Evolutionary Computation and Learning

Genetic Programming 2:
Automatic Algorithm/Heuristic Design

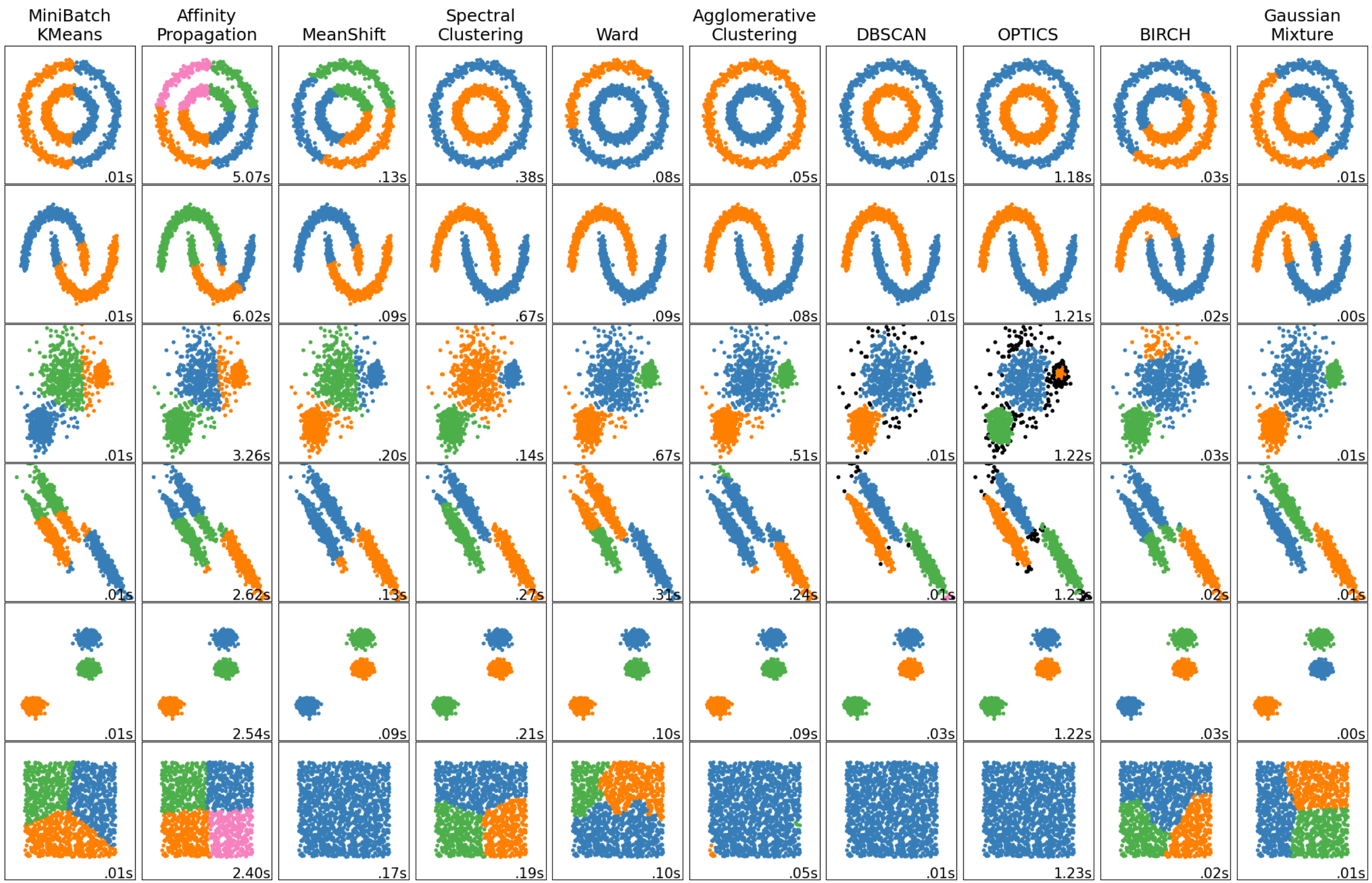
Represented By: DR. Vahid Ghasemi

Outline

- GA to GP
  - Representation
  - Evolutionary operators
- Genetic Programming
  - Terminal set
  - Function set
  - Fitness function
  - GP Parameters
  - Stopping criterion
- GP for regression
- GP for classification
- GP for automatic heuristic design
  - Design steps
  - Packing
  - Scheduling
- Project 1

