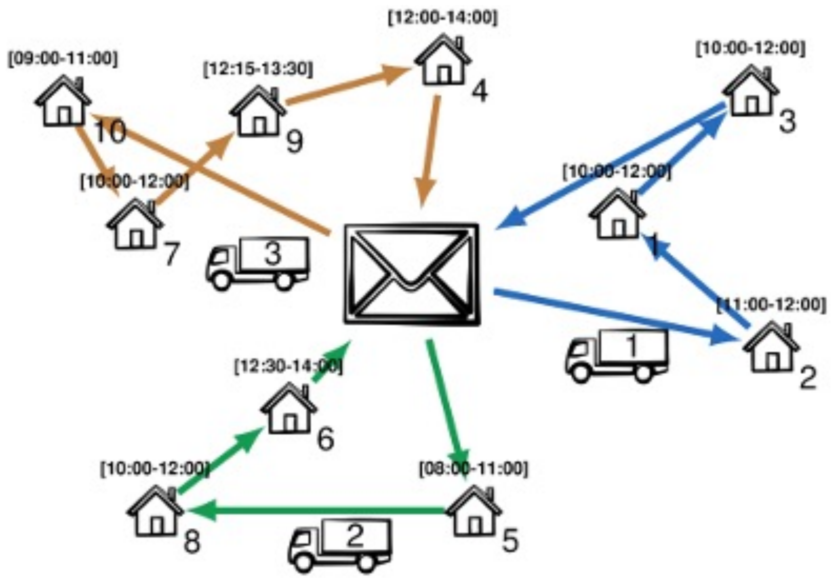
Genetic Programming Hyper-heuristics for Combinatorial Optimisation

Dr. Yi Mei
yi.mei@ecs.vuw.ac.nz
Evolutionary Computation Research Group
Victoria University of Wellington
IEEE Webinar, Dec 2016

Combinatorial Optimisation

- **Important** (many real-world applications)
- **Hard** to solve (usually NP-hard)

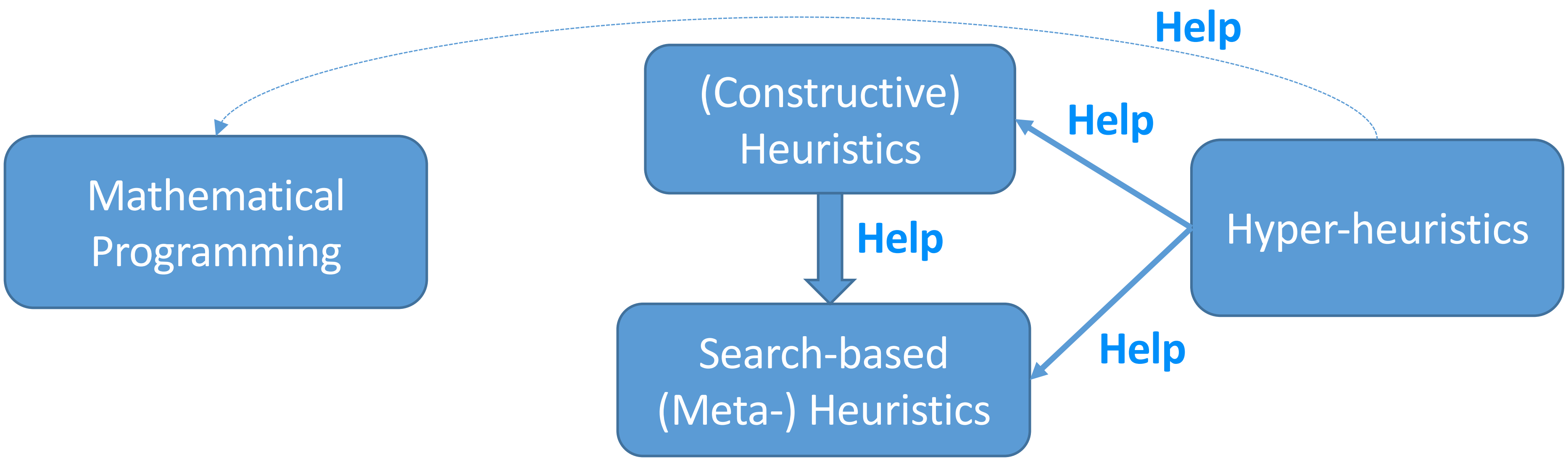
- Examples:
  - Traveling Salesman Problem
  - Knapsack Problem
  - Vehicle/Arc Routing Problem
  - Timetabling problem
  - Map Colouring
  - …



Example: Vehicle Routing Problem with Time Windows

Methods for Combinatorial Optimisation

- **Exact methods**
  - Mathematical programming
- **Approximated methods (heuristics)**
  - (Constructive) Heuristics
  - Search-based Heuristics (Meta-heuristics)
  - Hyper-heuristics



Mathematical Programming

- **Guarantee Optimality**
- Very **mathematical demanding**
- Can be very **slow**
- **Not flexible** in stochastic/dynamic environment
- Still need some **heuristics** (e.g. for branching)

$$\text{Max} \quad 2\kappa - \sum_{e \in E_R} h_e - \sum_{e \in E} \tilde{z}_e h_e$$

$$\text{s.t.} \quad h_e - s_i + s_j \geq 0 \quad \forall e = \{i,j\} \in E$$

$$h_e + s_i - s_j \geq 0 \quad \forall e = \{i,j\} \in E$$

$$-h_e + s_i + s_j \geq 0 \quad \forall e = \{i,j\} \in E$$

$$s_i - f_e \geq 0 \quad \forall e = \{i,j\} \in E$$

$$s_j - f_e \geq 0 \quad \forall e = \{i,j\} \in E$$

$$s_i + s_j - f_e \leq 1 \quad \forall e = \{i,j\} \in E$$

$$\sum_{e \in \delta(\{i\})} (h_e + f_e) - s_i \geq 0 \quad \forall i \in V$$

$$h_e + f_e \leq 1 \quad \forall e \in E$$

$$\kappa = \sum_{e \in E} \frac{d_e(h_e + f_e)}{Q} + \gamma$$

$$s_0 = 0$$

$$h_e f_e \in \{0,1\} \quad \forall e \in E$$

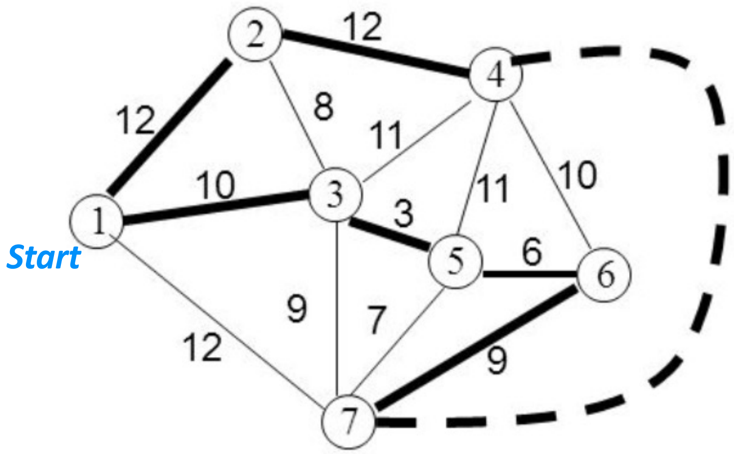
$$s_i \in [0,1] \quad \forall i \in V \setminus \{0\}$$

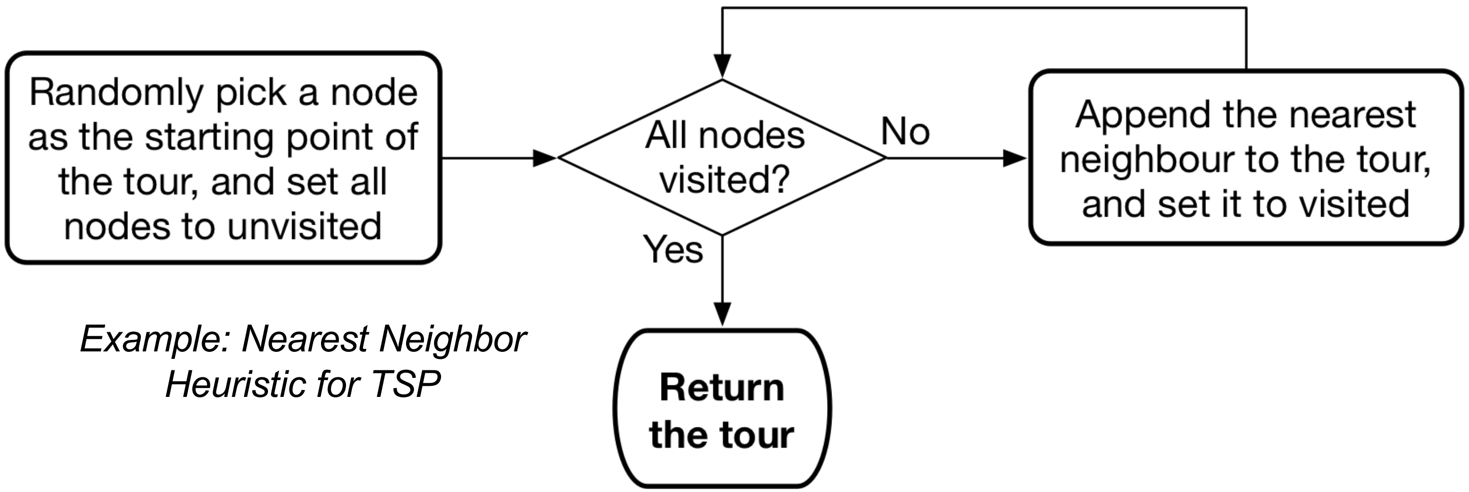
$$\kappa \in \mathbb{Z}_0^+$$

$$\gamma \in [0,1)$$

(Constructive) Heuristics

- Incrementally construct a solution from scratch
  - Easy to **understand** and **implement**
  - **Fast**
  - Reasonably **good solutions**
  - **Cannot guarantee optimality**





Example: Nearest Neighbor Heuristic for TSP

Search-based Heuristics (Meta-heuristics)

- Iteratively improve one or more solutions
  - Produce **high-quality solutions**
  - **Faster** than mathematical programming
  - Can **embed domain knowledge**
  - Can **combine with constructive heuristics** (initial solutions)
  - **Not flexible** in stochastic/dynamic environment
  - **Not scalable** well to large problem size

Simulated Annealing

Tabu Search

Variable Neighborhood Search

Memetic Algorithms

Guided Local Search

Genetic Algorithms

PSO

Ant Colony System

Hyper-heuristics

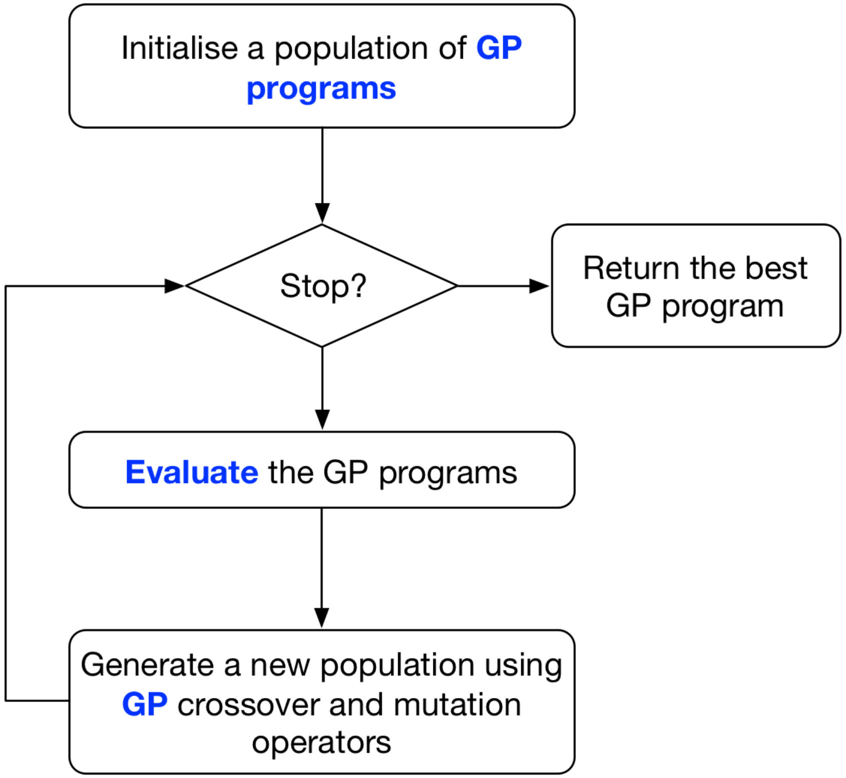
- Search for **heuristics** rather than solutions
    - **Fast** (Response immediately in dynamic environment)
    - **Flexible** (Solutions can be applied to **a range of** problem instances)
    - **Scalable** to large problems
    - Can discover **new knowledge** for problem solving

- A typical example: Genetic Programming Hyper-Heuristic (GPHH) for evolving dispatching rules for job shop scheduling

Branke, J., Nguyen, S., Pickardt, C.W. and Zhang, M., 2016. Automated design of production scheduling heuristics: a review. *IEEE Transactions on Evolutionary Computation*, *20*(1), pp.110-124.

