-create solution

Adaptive Decomposition

Decompose problem

- Only consider 2-3 routes
- Smaller problem is much easier to solve



Advanced – Re-create solution

Adaptive Decomposition

Decompose problem

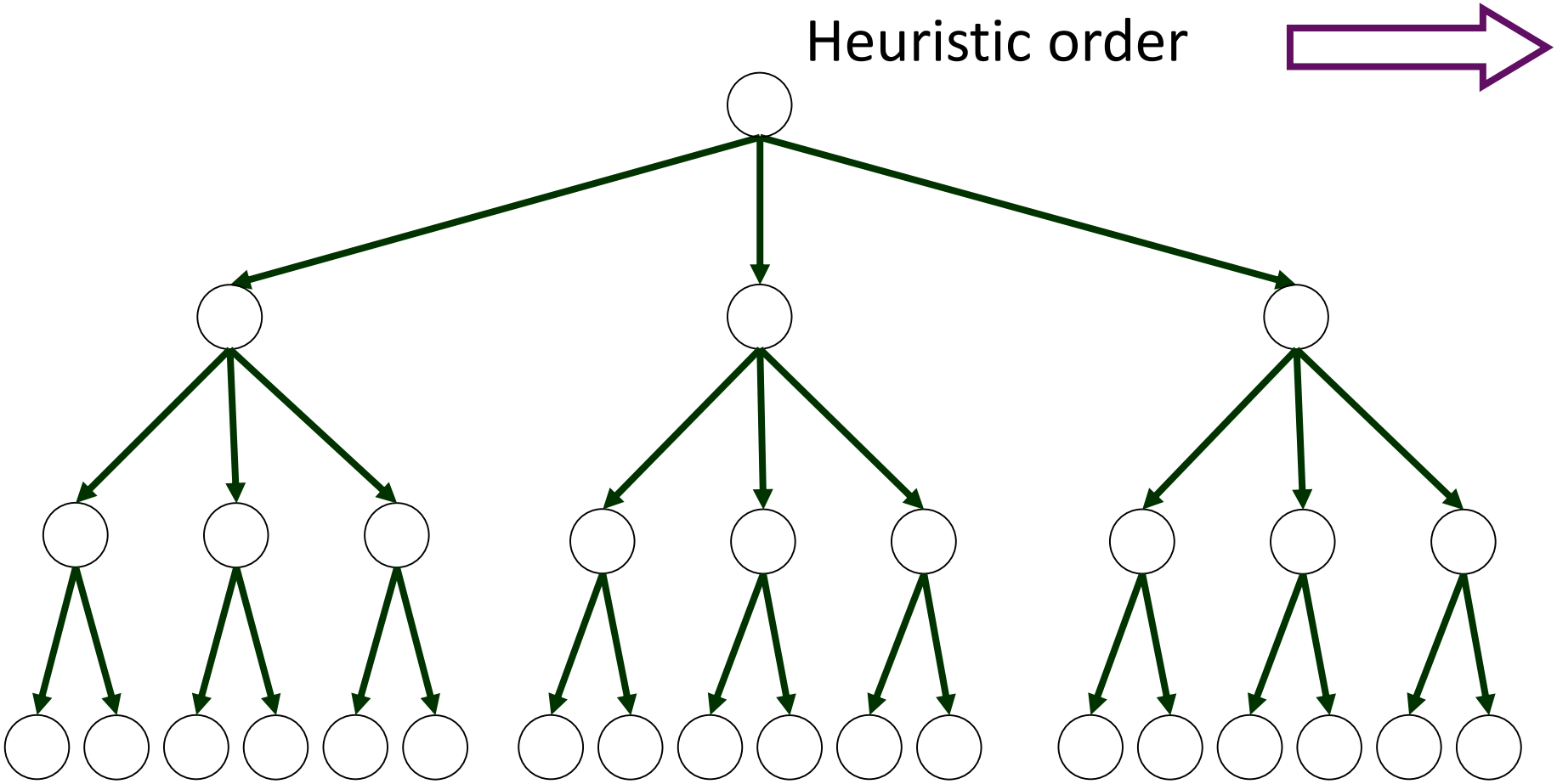
- Only consider 2-3 routes
- Smaller problem is much easier to solve

Adaptive

- Decompose in different ways
- Use problem features to determine decomposition
- Dependency graph analysis
  - [Acyclicity Bounds] Abdulaziz, Gretton and Norrish, ICAPS 2017
  - [Objective/Goal Decomposition] Rintanen and Gretton, IJCAI 2013
  - [HOL4 Proofs] Abdulaziz, Gretton and Norrish, JAR 2018
- Symmetry:
  - [Quotient Problem] Abdulaziz, Gretton and Norrish, IJCAI 2015