Clustering

- Clustering is an unsupervised learning task
  - Group the data points/objects without the class label
  - Many different clustering algorithms



K-means Clustering: Algorithm

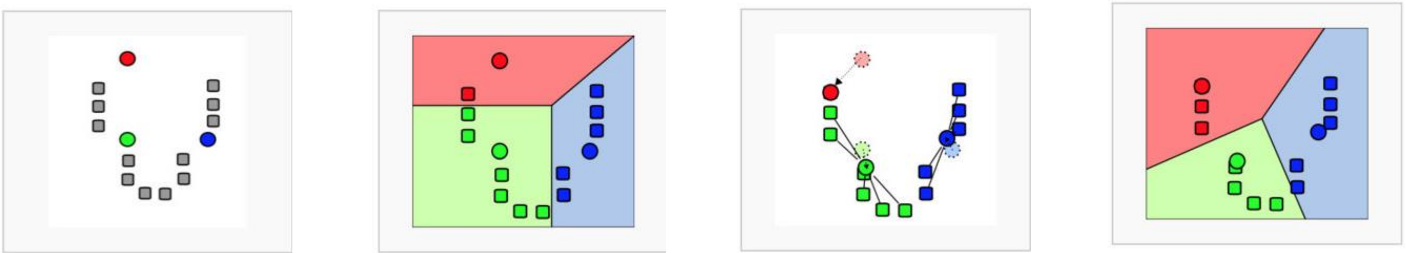
1. Initialise k initial “means” randomly from the data set

2. Create k clusters by assigning every instance to the nearest cluster: based on the nearest mean according to the distance measure