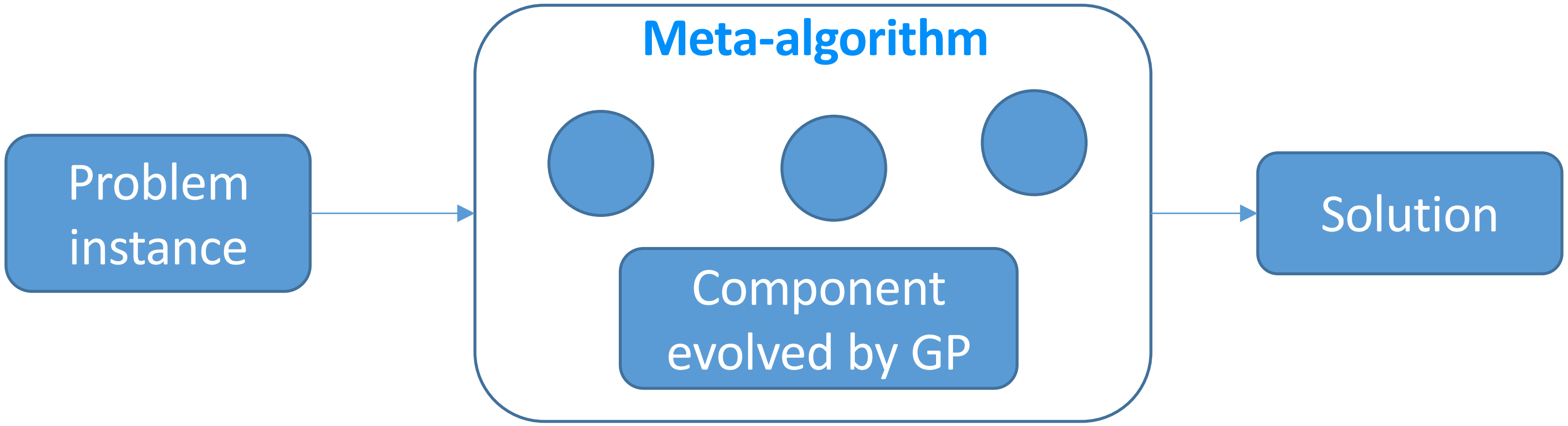
Genetic Programming

- Evolve a population of **computer programs**
- Crossover and mutation operators according to representation (e.g. tree, graph)

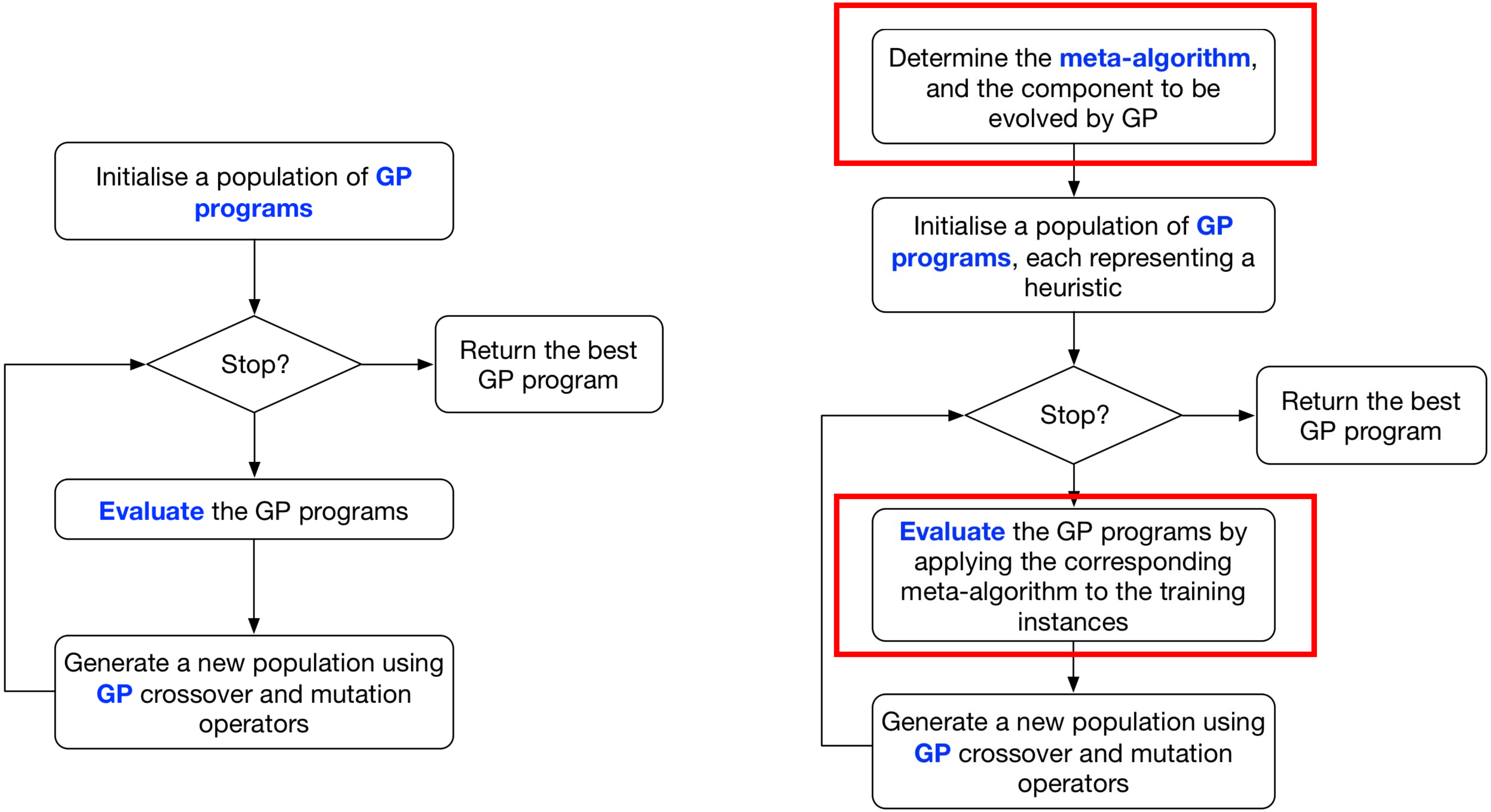


Genetic Programming as Hyper-Heuristic

- **Meta-algorithms**
  - An algorithm to generate a solution given a problem instance



Genetic Programming as Hyper-Heuristic

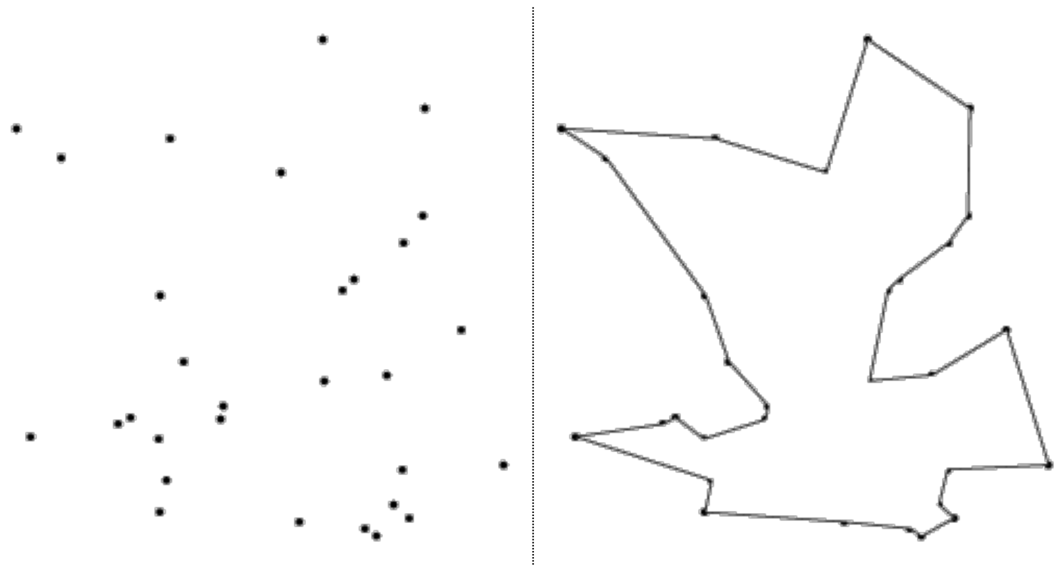


Issues for GPHH

- How to **represent** a heuristic (GP program)?
  - Tree?
  - Graph?
  - Sequence?
- How to **evaluate** a heuristic?
  - Performance on a set of problem instances?
  - Generalisation? Performance on unseen instances?

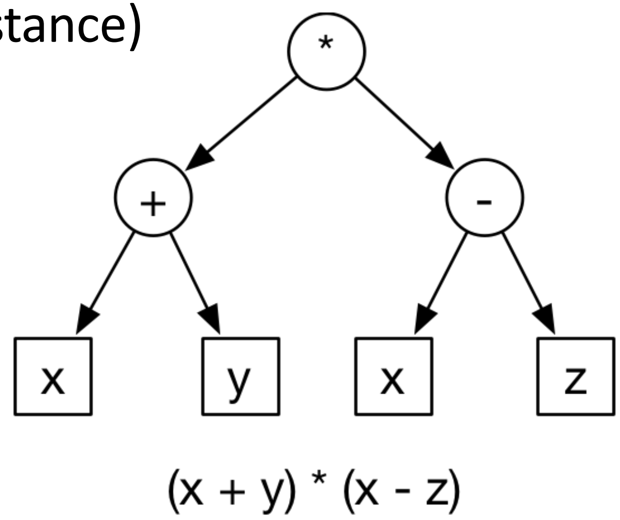
Representation of Heuristics

- Example: Constructive heuristic for TSP **Meta-algorithm**
  - **Step 0**: $S = ()$, all nodes unvisited;
  - **Step 1**: Select an unvisited node v^* based on some **priority function**, $S = (S, v^*)$;
  - **Step 2**: If all nodes visited, return S, otherwise, go back to Step 1;



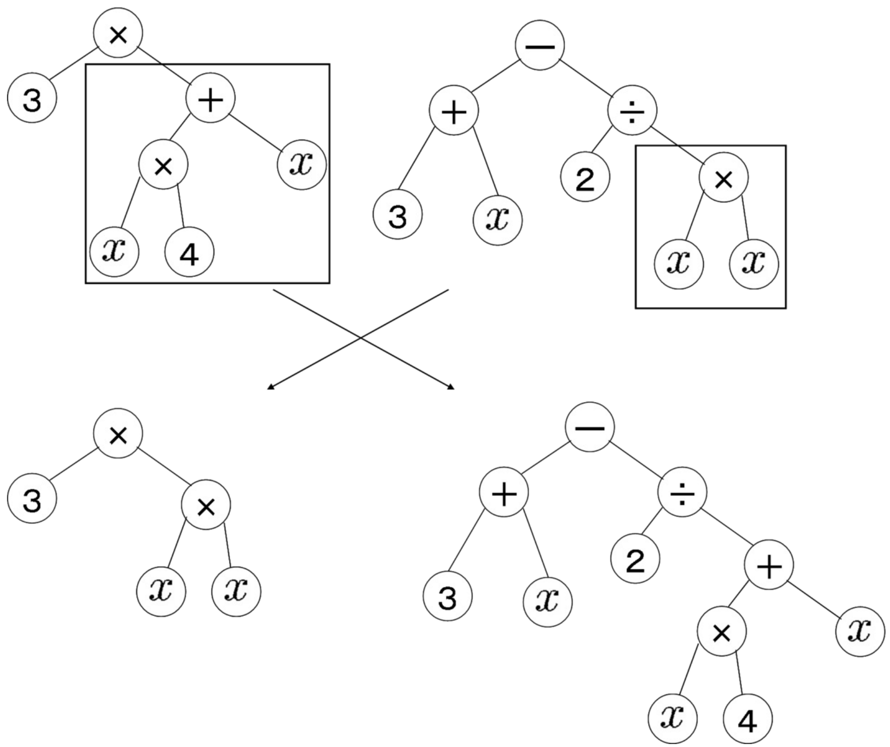
Representation of Heuristics

- Calculate the priority for all the unvisited node using the **priority function** $h(v;\Theta)$, then select the node with the highest priority
  - Nearest neighbour heuristic: $h(v;\Theta) = -d(v,S)$
- For evolving constructive heuristics for TSP using GP, one can represent the **priority functions as syntax trees**
  - **Terminals**: state features (e.g. location, distance)
  - **Functions**: +, -, *, /, min, max, …

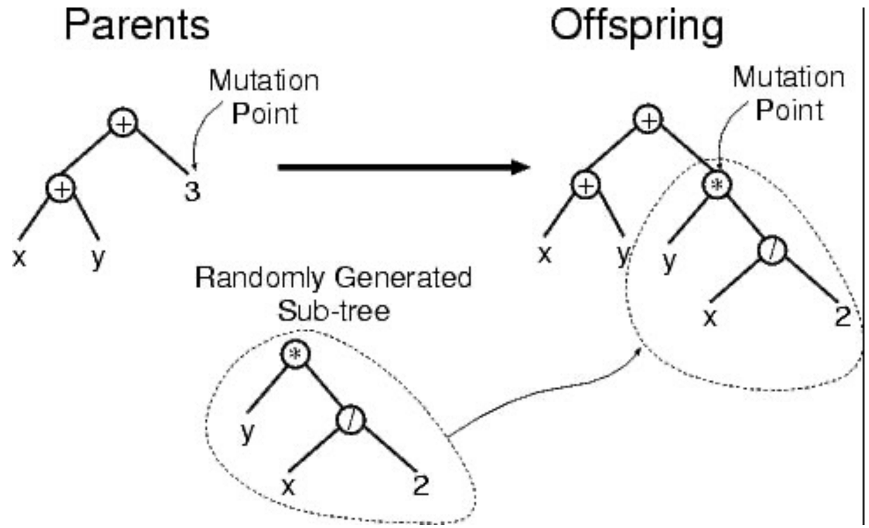


Representation of Heuristics

- Evolve the priority trees using GP crossover/mutation



GP crossover



GP mutation

Poli, R., Langdon, W.B., McPhee, N.F. and Koza, J.R., 2008. *A field guide to genetic programming*.

Evaluation of Heuristics

- A heuristic π produces a solution given a problem instance
- Performance on an instance i: $perf(\pi, i)$ = **objective value of the produced solution** to the instance i
- Overall performance on a set of instances I: $perf(\pi, I)$ = **mean of the normalised objective values** of the produced solutions to each instance $i \in I$
  - Normalise by the lower bound
  - Normalise by the performance of reference heuristic/method

- But a heuristic perform well on the training instance(s) may not perform well on unseen instances (**overfitting**)
- **Generalisation** is an important issue (performance on unseen instances)

Evaluation of Heuristics

- Various strategies to **improve generalisation**
  - Use **comprehensive training instances**
  - Use small training set + **change training set after each generation** (similar to stochastic gradient descent/mini-batch in machine learning)
  - **Regularisation**: restrict the maximal depth of GP trees
  - **Restrict the structure** of GP trees (e.g. strongly-typed GP, grammar-based GP)
  - …

In This Talk…

- Evolve dispatching rules for job shop scheduling
- Evolve heuristics for arc routing problem
- Evolve heuristics for memetic algorithm in traveling thief problem

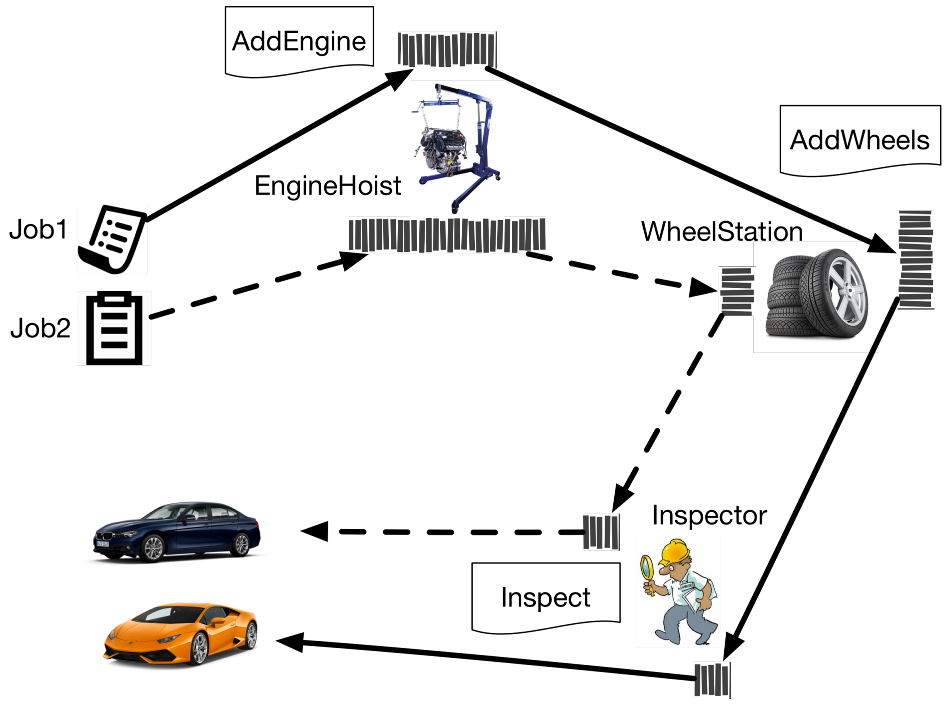
GPHH for Evolving Dispatching Rules for Job Shop Scheduling