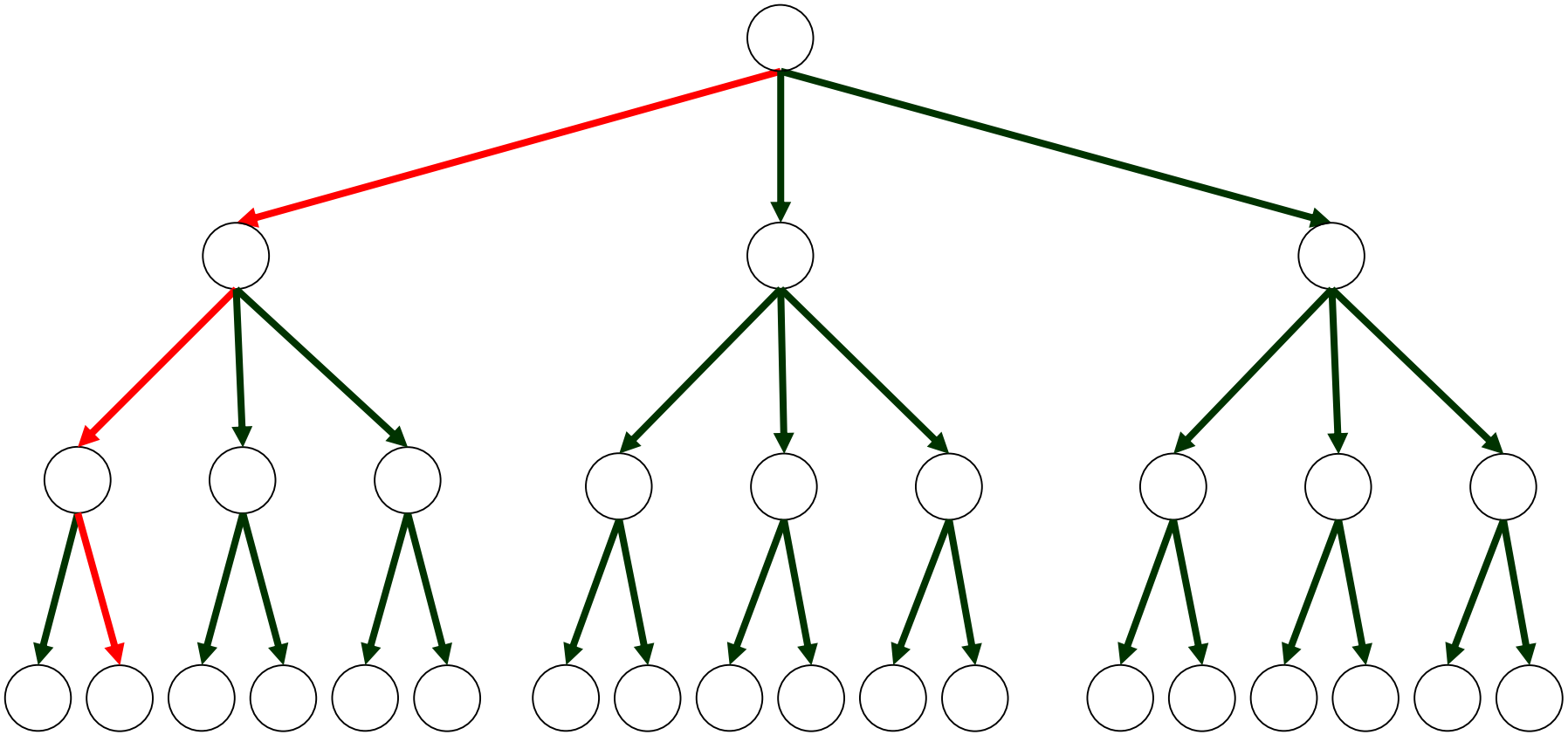
Advanced techniques – Re-create solution