3. Replace the old means with the centroid (mean) of each cluster

4. Repeat the above two steps until convergence (no change in each cluster centroid).

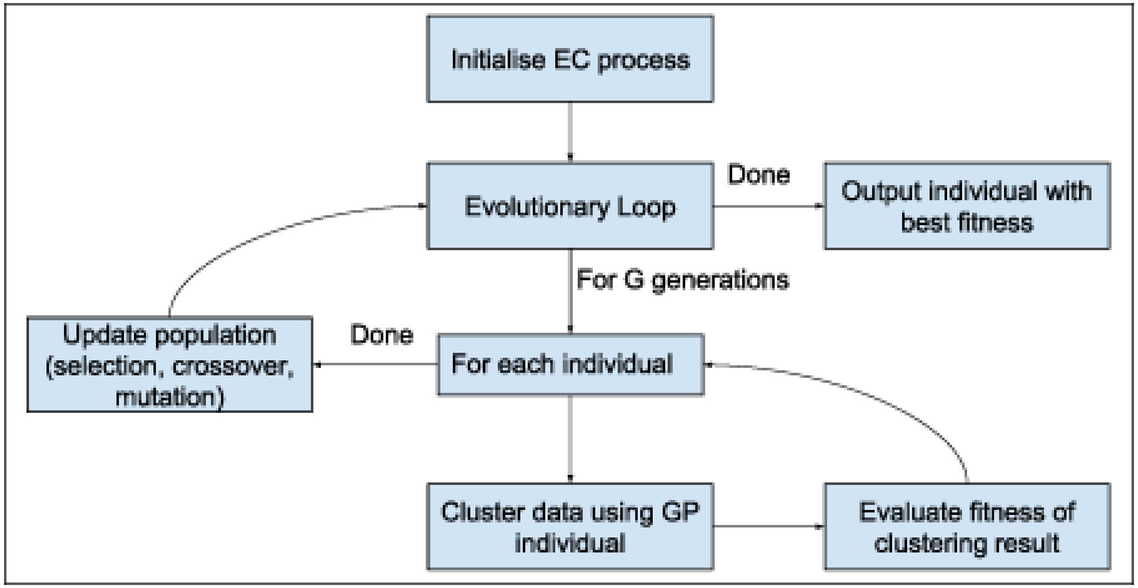
- *Centroid is not an instance*

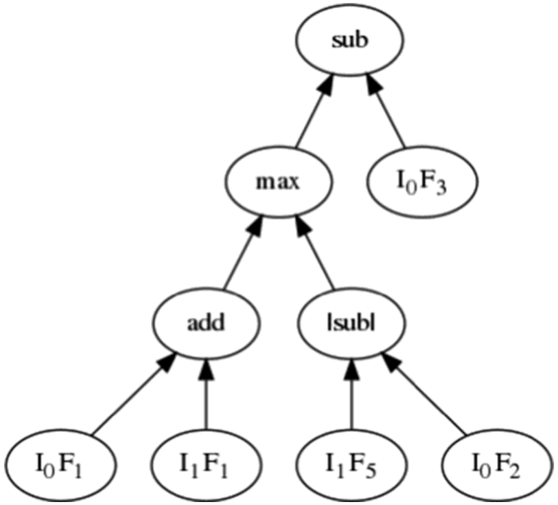


Clustering

- Clustering methods need similarity/distance measure
  - Remove redundant features
  - Normalise features with different scales
- Euclidean/Manhattan distance measures are not flexible enough

- Use GP to learn similarity/distance measure
  - How to evaluate a GP individual (similarity measure)?
  - How to cluster data using a GP individual?



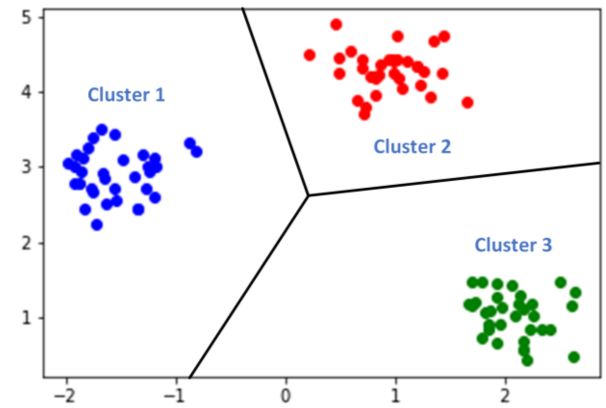


GP for Clustering

- **Fitness evaluation**: similar to wrapper-based feature selection
  - Use a clustering method based on the similarity/distance measure to cluster the training data

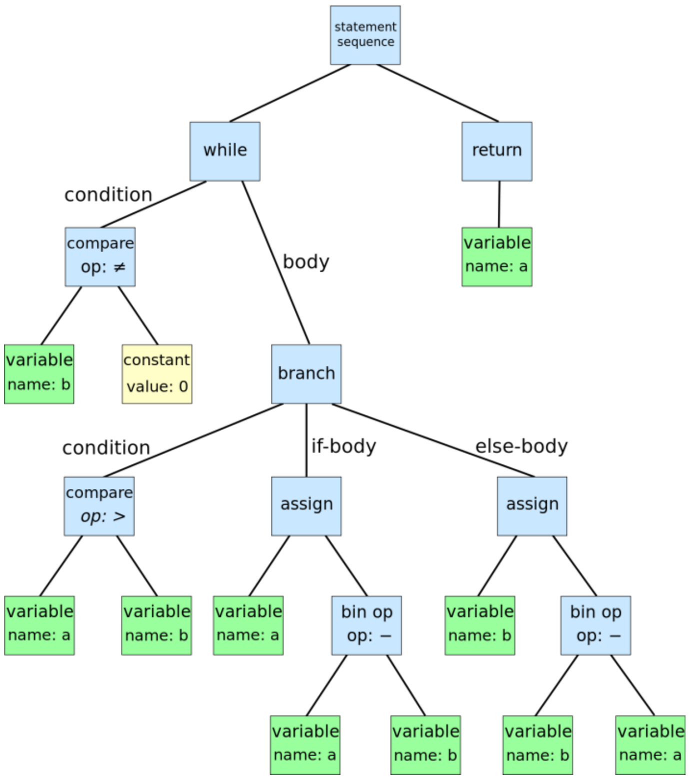
```
Input: a GP individual (similarity measure) X, number of neighbours k
Output: Clusters

Edges = {}
for each point I in training data do
  Get the k nearest neighbour of I in training data based on Euclidean distance
  Get the nearest neighbour N(I) of I from the k nearest neighbours based on X;
  Add an edge (I, N(I)) into Edges
Do clustering of the graph based on Edges
return Clusters
```

  - Use a cluster quality metric as the fitness
    - Compactness, Separability, …
    - A combination of them
      e.g., smaller distance between instances in the same cluster, larger distance between clusters)