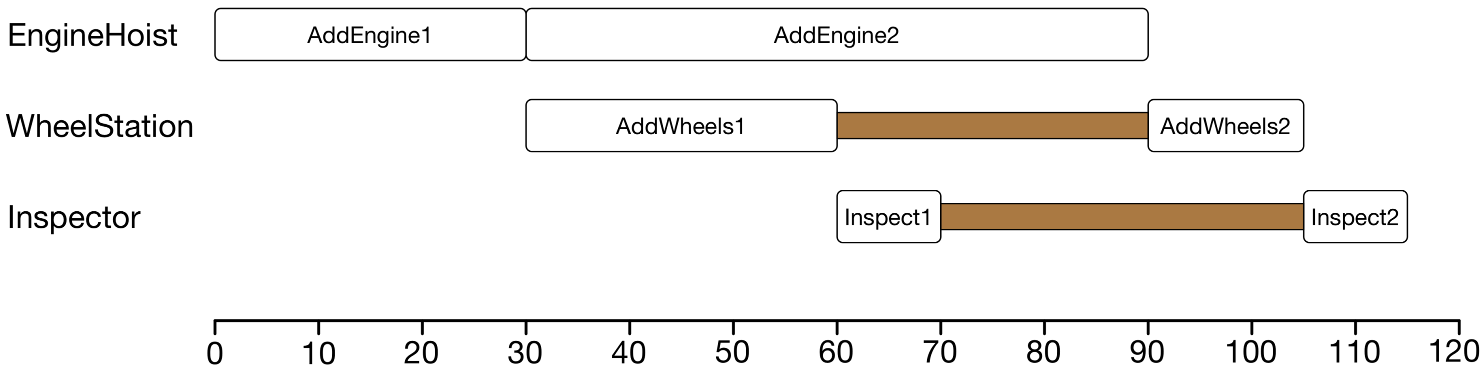
Job Shop Scheduling

- Process a set of **jobs** with a set of **machines**
- Each job has a sequence of **operations**, each processed by a certain machine
- Each job has **arrival time**, **due date**, **weight**, etc
- Each operation has a **processing time**
- **Objective**: minimise makespan/flowtime/tardiness
- **Constraint**
  - Each machine can process at most one operation at a time
  - An operation cannot start until its preceding operations have completed

Job Shop Scheduling

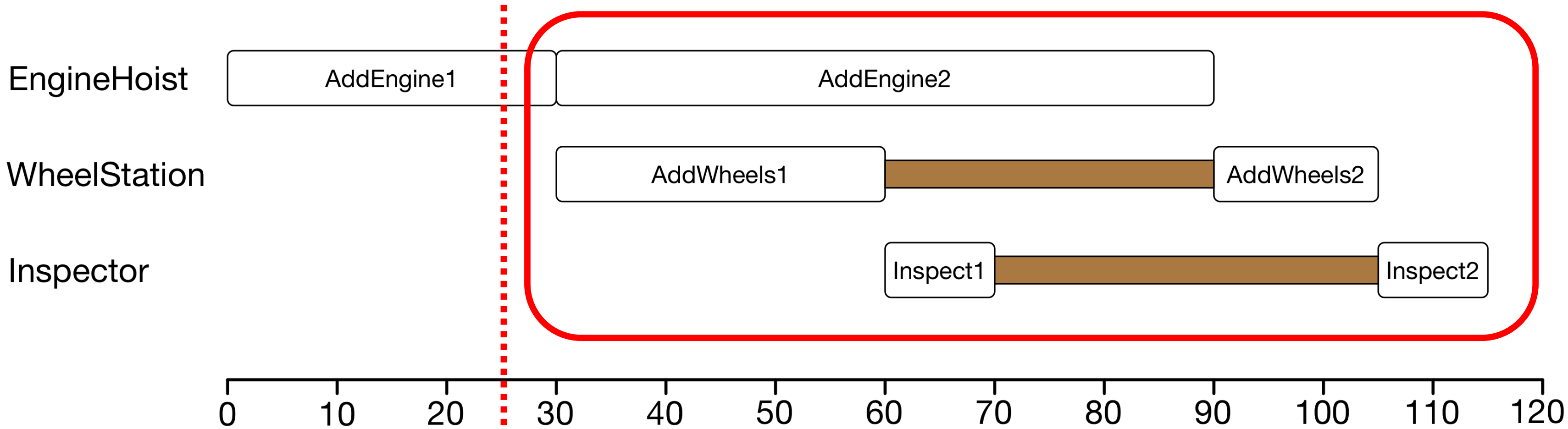
- Car manufacturing
- 3 machines (Engine Hoist, Wheel Station, Inspector)
- 2 jobs, each with 3 operations
  - 1) AddEngine
  - 2) AddWheel
  - 3) Inspect





Dynamic Job Shop Scheduling

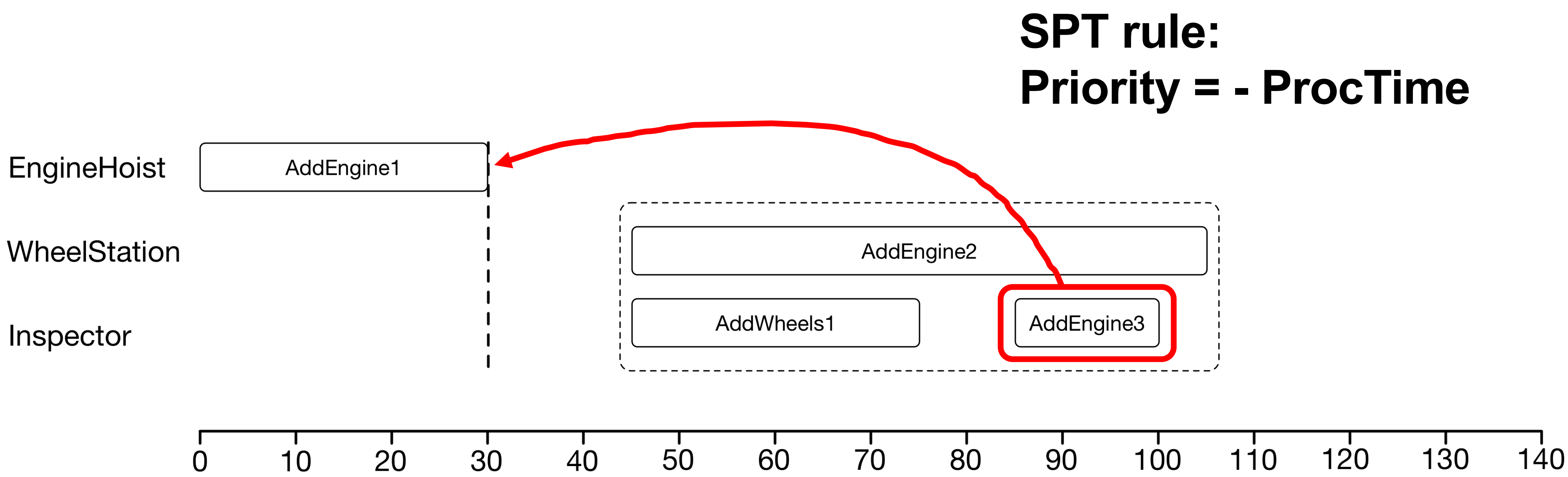
- **Unpredicted events** (e.g. new job arrivals) occur during the execution of the schedule
- **Immediate** response is needed
- Solution optimisation methods are usually **too slow** to respond effectively



Job3 [AddEngine3 -> AddWheels3 -> Inspect3] arrives

Dispatching Rule

- Whenever a machine becomes idle and its queue is not empty
  - Calculate the priority of the operations waiting in the queue
  - Select the most prior operation to process next



Dispatching Rule

- Many rules have been designed manually (FCFS, SPT, EDD, PT+WINQ, 2PT+WINQ+NPT, WATC, …)

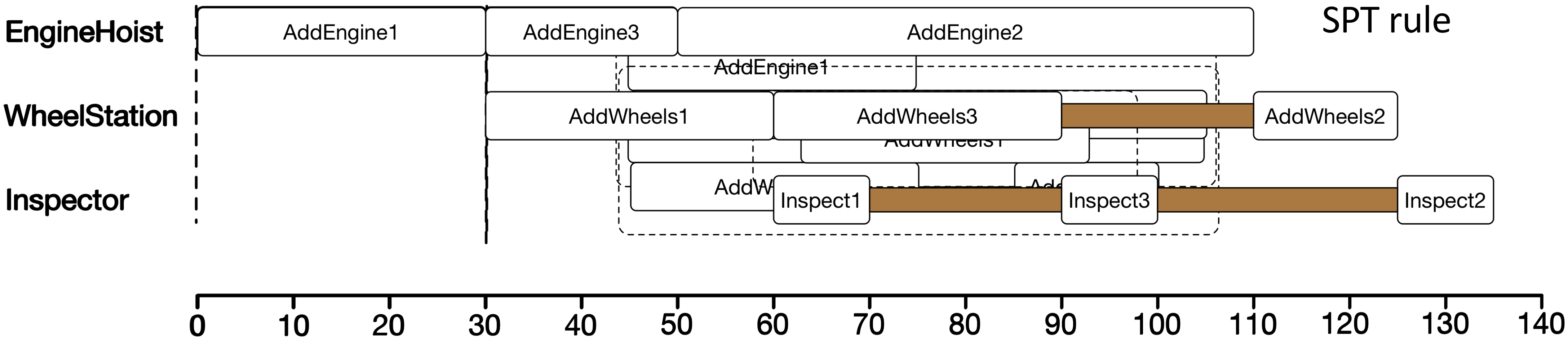
- Can handle dynamic JSS very well
  - **Quick** response
  - Good **scalability** (work well for huge problems)
  - **Flexibility** (can apply to a range of JSS instances)

- Manually designing effective dispatching rules is very challenging
  - **Many interdependent factors** (features) to consider

- Evolve dispatching rules using GPHH

Evolve Dispatching Rules by GPHH

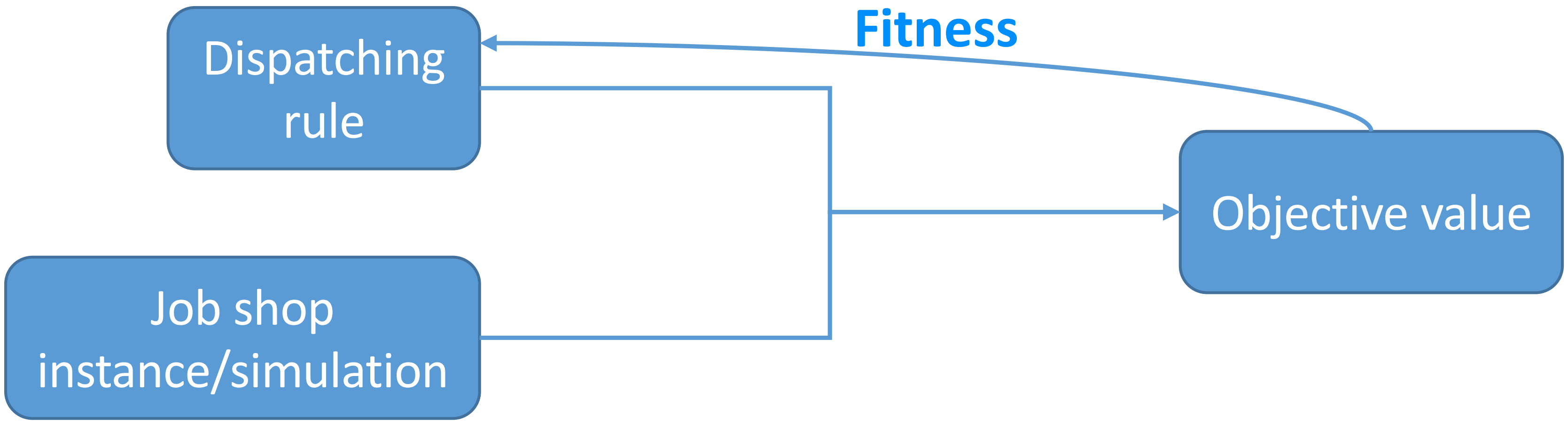
- **Meta-algorithm**: discrete event simulation
  - Start from time 0, empty schedule, initial jobs waiting in their machines
  - New jobs may arrive in real time (e.g. Poisson process)
  - As soon as a machine is idle and there are jobs waiting in its queue, select a job from its queue to be processed next using the **dispatching rule**
  - Stop if all jobs completed



SPT rule

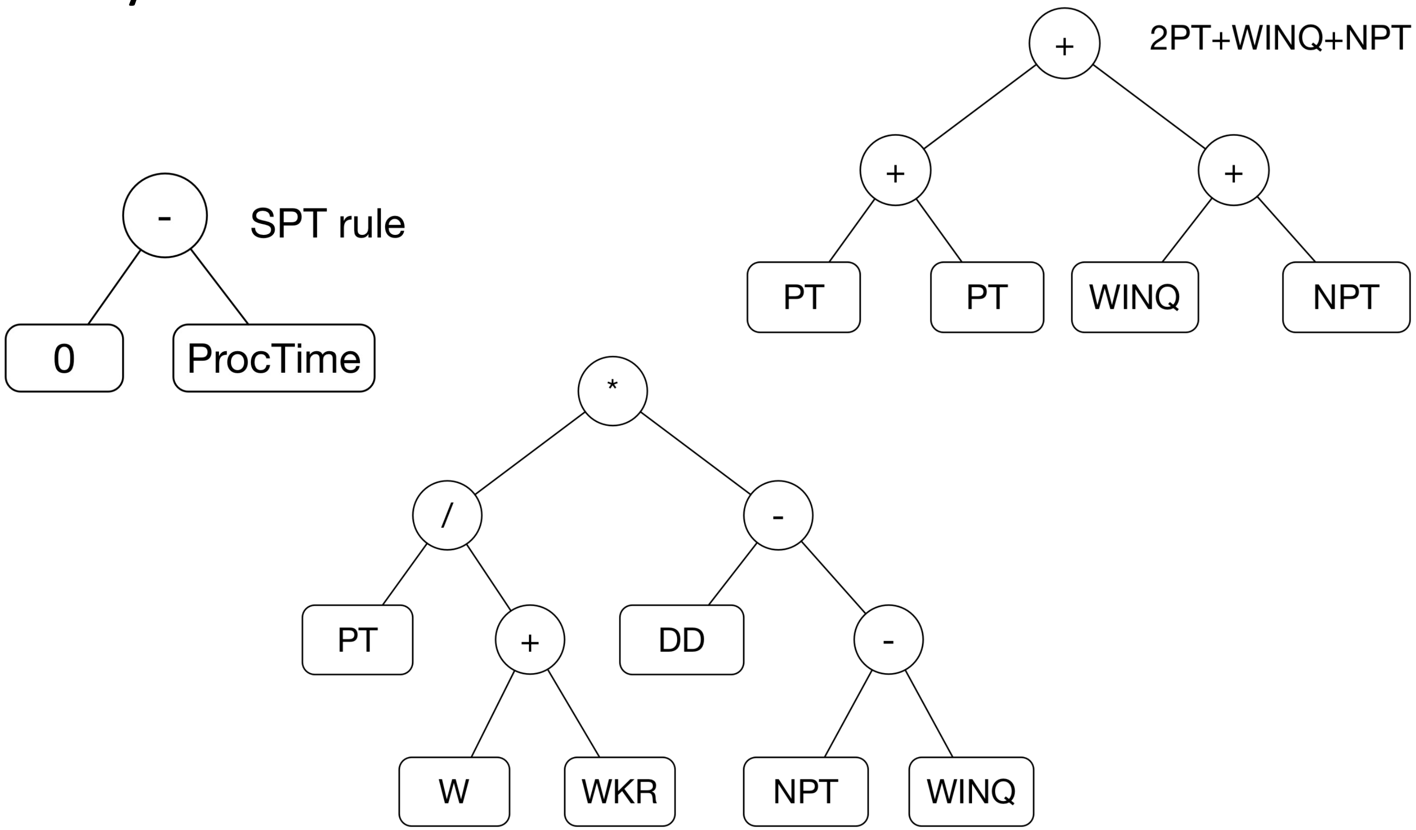
Evolve Dispatching Rules by GPHH

- **Objectives**
  - Makespan: $\max C_j$
  - Mean flowtime: $\frac{1}{N}\sum_{j=1}^{N}(C_j - a_j)$
  - Mean weighted tardiness: $\frac{1}{N}\sum_{j=1}^{N} w_j T_j$, where $T_j = \max\{C_j - d_j, 0\}$