Limited Discrepancy Search



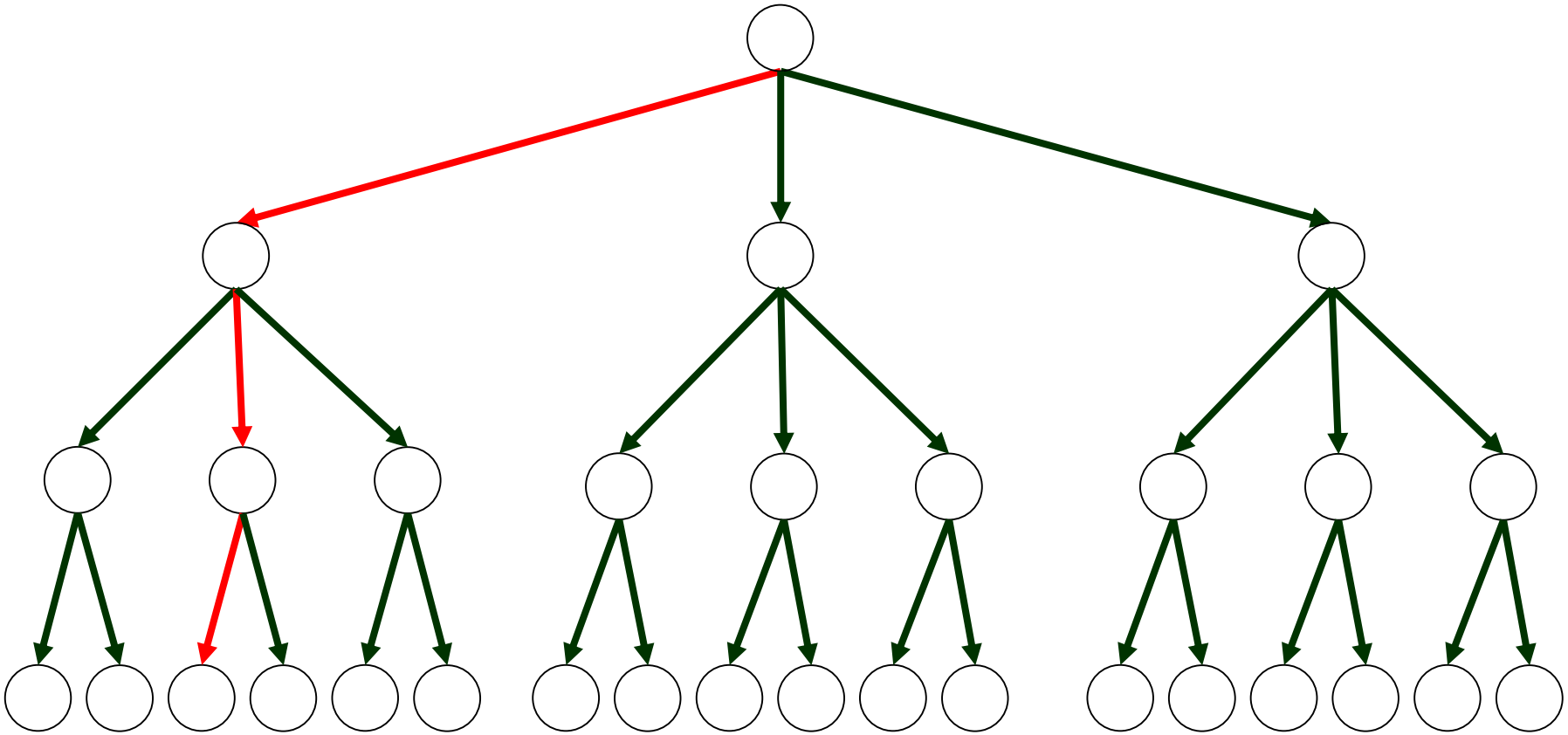
Advanced techniques

Limited Discrepancy Search



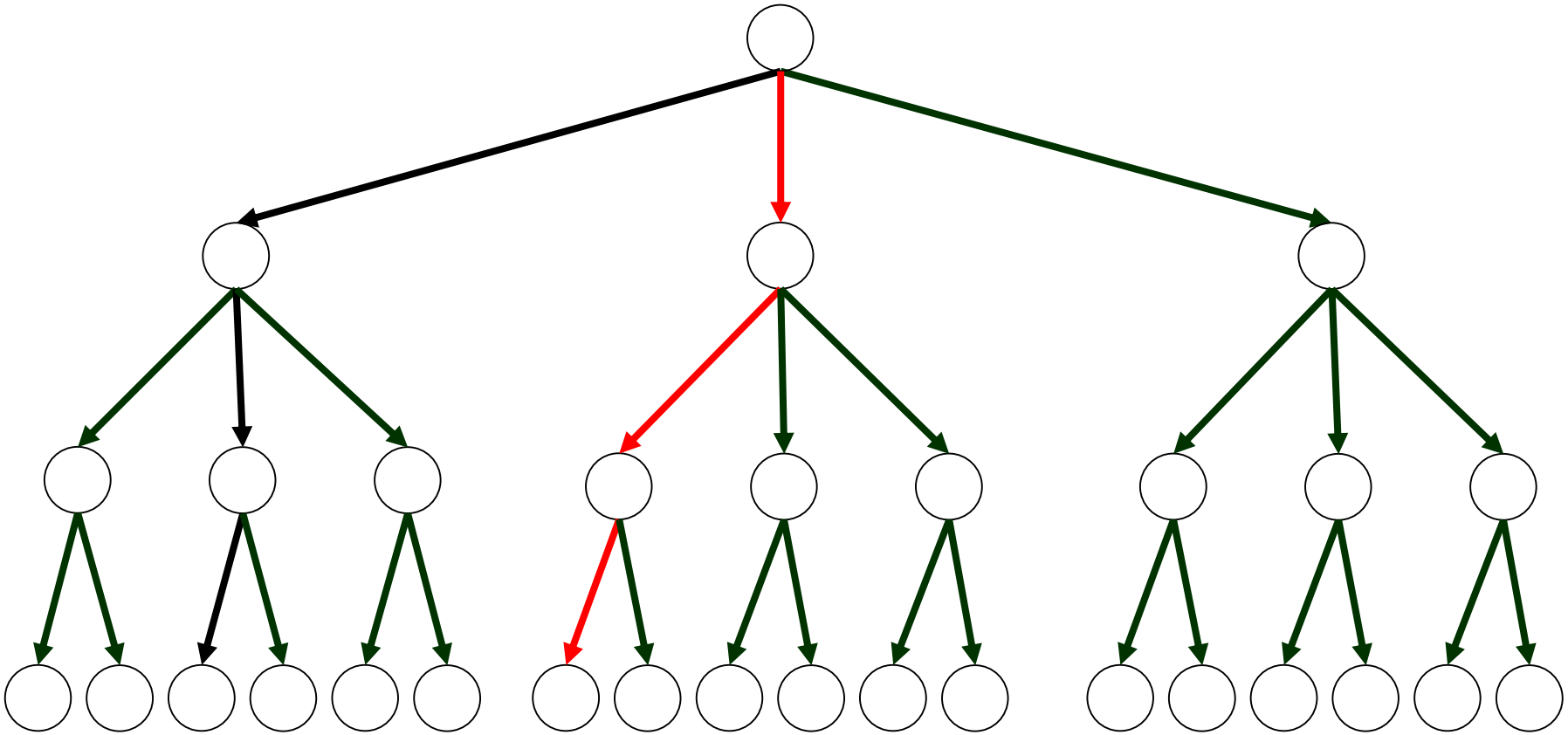
Advanced techniques

Limited Discrepancy Search



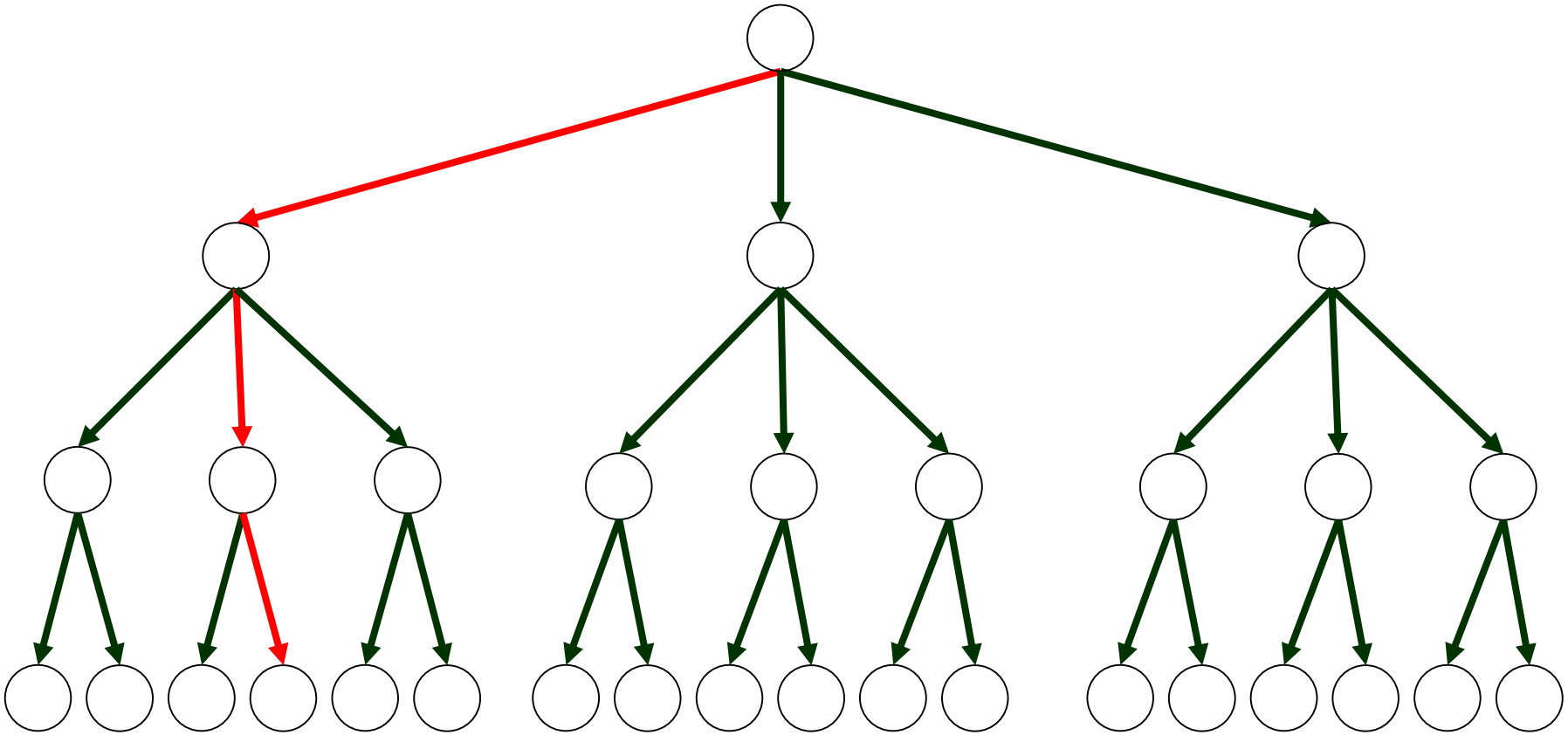
Advanced techniques

Limited Discrepancy Search



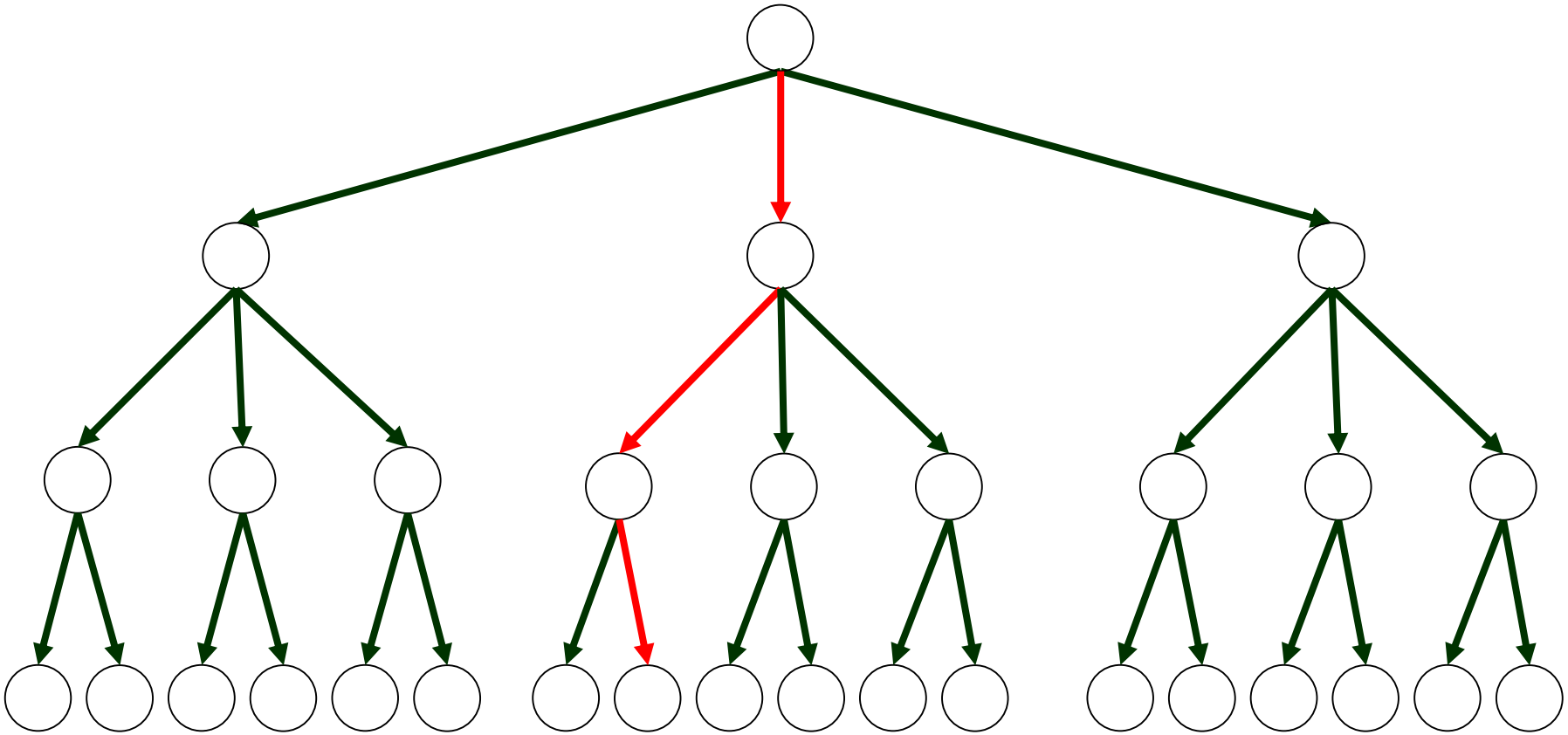
Advanced techniques