Automatic Algorithm/Heuristic Design

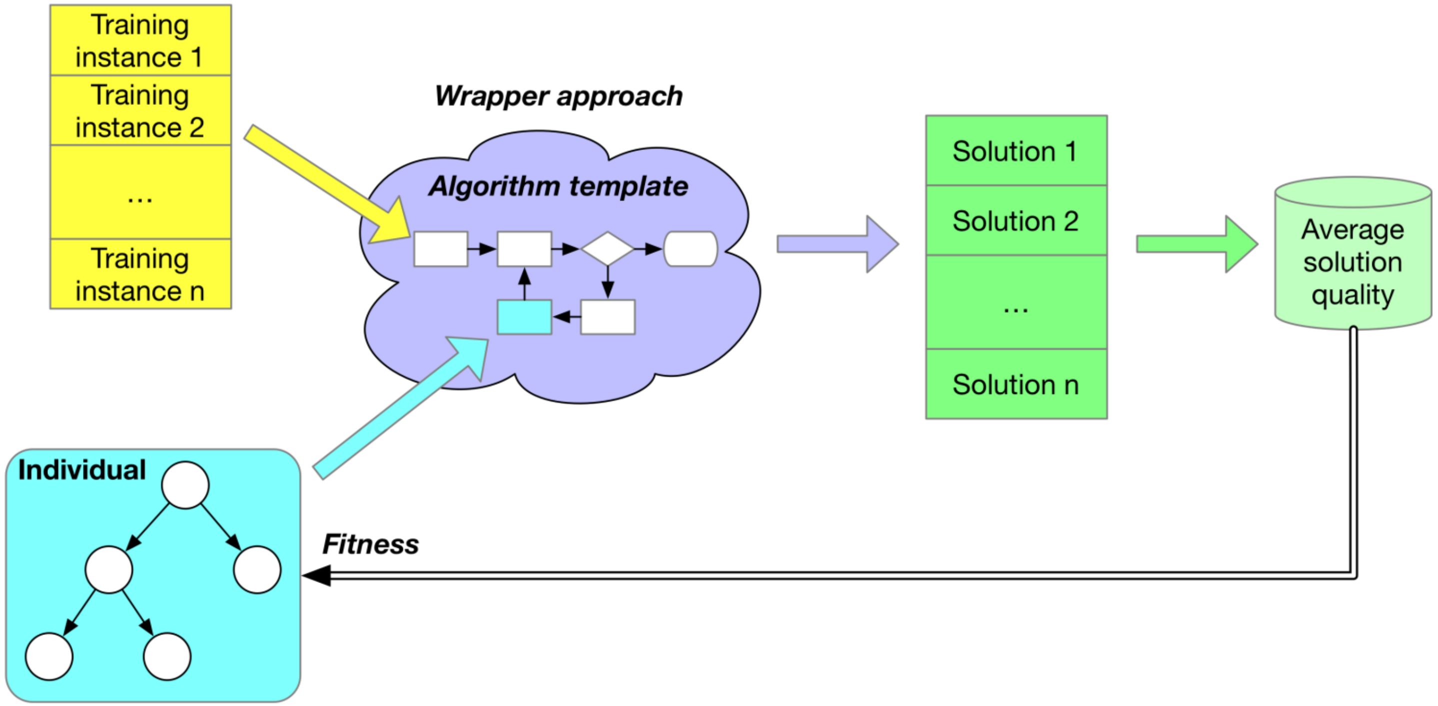
- GP evolves programs, and programs are essentially algorithms
- The whole algorithm is too complex to evolve, GP can evolve a part of the algorithm (heuristic)
  - The algorithm becomes a template
  - A part can be replaced by GP individuals
  - Like wrapper-based approach
- **Key issue**:
  - Fitness evaluation (how good a heuristic is)
- The process of evolving heuristics is called **hyper-heuristic**
  - Search in the heuristic space rather than solution space
  - knapsack problem, solution: specific items
    heuristic: the rule to choose item, e.g., higher value first




```
while b ≠ 0
  if a > b
    a := a − b
  else
    b := b − a
return a
```