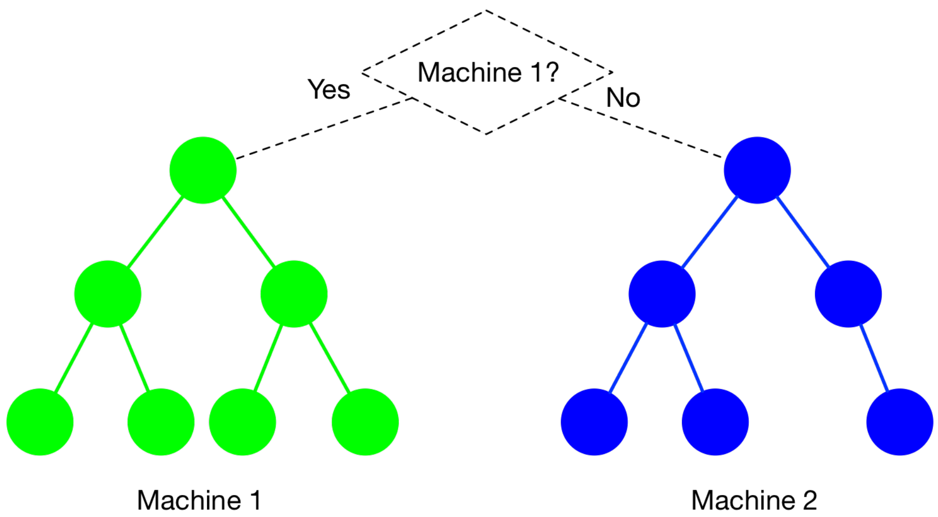
Representation

- Single priority tree for all the machines



Representation

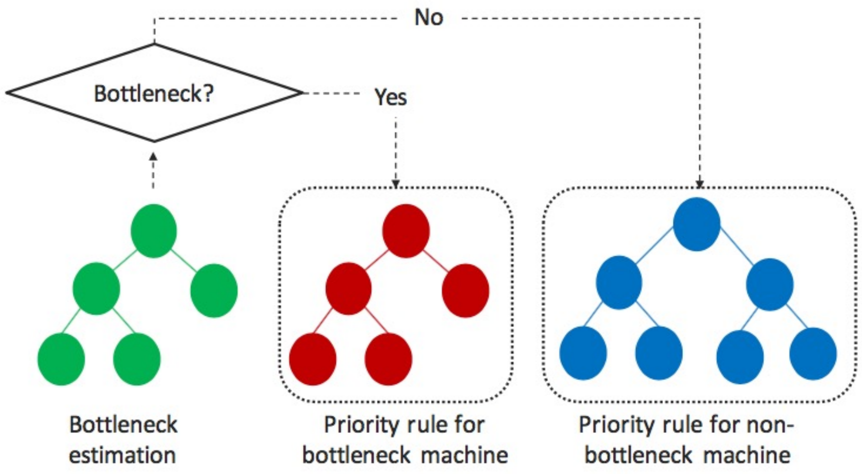
- Machine-specific priority trees
- Effective especially when machines have different scenarios
  - Unbalanced job shop
- Two machines with **different utilisations**



Hunt, R., Johnston, M. and Zhang, M., 2014, July. Evolving machine-specific dispatching rules for a two-machine job shop using genetic programming. In *2014 IEEE Congress on Evolutionary Computation (CEC)* (pp. 618-625). IEEE.

Representation

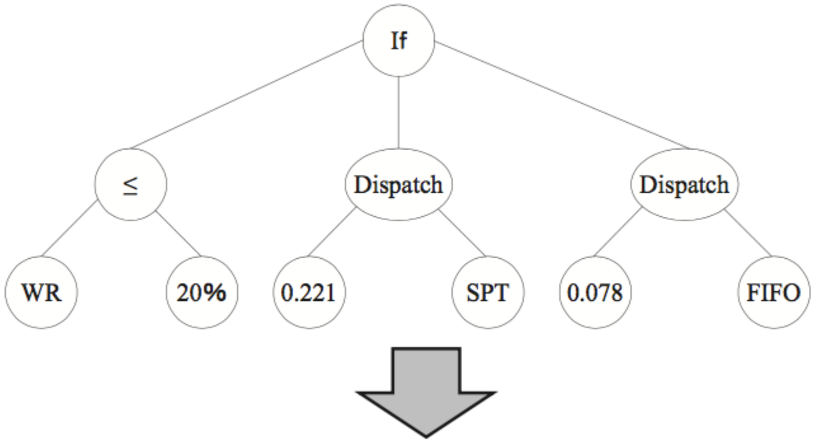
- Machine-specific priority trees
- Effective especially when machines have different scenarios
  - Unbalanced job shop
- **Bottleneck** machines vs **non-bottleneck** machines



Jakobović, D. and Budin, L., 2006, April. Dynamic scheduling with genetic programming. In *European Conference on Genetic Programming* (pp. 73-84). Springer Berlin Heidelberg.

Representation

- **Decision tree**-like representation
  - Allow idle machines to wait some time even with non-empty queue

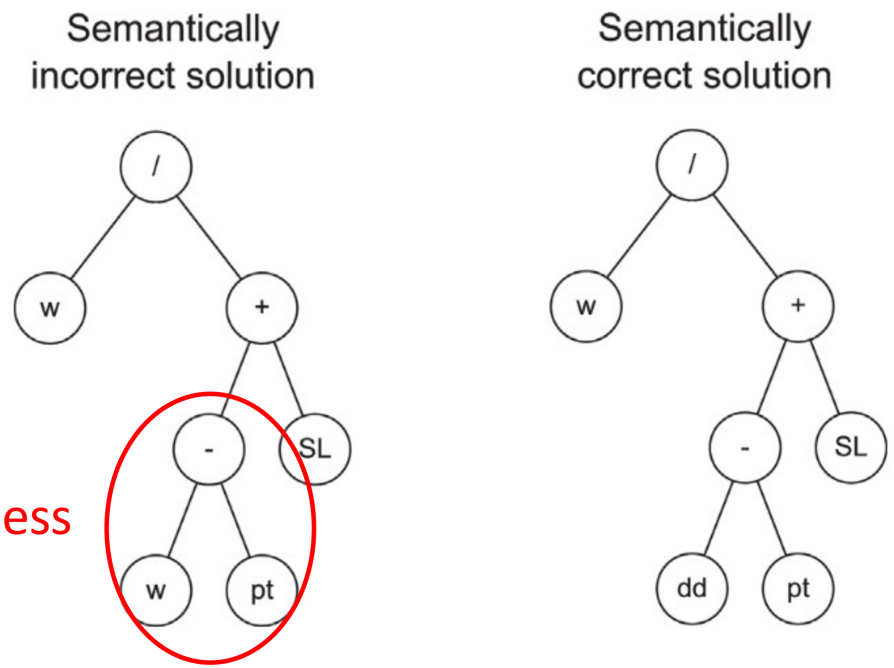



```
If (workload ratio is less than or equal to 20%)
       Use the SPT rule with non-delay factor of 0.221
Else
       Use the FIFO rule with non-delay factor of 0.078
```

Nguyen, S., Zhang, M., Johnston, M. and Tan, K.C., 2013. A computational study of representations in genetic programming to evolve dispatching rules for the job shop scheduling problem. *IEEE Transactions on Evolutionary Computation*, *17*(5), pp.621-639.

Representation

- Dimensionality-Aware GP
  - Different attributes have different **dimensions** (units: time, count, weight, …)
  - Keep **semantic correctness** with respect to dimensionality



meaningless

Đurasević, M., Jakobović, D. and Knežević, K., 2016. Adaptive scheduling on unrelated machines with genetic programming. *Applied Soft Computing*, *48*, pp.419-430.

Feature Selection for GP Terminals

- **Many features**: huge search space
- Some features are **redundant/irrelevant** (e.g. due date is irrelevant when minimising makespan)
- Select a subset of important features
- Feature selection is **challenging** as it depends on
  - Job shop scenario (utilisation level, due date factor, …)
  - Objective (flowtime, tardiness, …)
  - Complex interaction between features

- Learn the **importance** of features

Feature Selection for GP Terminals

- Ideally, we only need the features that **contribute to the optimal individual**
- However, the optimal individual is unknown

- **Approximation**
  - If a feature contributes to a **better** individual, then it is more likely to contribute to the optimal individual
  - If a feature contributes to **more** individuals, then it is more likely to contribute to the optimal individual

Feature Selection for GP Terminals

- Use the **number of appearances** to measure the contribution of a feature to an individual
- Update the importance estimation during GP process

$$w_i \leftarrow w_i + \sum_{j=1}^{P} \frac{count_{ij}}{\sum_{k=1}^{n} count_{kj}} \times fitness_j$$

- During mutation, the **probability** of choosing a feature when generating the new sub-tree depends on its importance

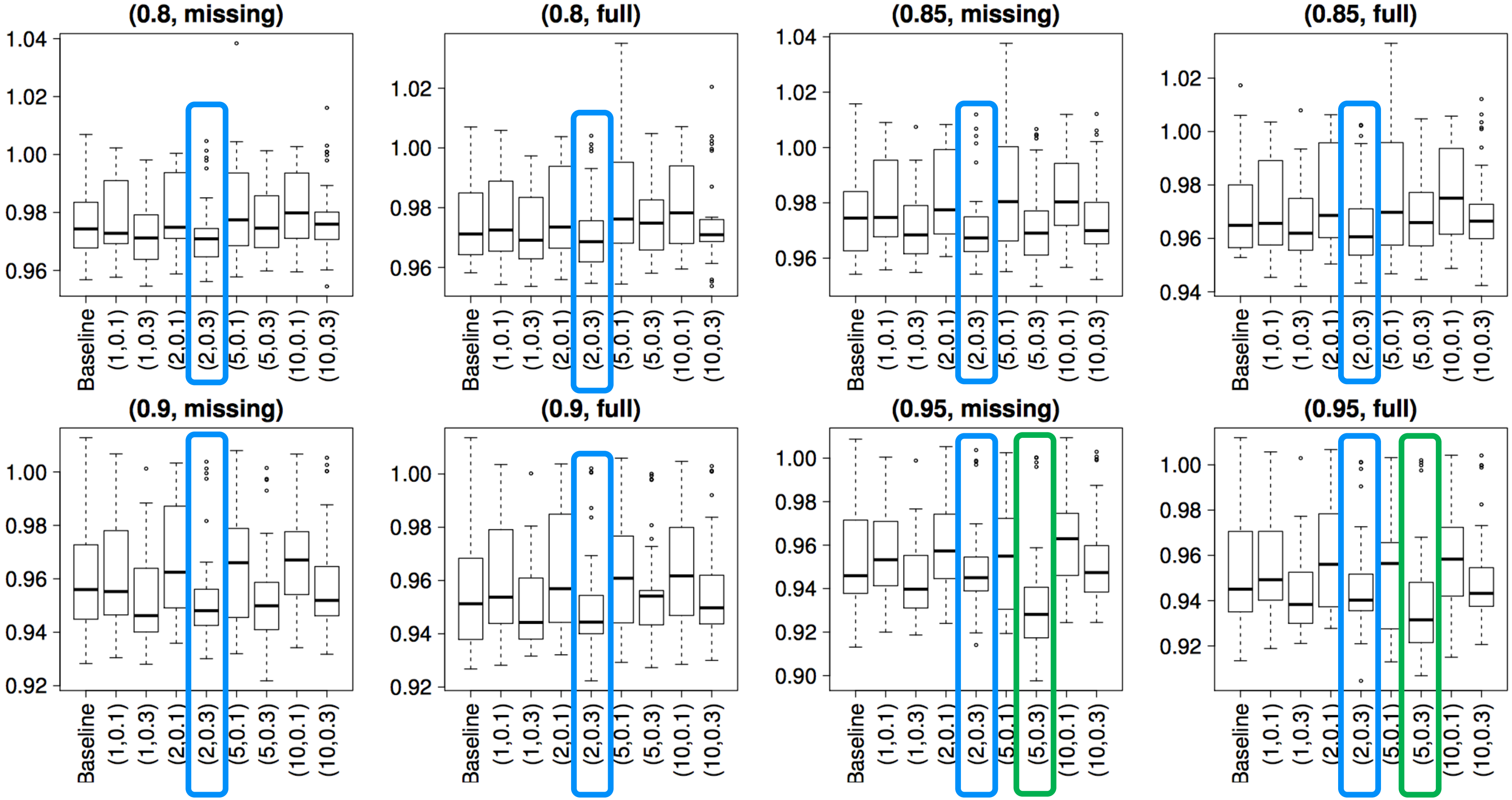
$$p_i = \frac{w_i^{\lambda}}{\sum_{k=1}^{n} w_k^{\lambda}}$$

Riley, M., Mei, Y. and Zhang, M., 2016, November. Improving job shop dispatching rules via terminal weighting and adaptive mutation in genetic programming. In *Evolutionary Computation (CEC), 2016 IEEE Congress on* (pp. 3362-3369). IEEE.