2-LDS



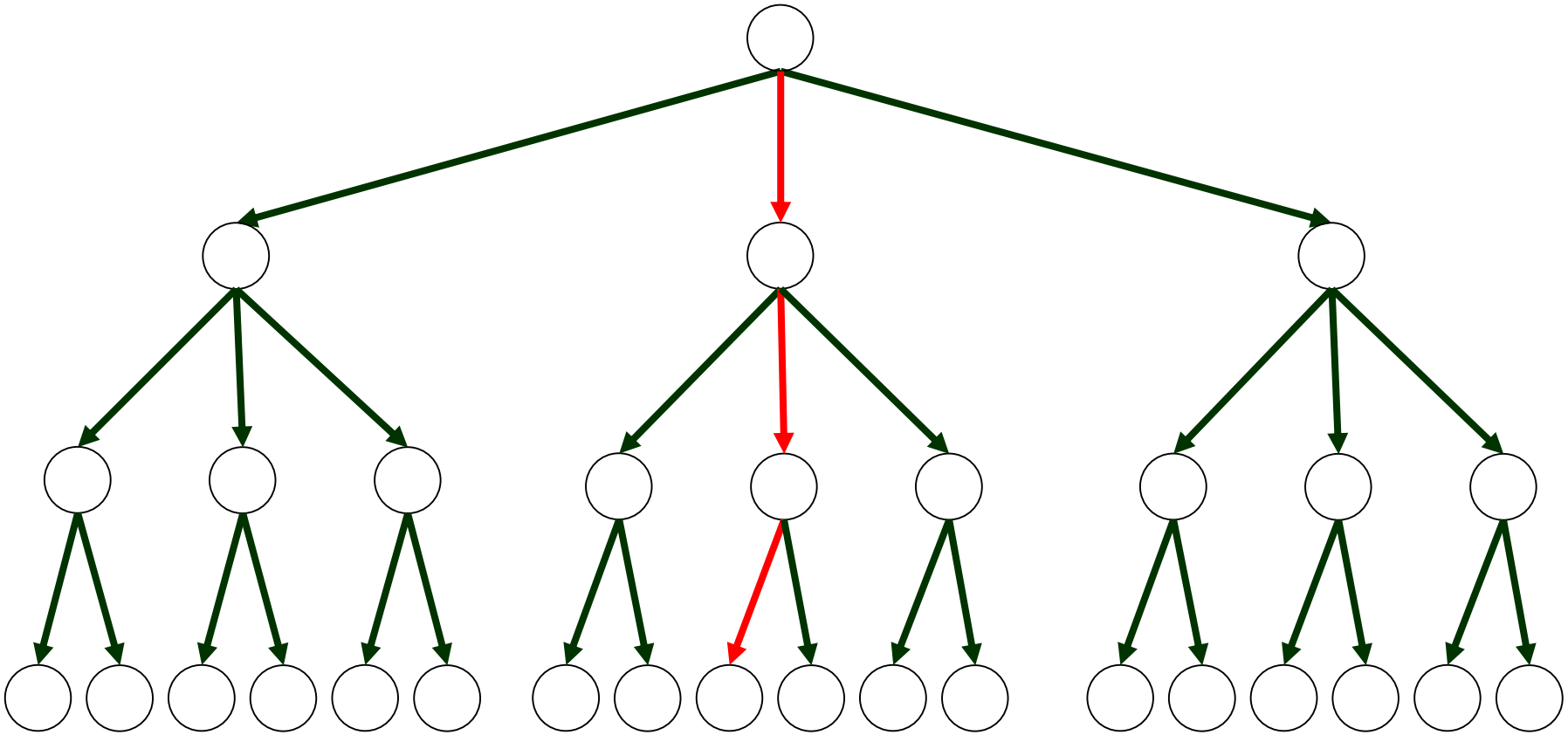
Advanced techniques

2-LDS



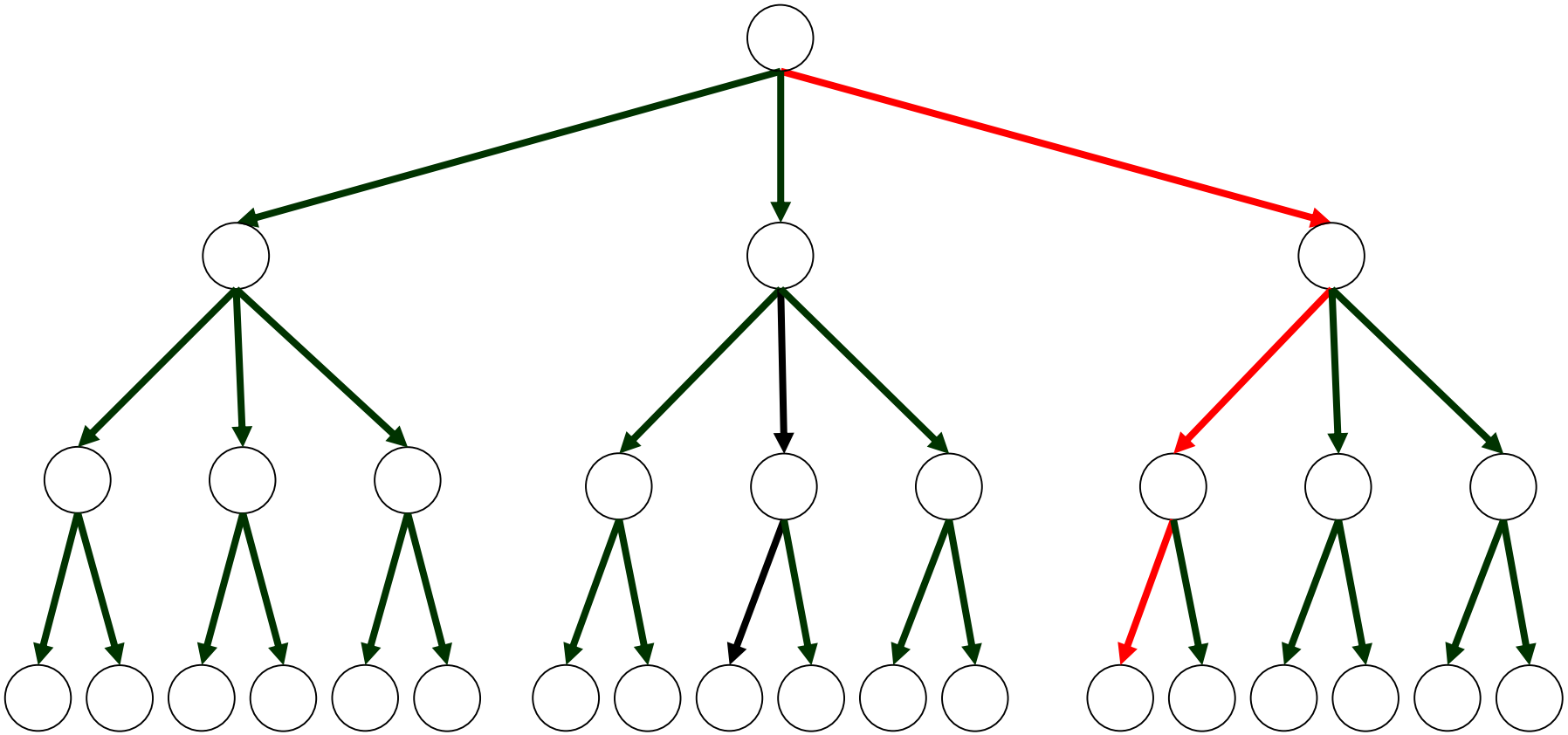
Advanced techniques

2-LDS



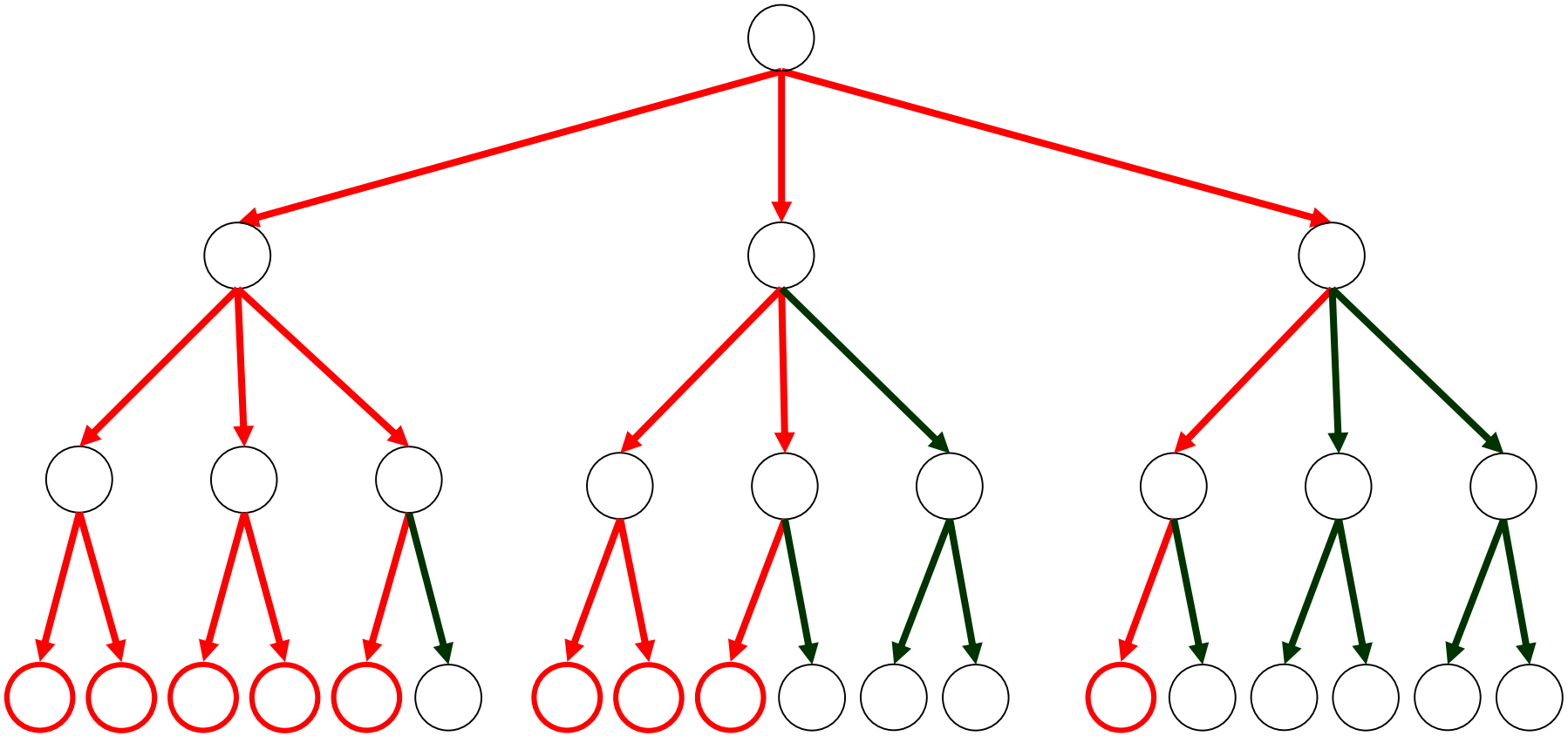
Advanced techniques

2-LDS



Advanced techniques

2-LDS



Advanced techniques

Limited Discrepancy Search

- Structured incomplete search
- Budget of discrepancies