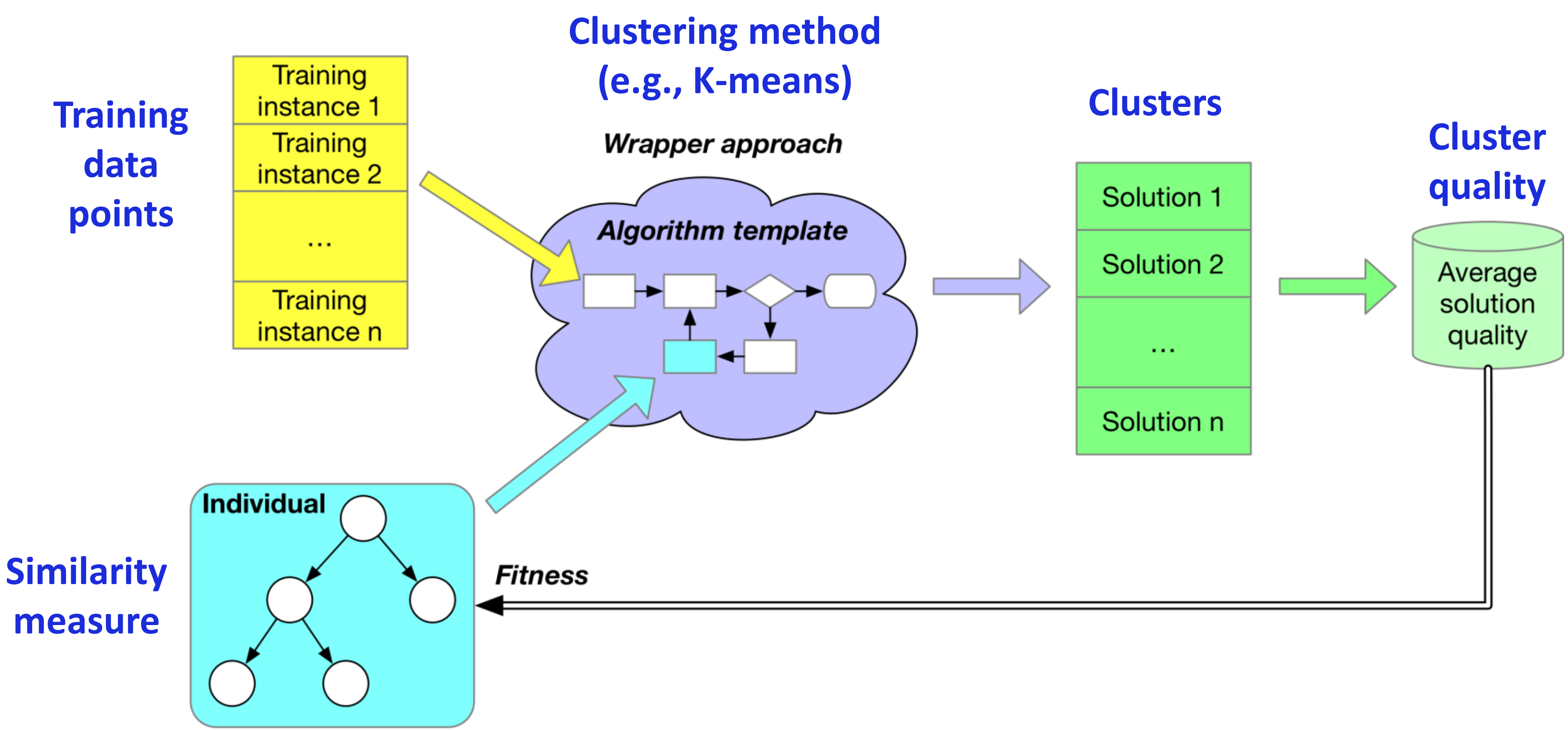
GPHH for Automatic Heuristic Design

- **Hyper-heuristic:** search in the heuristic space rather than solution space
- **Fitness evaluation**:
  - Given a set of training instances/cases, and a GP individual (heuristic)
  - Replace the part in the algorithm template with the GP individual
  - Run the resultant algorithm on each training instance, to get solutions
  - Fitness is set to the aggregated solution quality



GPHH for Automatic Heuristic Design

- **Example 1: Clustering**
  - Heuristic: similarity measure
  - Training instance: a training dataset to be clustered
  - Wrapper approach/template: A clustering method
  - Solution: obtained clusters
  - Solution quality: cluster quality