Feature Selection for GP Terminals

- Experiments
  - 8 scenarios (4 utilisation levels $\times$ 2 operation settings)
  - Utilisation: 0.8, 0.85, 0.9, 0.95
  - Ops:
    - Missing: uniform from 2 to the number of machines
    - Full: equal to the number of machines
  - λ values: 1, 2, 5, 10
  - 2 mutation rates: 0.1 and 0.3

Feature Selection for GP Terminals

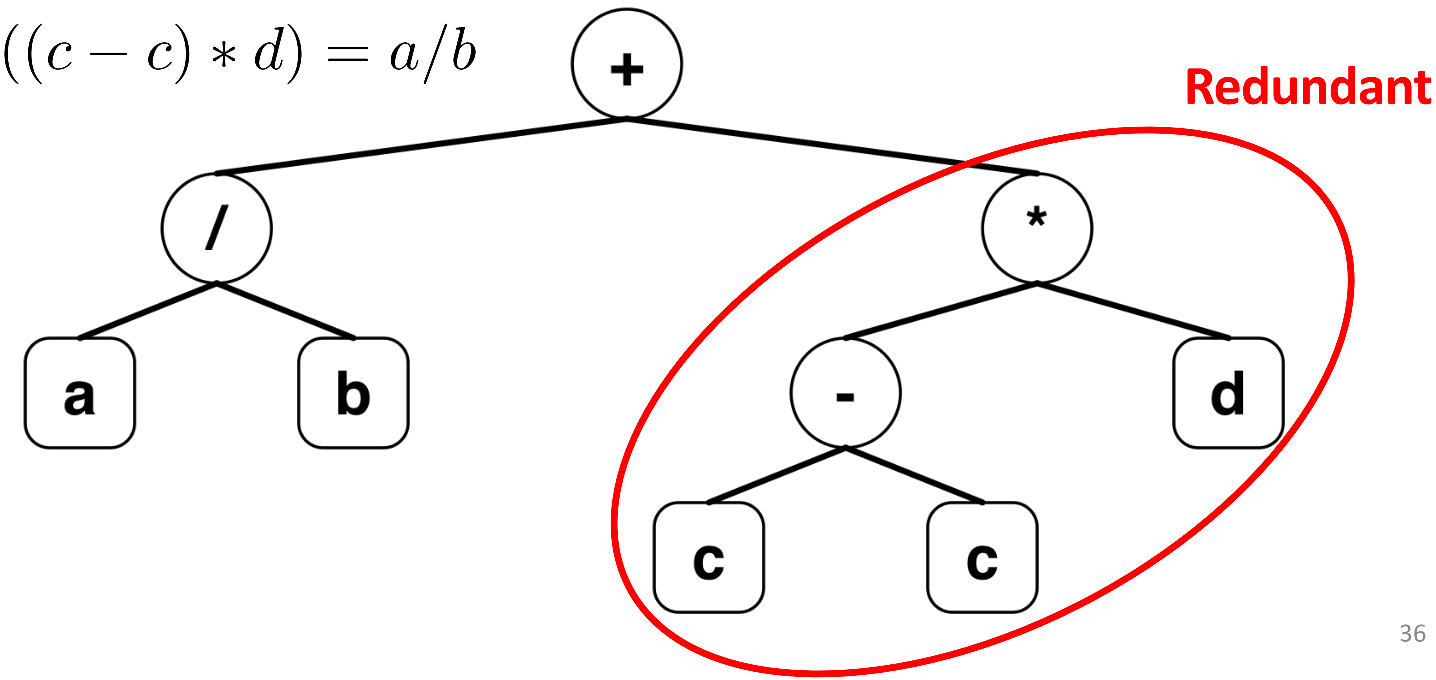


$\lambda = 2$ or 5 mutation rate $= 0.3$

Feature Selection for GP Terminals

- Using **number of appearances** may be **misleading**

$$(a/b) + ((c - c) * d) = a/b$$

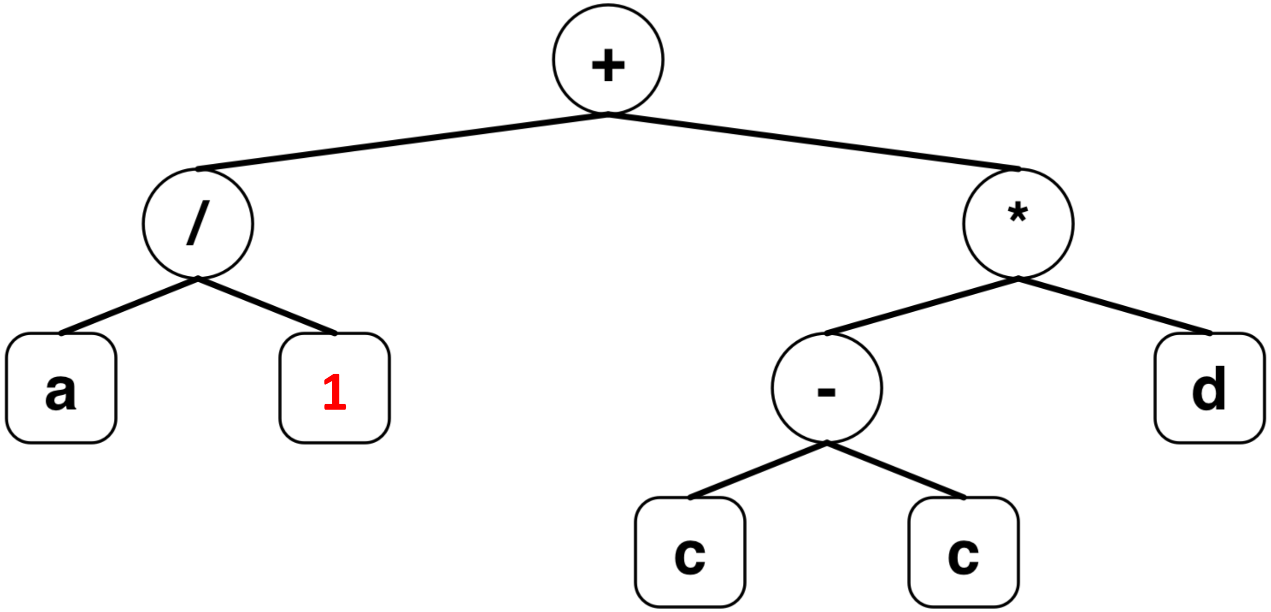


But feature *c* appears twice, which is more than *a* and *b*.

Feature Selection for GP Terminals

- A **new contribution measure**

$$contribution(feature, tree) = fit(tree|feature = 1) - fit(tree)$$



Fit(tree) = 0.9

Fit(tree|b=1) = 1.1

Contribution(b) = 0.2

Contribution(c) = 0
Contribution(d) = 0

Yi Mei, Mengjie Zhang, Su Nguyen, "Feature Selection in Evolving Job Shop Dispatching Rules with Genetic Programming," *Genetic and Evolutionary Computation Conference (GECCO),* Denver, USA, 2016.

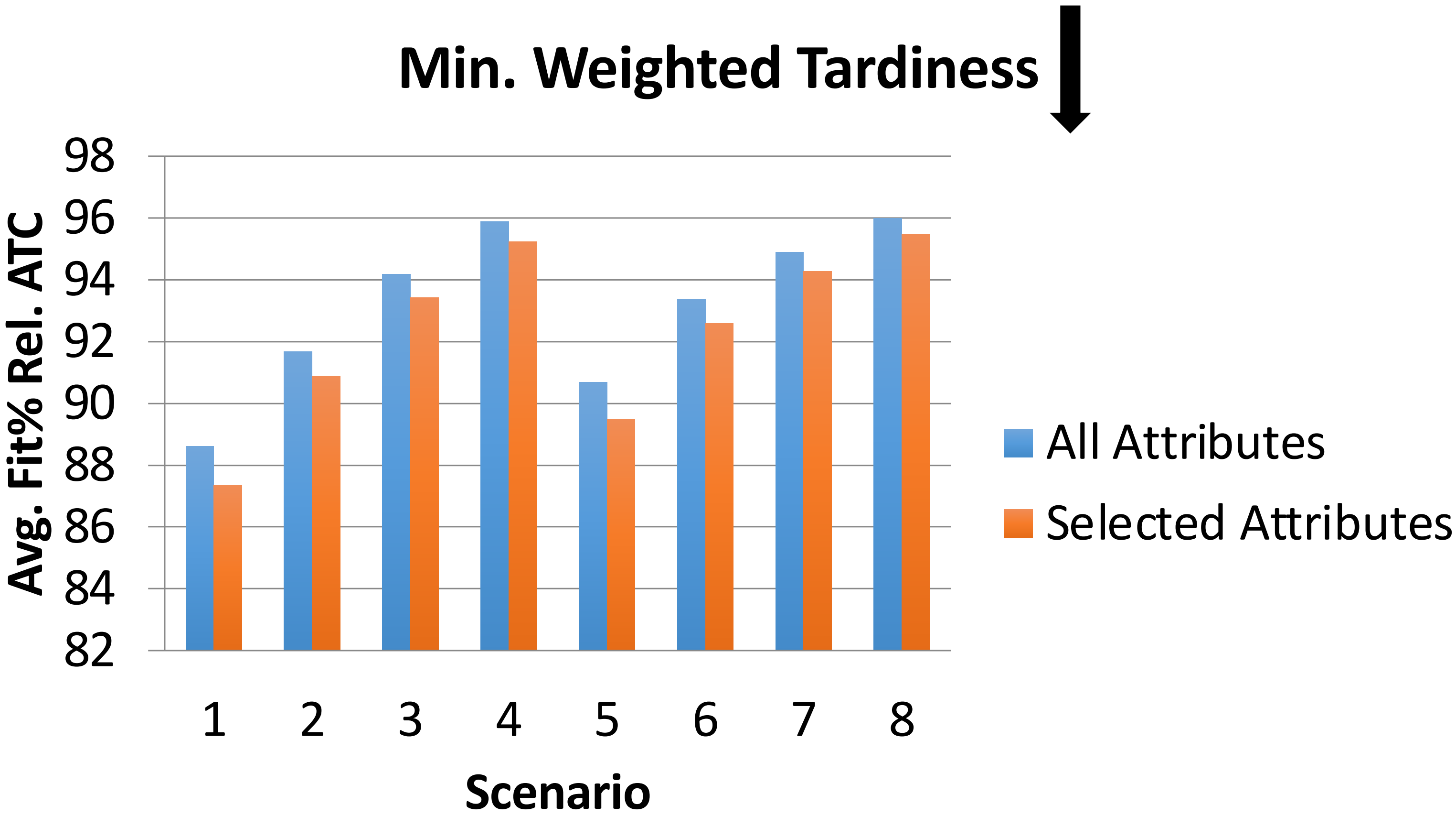
Feature Selection for GP Terminals

- Step 1: Conduct 30 pilot GP runs, collect 30 best individuals
- Step 2: Calculate contribution of each feature to each individual
- Step 3: Select a feature if it contributes to more than 15 individuals

Notation	Description
NOW	The current time.
$\boxed{\text{PT}}$	Processing time of the operation.
IPT	Inverse of the processing time.
$\boxed{\text{NOPT}}$	Processing time of the next operation.
ORT	Ready time of the operation.
$\boxed{\text{MRT}}$	Ready/Idle time of the machine.
NMRT	Ready time of the next machine.
WIQ	Work in the current queue.
$\boxed{\text{WINQ}}$	Work in the next queue.
NOIQ	Number of operations in the current queue.
$\boxed{\text{NOINQ}}$	Number of operations in next queue.
WKR	Work remaining (including the current operation).
NOR	Number of operations remaining.
FDD	Flow due date of the operation.
DD	Due date of the job.
$\boxed{\text{W}}$	Weight of the job.

Min weighted tardiness

Feature Selection for GP Terminals



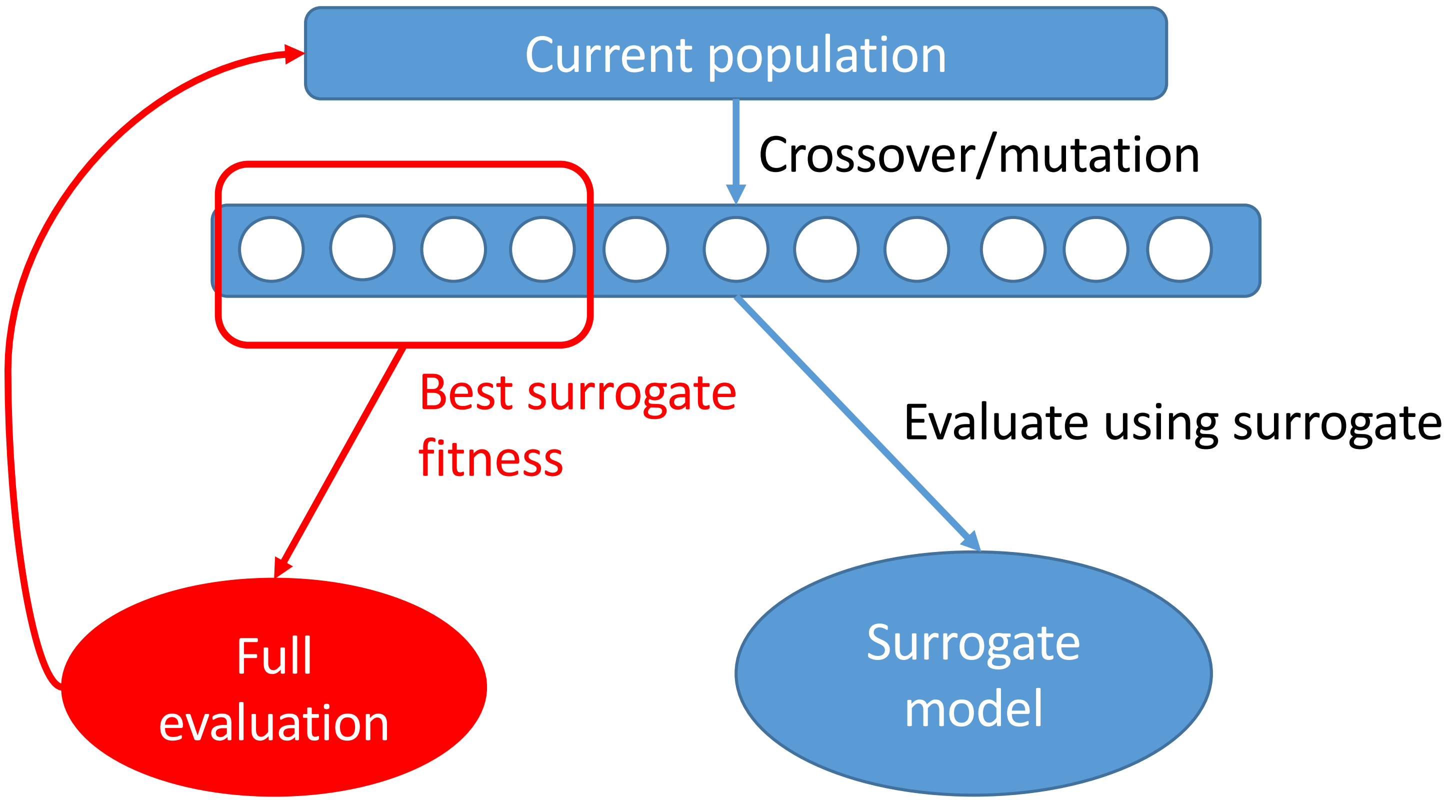
Evaluation Model

- **Standard**
  - A set of **static instances** (normalised by lower bound/reference rule)
  - **Dynamic discrete event simulation(s)**
    - 10 machines, 2500 jobs, 2~10 operations per job
    - 500 warm-up jobs for steady-state performance
    - Different utilisation levels (0.85, 0.9, 0.95) and due date factors (3, 4, 5)

- **Change the random seed** of the simulation(s) at each generation
  - Much **better generalisation**
  - Much **faster** (only one replication per generation)

Hildebrandt, T., Heger, J. and Scholz-Reiter, B., 2010, July. Towards improved dispatching rules for complex shop floor scenarios: a genetic programming approach. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation* (pp. 257-264). ACM.