- ‘Spend’ a discrepancy by going against the heuristic

Solving VRPs

- CP is “natural” for solving vehicle routing problems
  - Real problems often have unique constraints
  - Easy to change CP model to include new constraints
  - New constraints do not change core solve method
  - Infrastructure for complete (completish) search in subproblems

- LNS is “natural” for CP
  - Insertion leverages propagation

Presenter's Transportation Publications

- H. Aziz, C. Cahan, **C. Gretton, P. Kilby**, N. Mattei and T. Walsh. *A Study of Proxies for Shapley Allocations of Transport Costs*. Journal of Artificial Intelligence Research 56:573-611, 2016.
- H. Grzybowska, **C. Gretton, P. Kilby**, S. T. Waller. A Decision Support System for a Real-Time Field Service Engineer Scheduling Problem with Emergencies and Collaborations. Journal of the Transportation Research Board 2497:117-123*. 2015.*tificial Intelligence Research 56:573-611, 2016.
- **C. Gretton and P. Kilby**. A Study of Shape Penalties in Vehicle Routing. TRISTAN VIII, 2013.

Presenter's Local Search Publications

- D. Pham, J. Thornton, **C. Gretton**, and A. Sattar. *Combining Adaptive and Dynamic Local Search for Satisfiability*. Journal on Satisfiability, Boolean Model Checking, and Computation, 2008.
- S.Richter, M.Helmert and **C.Gretton**. *A Stochastic Local Search Approach to Vertex Cover*. Proceedings of the 30th German Conference on Artificial Intelligence (KI-2007), 2007.