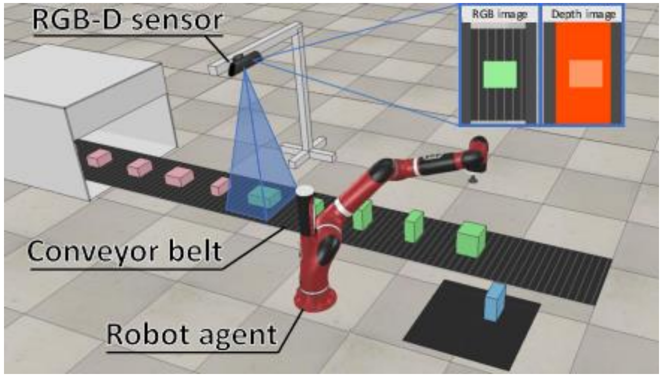


Steps of GP Hyper-heuristic

- Examine currently used heuristics
  - Understand how human designed and used existing heuristics
- A framework for the heuristics to operate in
  - Commonalities among the existing heuristics
  - Summarise into a template
  - Can be a simulation/decision making process
- Decide on the terminal set
  - Which features/attributes are useful for the heuristics?
- Decide on the function set
  - How to combine the features/attributes?
- Identify a fitness function
  - Training instances (consider generalisation)
  - How to aggregate a single fitness from multiple training instances?
- Run the GP

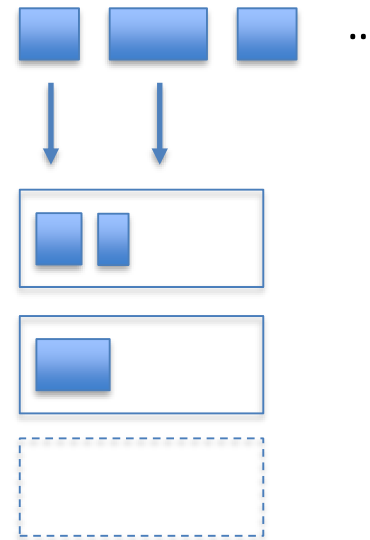
GPHH for Automatic Heuristic Design

- **Example 2: Online Packing heuristic**
- The items come over time, and the robot decides where to place each incoming item (which existing bin, or create a new empty bin)
- Existing manual heuristics: (NOT good enough)
  - **Best-Fit**: put to the fullest bin that can accommodate it
  - **Worst-Fit**: put to the empties bin that can accommodate it
  - **Next-Fit**: put in the last available bin
- GP to evolve bin packing heuristic
  - Heuristic: bin packing heuristic
  - Training instance: a packing problem: a sequence of items and bins
  - Wrapper approach/template: The process of placing items into bins
  - Solution: obtained packing plan
  - Solution quality:
    - ***Wasted space*** of the packing plan
    - Average over all packing problems



GPHH for Automatic Heuristic Design

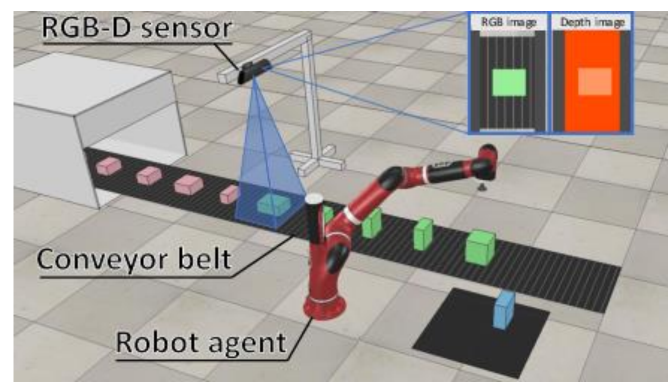
- **Example 2: Online Packing heuristic**
- Terminals: (Designed based on domain knowledge)
  - Load Ratio of the bin (LR)
  - Index of the bin (ID)
  - Predicted next item size?
  - Regret (how much worse if not put here)?
  - …
- Functions: {+, -, *, / (protected), max, min, …}
- Manual heuristics can be represented by the terminals