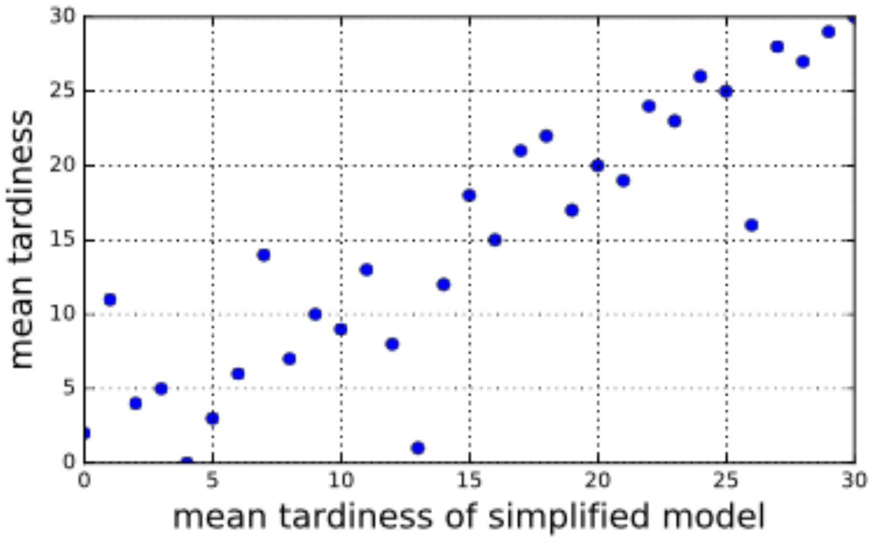
Surrogate Evaluation Models



Surrogate Evaluation Models

- Smaller job shop simulation

	Original	Surrogate
No. Machines	10	5
No. Jobs	5000	500
No. Warmup Jobs	500	100
Min Ops	2	2
Max Ops	14	7



Nguyen, S., Zhang, M. and Tan, K.C., Surrogate-Assisted Genetic Programming With Simplified Models for Automated Design of Dispatching Rules. IEEE Transactions on Cybernetics, in press, DOI 10.1109/TCYB.2016.2562674.

Surrogate Evaluation Models

- **Phenotypic characterisation**
  - A set of **decision situations** and a **reference rule**
  - For each decision situation, measure the difference between the reference rule and the characterised rule
  - Characterised by a **decision vector**

decision situation	attribute set **s** s_1	s_2	s_3	ranking by reference rule	ranking by other rule	decision vector **d**
1	3	4	8	1	2	
1	7	6	15	**2**	1	**2**
2	23	17	1	2	2	
2	8	9	3	**3**	1	**3**
2	6	4	6	1	3	
⋮		⋮		⋮	⋮	⋮
k	7	3	9	2	2	
k	4	8	6	**1**	1	**1**

Hildebrandt, T. and Branke, J., 2015. On using surrogates with genetic programming. *Evolutionary computation*, *23*(3), pp.343-367.

Surrogate Evaluation Models

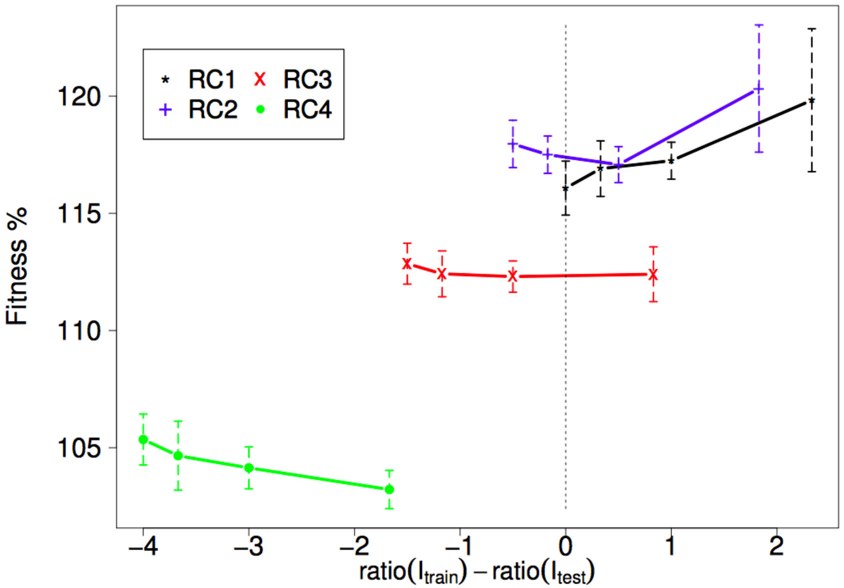
- If two rules have similar phenotypic characterisation, i.e. decision vectors, then they tend to have similar fitness values
- A **<decision vector, fitness>** database (the fully evaluated individuals in the last 2 generations)

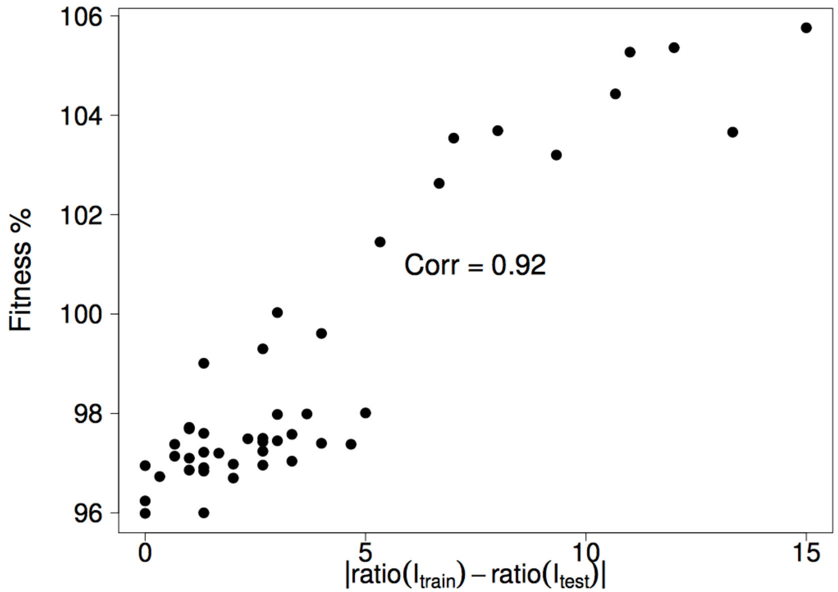
	d_1	d_2	…	d_k	fitness
$rule_1$:	2	3	…	1	1456
$rule_2$:	1	2	…	2	1123
⋮		⋮			⋮
$rule_m$:	1	3	…	1	1293

- **Nearest neighbour regression** – set the approximated fitness to the fitness of the closest rule in the database

Surrogate vs Reusability

- In static case, we aim to train dispatching rules using small instances (surrogate), which can be reused on large instances
- Such reusability strongly relates to $\boldsymbol{ratio} = \dfrac{\boldsymbol{numJobs}}{\boldsymbol{numMachines}}$



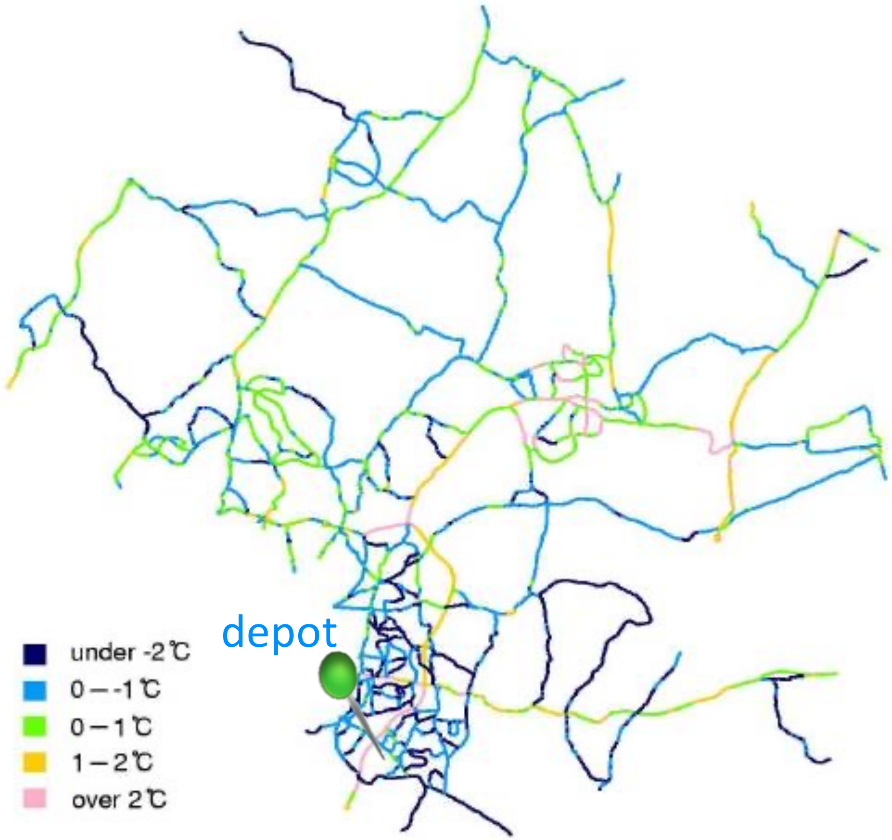


Yi Mei, Mengjie Zhang, "A Comprehensive Analysis on Reusability of GP-Evolved Job Shop Dispatching Rules," *IEEE World Congress in Computational Intelligence (WCCI)*, Vancouver, Canada, 2016.

GPHH for Evolving Heuristics for Arc Routing Problem

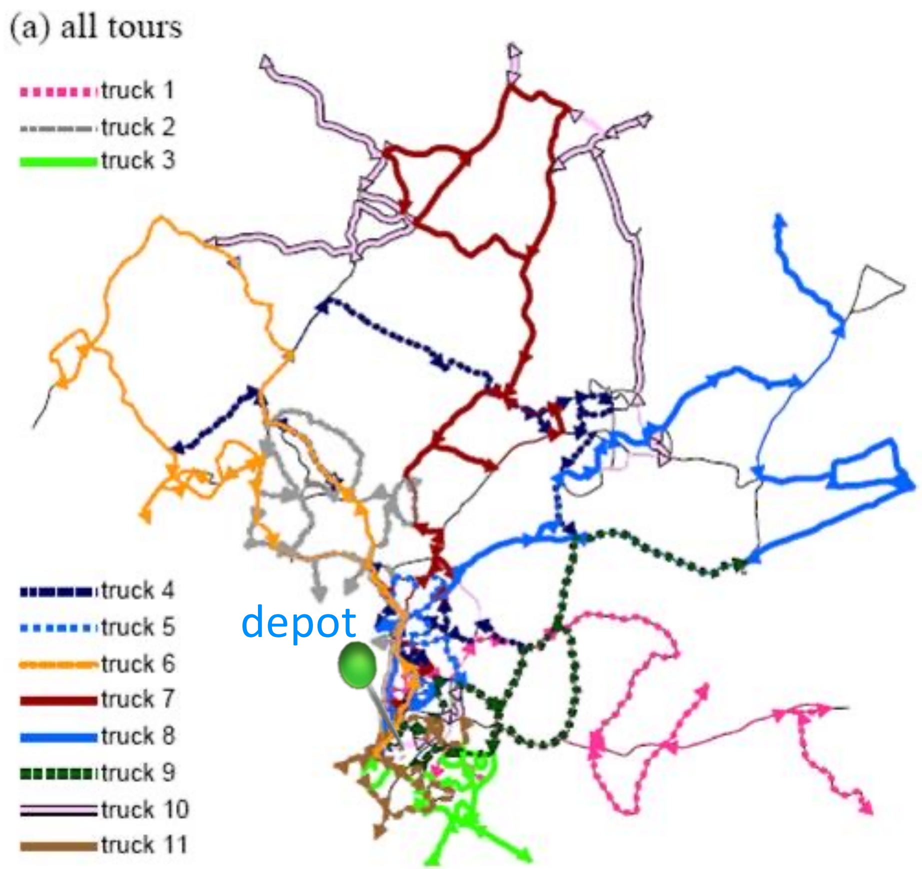
Arc Routing Problem

- A Graph
  - A set of arcs to be served (tasks)
  - A special node (depot)
- Arc
  - Demand
  - Serving cost
  - Deadheading cost
- A fleet of vehicles
  - Capacity



Arc Routing Problem

- A solution
  - A set of **routes** to serve the tasks
- Objective
  - Minimize the **total cost**
- Constraints
  - Each task is served exactly once
  - Each vehicle starts and ends at the depot
  - The total demand served by each vehicle cannot exceed its capacity



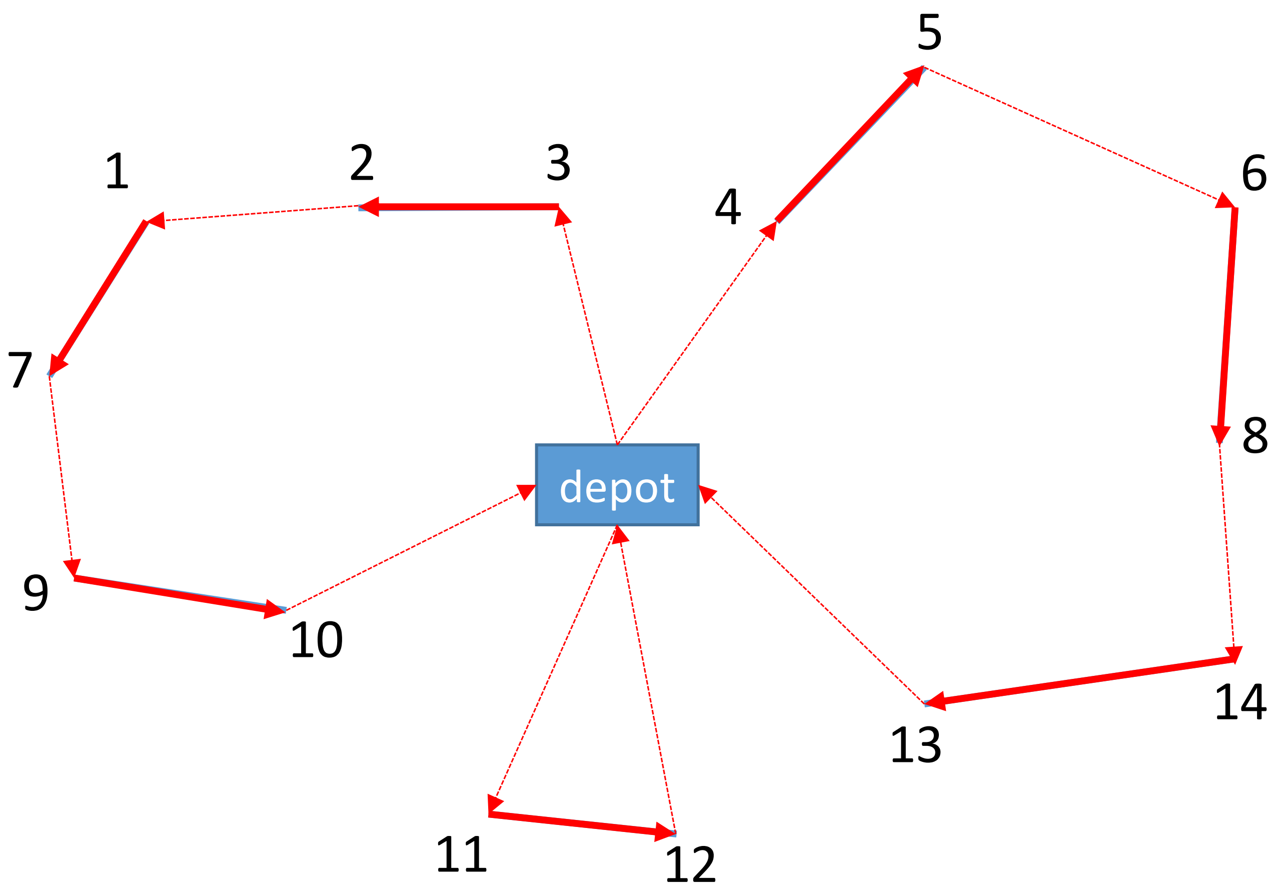
Developmental CARP Solving

- A single vehicle, but can go back to refill

- **Meta-algorithm**
  - **Step 0**: A vehicle at the depot, all tasks unserved;
  - **Step 1**: Select an unserved task by the **heuristic function;**
  - **Step 2**: If the vehicle can serve the task without violating the capacity constraint, then go; otherwise go back to the depot to refill;
  - **Step 3**: If all the tasks have been served, then go back to depot and stop; otherwise go to Step 1;

Weise, T., Devert, A. and Tang, K., 2012, July. A developmental solution to (dynamic) capacitated arc routing problems using genetic programming. In *Proceedings of the 14th annual conference on Genetic and evolutionary computation* (pp. 831-838). ACM.

Developmental CARP Solving



Decisions

Go to 3, serve <3,2>

Go to 1, serve <1,7>

Go to 9, serve <9,10>

Go back to depot

Go to 4, serve <4,5>

Go to 6, serve <6,8>

Go to 14, serve <14,13>

Go back to depot

Go to 11, serve <11,12>

Go back to depot

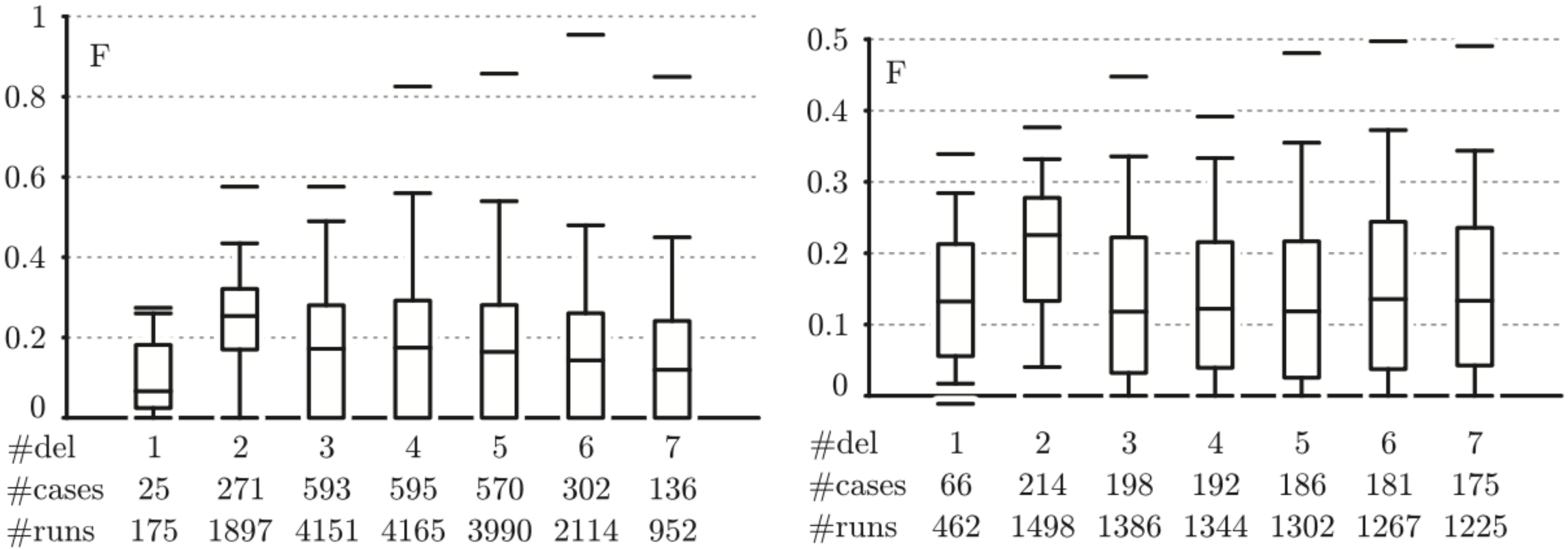
Evolve Heuristic Function to Make Decisions

- Standard GP
  - A single tree to calculate heuristic value
  - Select the task with the lowest heuristic value

Terminal	Description
Demand(e)	Demand of the task e
Load	Remaining load of the vehicle / capacity
Cost(e)	Cost of the task e
DepotCost(e)	Cost to go back to depot from task e
Satisfied	Fraction of satisfied (served) tasks
Last(e)	Heuristic value calculated in the last round

Functions
+, -, *, /, max, exp, sin, angle

Results



- **Outperform existing heuristics in uncertain environment**

Open Issues

- Generalisation
- Dynamic problems (new tasks arrive in real time)
- Multiple vehicles serving simultaneously
- Better meta-algorithms
- Interpretability

GPHH for Evolving Heuristics for Memetic Algorithm in TTP

Traveling Thief Problem

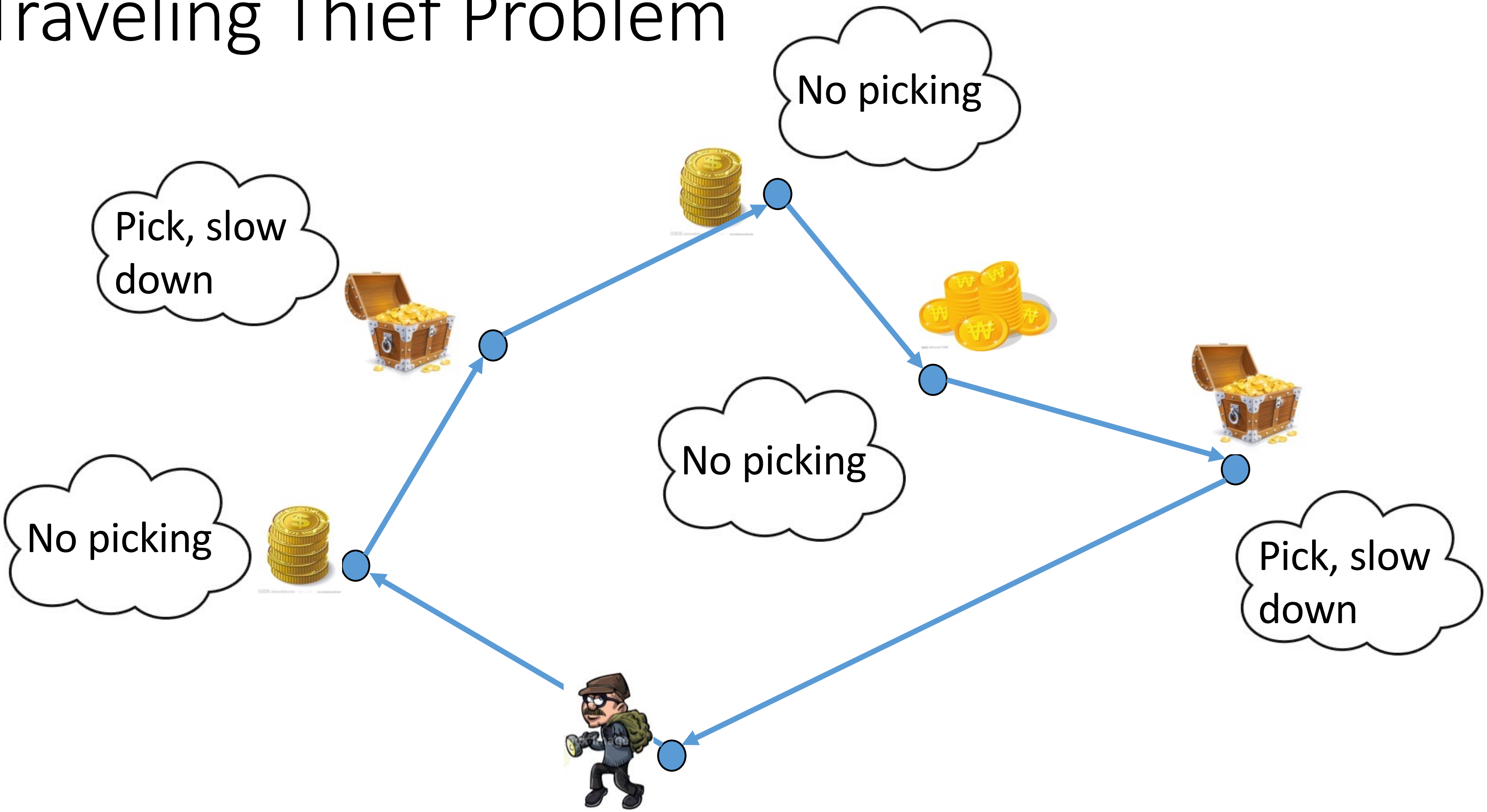
- A new benchmark problem for studying **interdependent components**
- A combination of TSP and KP

- A set of **cities**
- Each city has an **item**
- Each item has a **value** and a **weight**
- A thief with **capacity** and a **speed** depending on weight carried

- Visit all the cities and collect some items to maximise profit

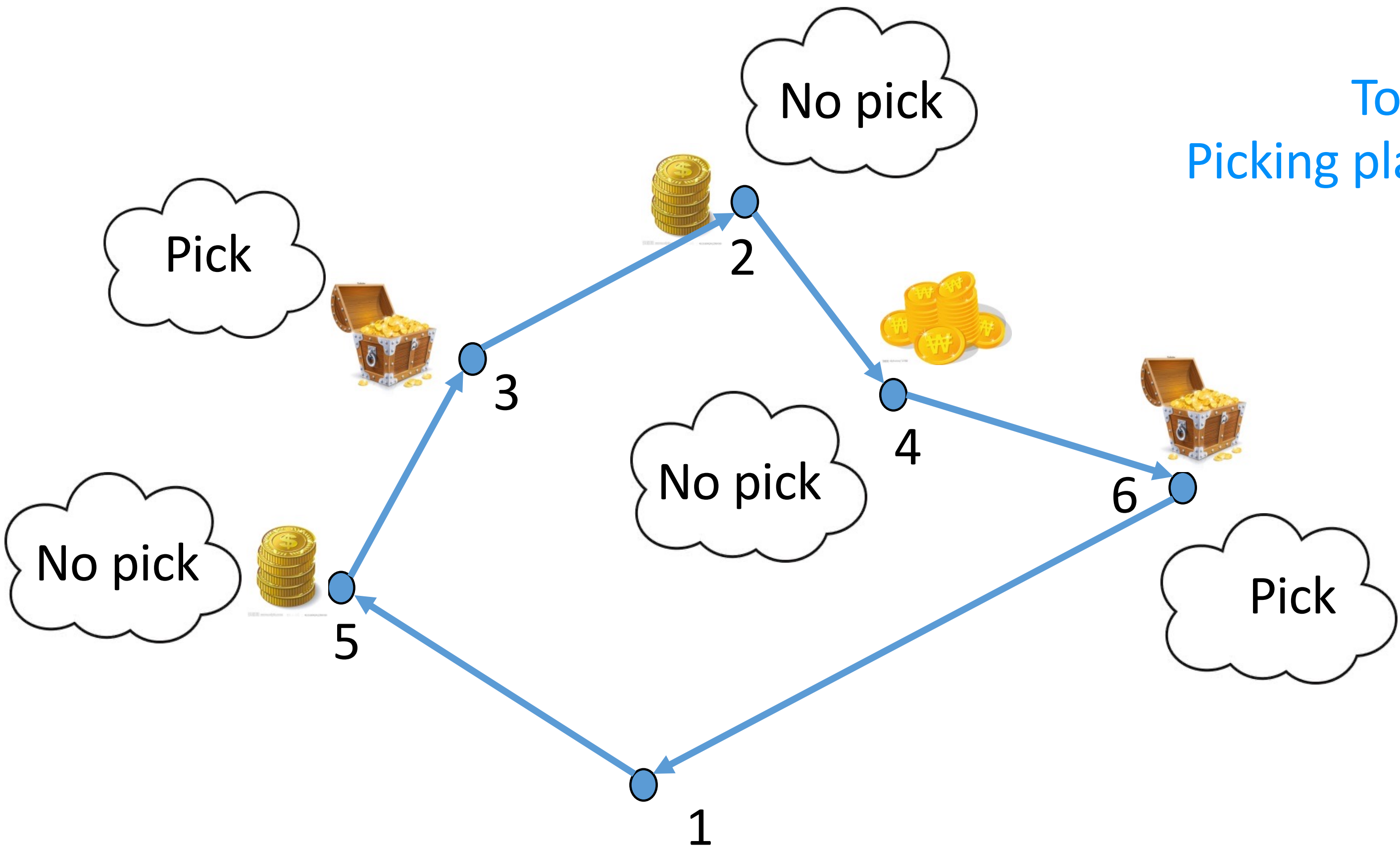
$$\sum_{i \in selected} c_i - \alpha \cdot T$$