Best-Fit

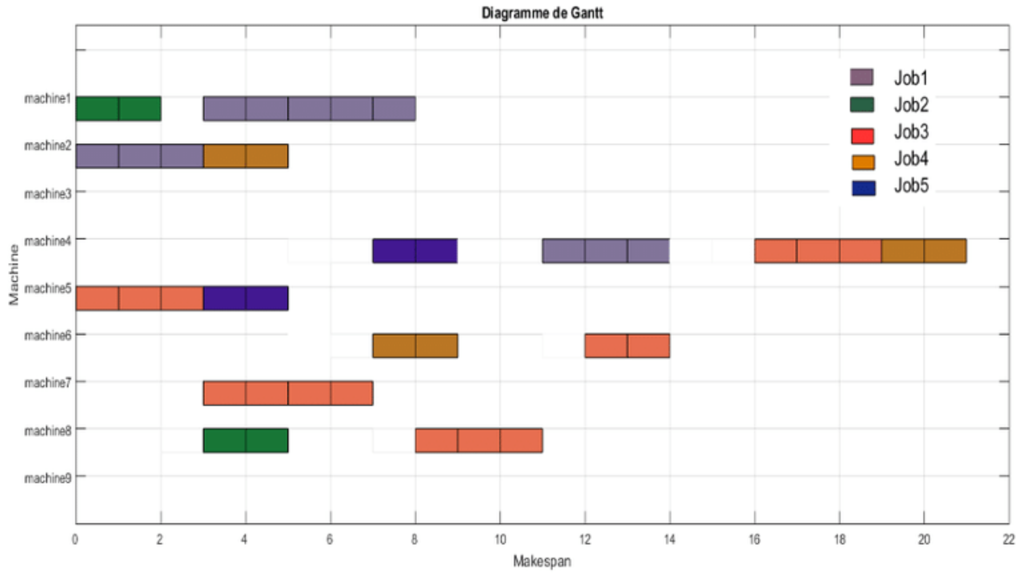
Next-Fit

- How about **Worst-Fit**?



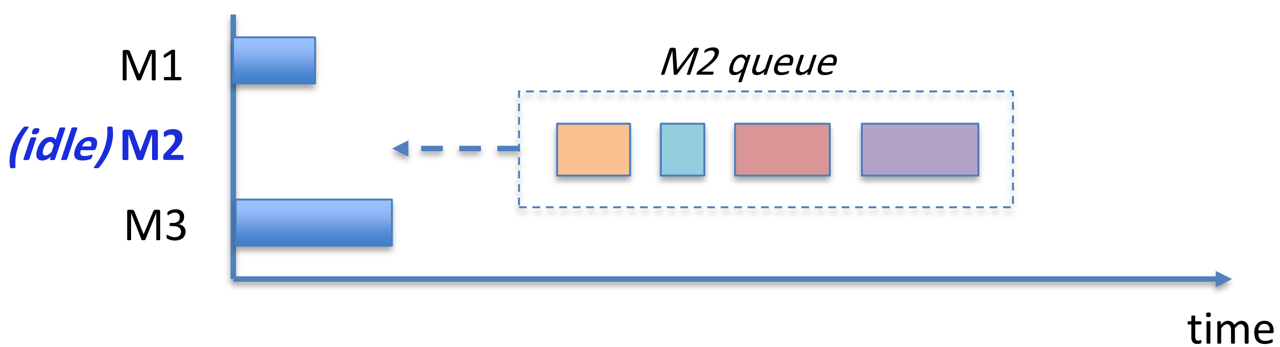
GPHH for Automatic Heuristic Design

- **Example 3: Scheduling rules (dispatching rules)**
- **Dynamic Scheduling**: process a set of jobs by a set of machines
  - Decide the start time of each job on each machine
  - Resource and precedence constraints
  - A job is unknown before it arrives (e.g., job 4 arrives at time 2)
- Existing manual dispatching rules: (NOT good enough)
  - First come first serve
  - Shortest processing time
  - Earliest due date