Traveling Thief Problem



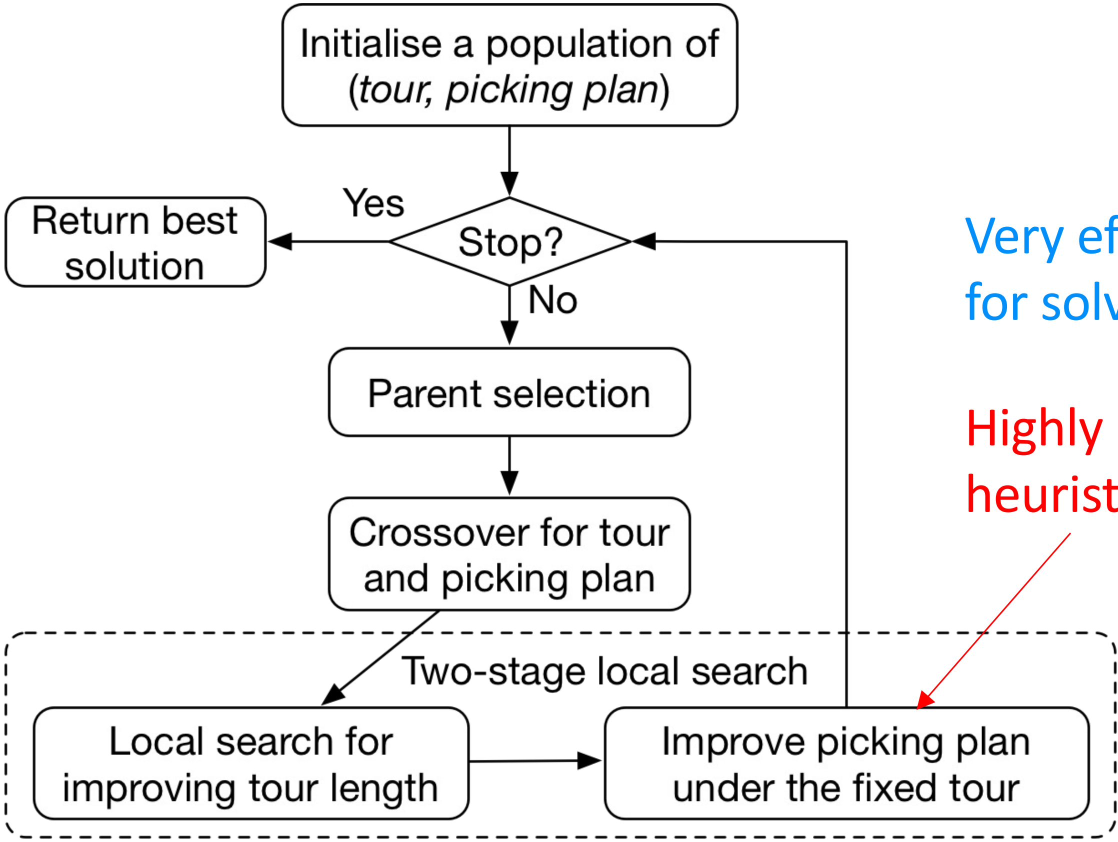
Traveling Thief Problem

- A solution contains a **TSP tour** and a **picking plan**



Tour: (1,5,3,2,4,6,1)
Picking plan: (0,0,1,0,0,1,0)

Memetic Algorithm for TTP



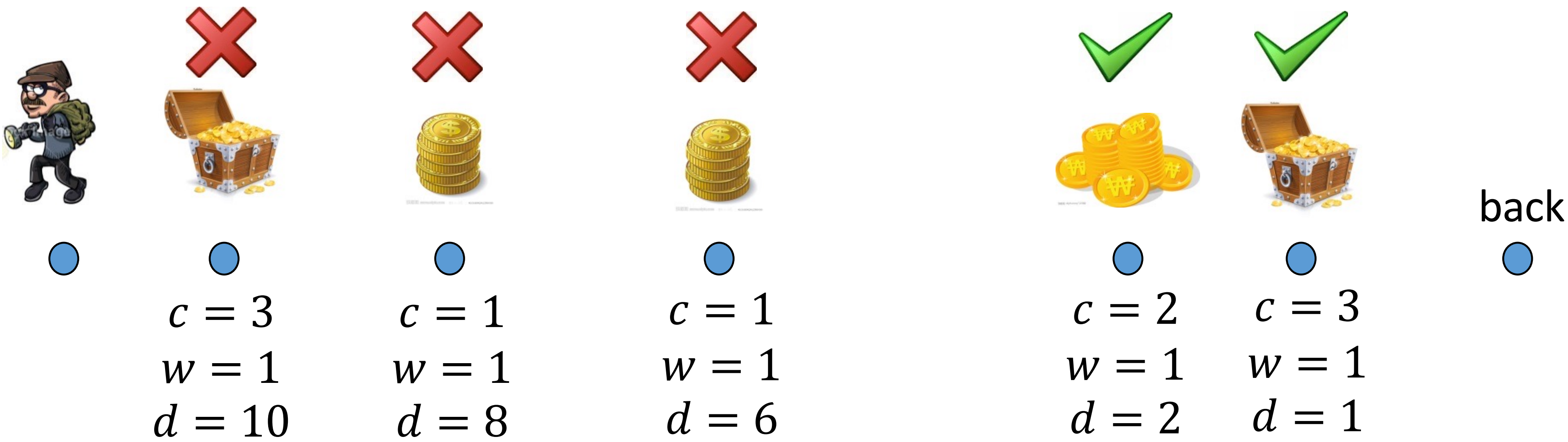
Very effective especially for solving large scale TTP

Highly dependent on heuristic

Mei, Y., Li, X. and Yao, X., 2014, December. Improving efficiency of heuristics for the large scale traveling thief problem. In *Asia-Pacific Conference on Simulated Evolution and Learning* (pp. 631-643). Springer International Publishing.

Item-Picking Heuristic Given Tour

- Different from conventional knapsack heuristics
- The efficiency of an item depends on
  - $\frac{\boldsymbol{value}}{\boldsymbol{weight}}$
  - **Distance** from where it is to the starting city (not to slow down too early)

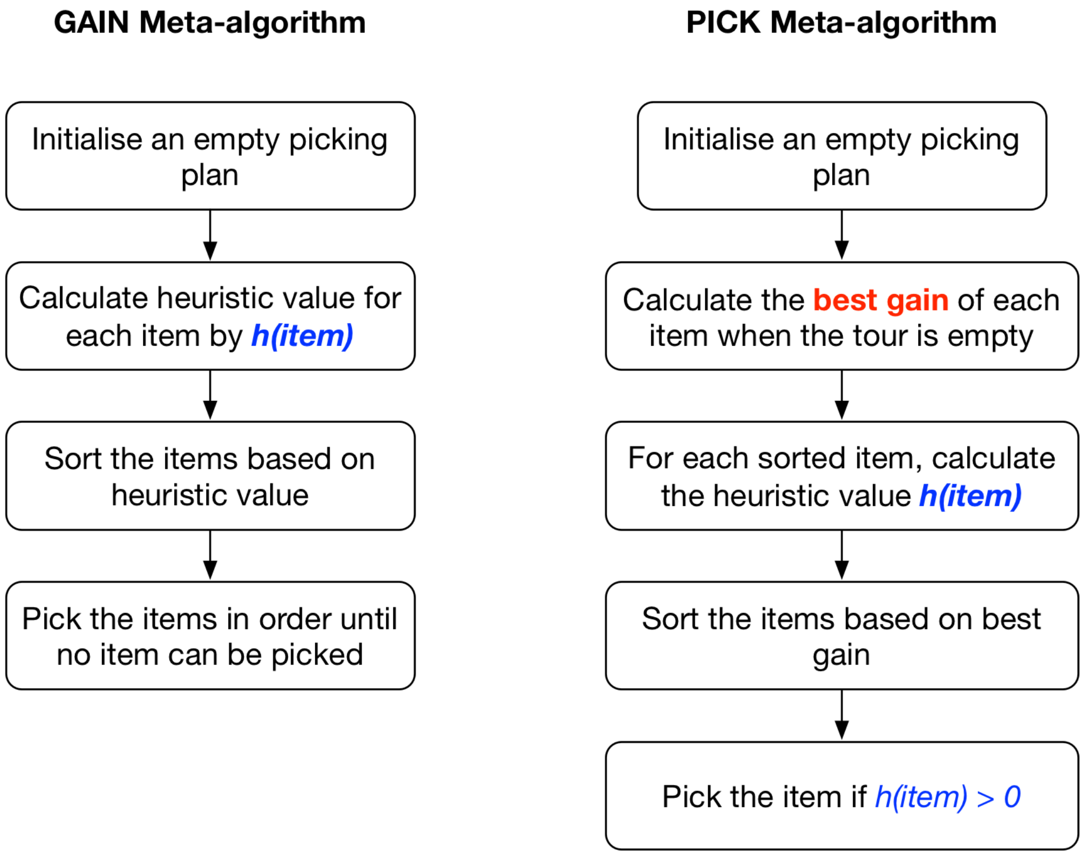


Item-Picking Heuristic Given Tour

- A very sophisticated heuristic
  - Step 0: All items not picked, current load of the tour is zero;
  - Step 1: For each item, calculate the **best gain** when the tour is empty;
  - Step 2: Sort the item in the decreasing order of the best gain;
  - Step 3: For each sorted item, if feasible and **expected gain under the current load of the tour** is positive, then pick the item and update the current load of the tour
  - Step 4: If all sorted item is scanned, stop; otherwise go to the next sorted item;

- Complex calculation formulas for the best gain and expected gain
- Evolve using GP

Item-Picking Heuristic Given Tour



Mei, Y., Li, X., Salim, F. and Yao, X., 2015, May. Heuristic evolution with genetic programming for traveling thief problem. In *2015 IEEE Congress on Evolutionary Computation (CEC)* (pp. 2753-2760). IEEE.

Item-Picking Heuristic Given Tour

- Evaluation Model
  - Three small TTP instances
  - Run MA with the heuristic function once, and get the best solution
  - Fitness of the heuristic = the fitness of this best solution

Terminal	Description
profit	Profit of the item
weight	Weight of the item
bdist	Distance of the location of the item to the end of the tour
Q	The capacity of the knapsack
L	The total length of the tour
R	The rent ratio of the knapsack
ν	The coefficient defined by Eq. (2)
$v_{\max}$	The maximal speed
W	The total weight of the items selected so far

Functions
+, -, *, /

Item-Picking Heuristic Given Tour

- Very similar performance as the manually designed heuristic

Name	n	m	TSMA	TSMA-GAIN	TSMA-PICK
brd14051	14051	140500	2.66e+7(2.07e+5)	2.66e+7(2.59e+5)	2.66e+7(3.41e+5)
d15112	15112	151110	2.85e+7(5.35e+5)	2.83e+7(3.40e+5)	2.84e+7(4.75e+5)
d18512	18512	185110	3.07e+7(2.73e+5)	3.07e+7(3.47e+5)	3.06e+7(4.95e+5)
pla33810	33810	338090	6.34e+7(4.59e+5)	6.32e+7(6.87e+5)	6.33e+7(5.11e+5)
rl11849	11849	118480	1.97e+7(8.29e+4)	1.97e+7(9.06e+4)	1.97e+7(1.04e+5)
usa13509	13509	135080	2.92e+7(2.60e+5)	2.92e+7(2.55e+5)	2.92e+7(1.70e+5)

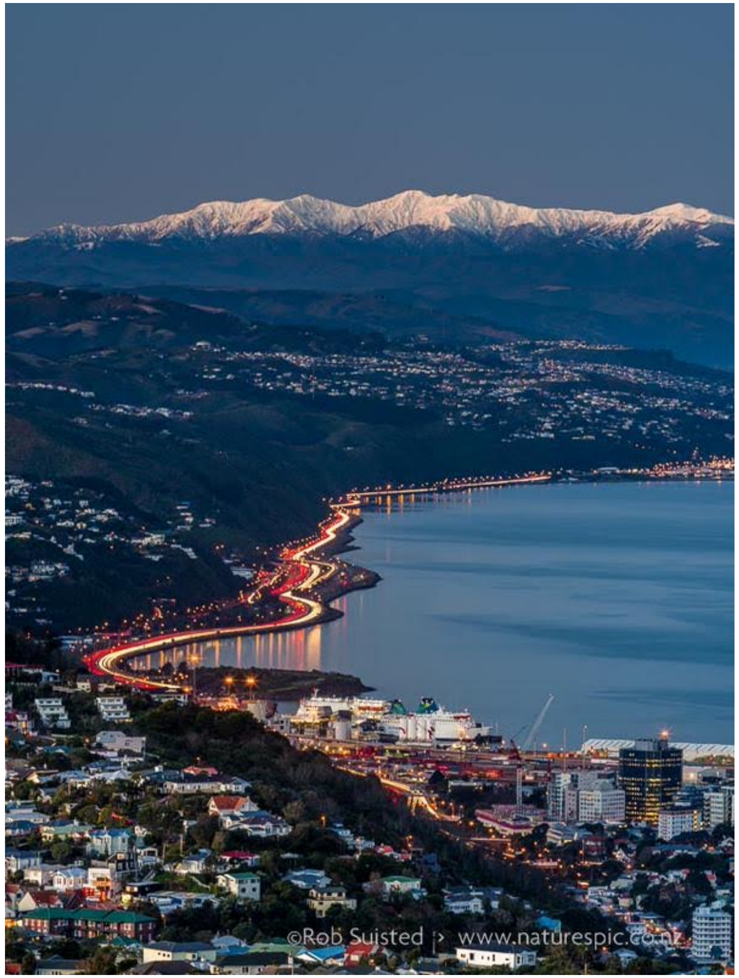
Conclusion

- Genetic Programming has been successfully used as a hyper-heuristic for automatically designing heuristics
- Very useful in combinatorial optimisation, where heuristics are usually needed for decision making
- Especially powerful in dynamic environment, in which immediate response is needed
- Many open issues to be addressed
  - Representation
  - Evaluation model
  - Generalisation
  - Interpretability
  - …

We're Looking for PhD Students!

- **5-8** fully funded PhD scholarships
- Supported by The Royal Society of NZ's **Marsden Fund** (the most prestigious in NZ, <8% success rate)
- **$23,500-27,500** NZD/year for up to 3 years
- Coolest Little Capital in the world
- Research **No. 1** in NZ (2015)
- Closing date: **1 March 2017**.

TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI
VICTORIA
UNIVERSITY OF WELLINGTON



We’re Looking for PhD Students!

- Research Areas
  - Evolutionary Scheduling, Routing and Combinatorial Optimisation
  - Evolutionary Feature Selection and Dimensionality Reduction
  - Evolutionary Web Service Composition and Resource Allocation
  - Evolutionary Image Analysis and Pattern Recognition
  - Evolutionary Machine Learning and Transfer Learning
  - Genetic Programming, PSO, Learning Classifier Systems
- More details
  - https://ecs.victoria.ac.nz/Groups/ECRG/ResearchAreas#Areas
- Contact
  - Dr Yi Mei: yi.mei@ecs.vuw.ac.nz
  - Dr. Bing Xue: bing.xue@ecs.vuw.ac.nz
  - Prof. Mengjie Zhang: mengjie.zhang@ecs.vuw.ac.nz

We’re Looking for PhD Students!

- Requirement
  - First class Honours or Masters degree in Computer Science or Statistics/Operations Research (GPA > 3.5/4.0)
  - Research experience/publications in EC, combinatorial optimisation, …
  - Strong programming skills in Java, Python, R, …