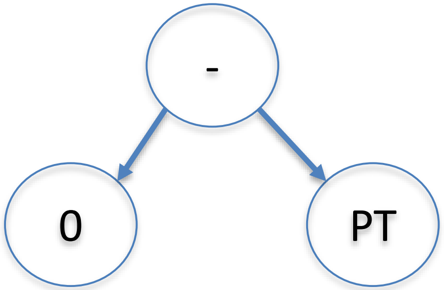
GPHH for Automatic Heuristic Design

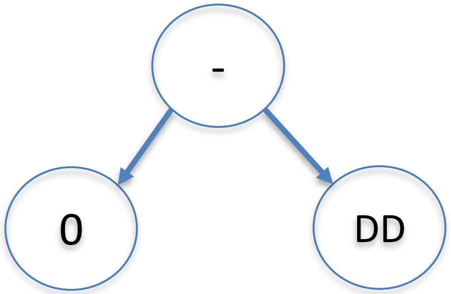
- **Example 3: Scheduling rules (dispatching rules)**
- GP to evolve dispatching rules
  - Heuristic: dispatching rule (**priority function**)
  - Training instance: a scheduling problem: job arriving over time, a set of machines
  - Wrapper approach/template: The process of schedule the jobs to the machines
    - When a machine becomes idle, use the dispatching rule to calculate the priority value of each job in its queue
    - Process the job with the highest priority next
  - Solution: obtained schedule
  - Solution quality: makespan/tardiness of the obtained schedule, average over all scheduling problems



GPHH for Automatic Heuristic Design

- **Example 3: Scheduling rules (dispatching rules)**
- Terminals:
  – Processing Time of the operation (PT)
  – Due date of the job (DD)
  – Workload in the machine’s queue (WIQ)
  – …
- Functions: {+, -, *, /, max, min, …}
- What are these rules?
  – Can we make them simpler?

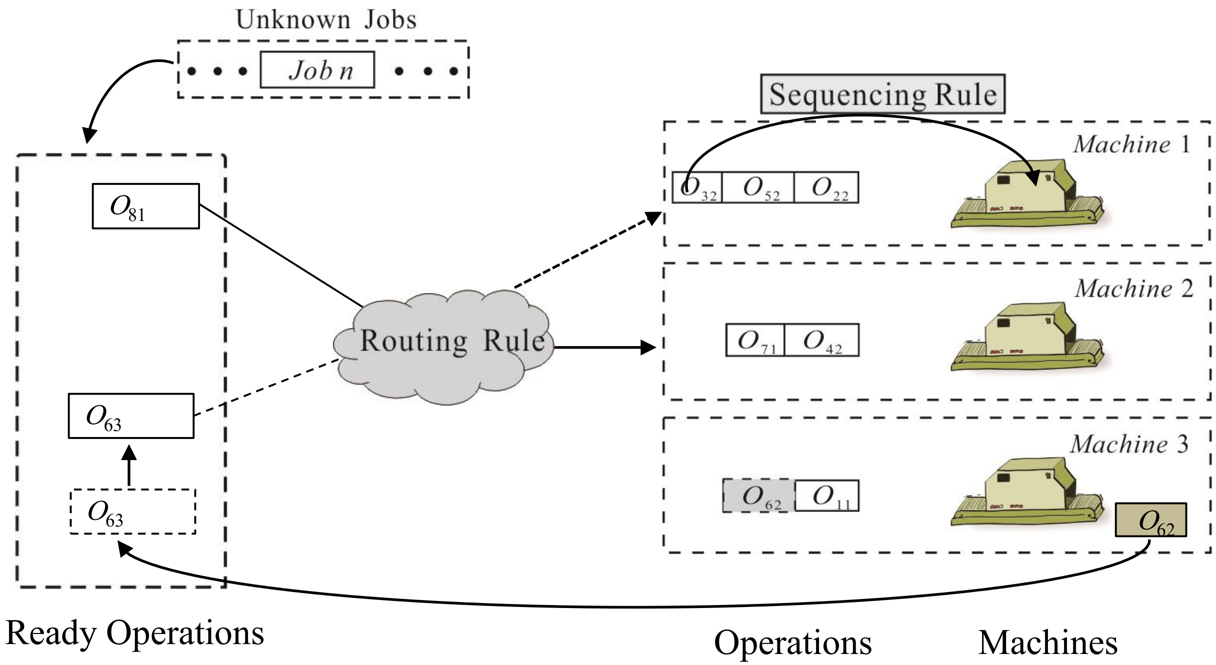




A. Shortest processing time B. Earliest due date

GPHH for Automatic Heuristic Design

Scheduling Heuristics for Dynamic Flexible Job Shop Scheduling



GPHH for Automatic Heuristic Design

- Aggregating multiple results into a single fitness
  - Training scheduling instance 1: 10 jobs, 10 machines, makespan ranges from 100~500
  - Training scheduling instance 2: 100 jobs, 20 machines, makespan ranges from 2000~10000
  - How to aggregate? Normalise for each instance:
    - $fit = mean_{inst}(q_{norm}(inst))\ ,\ q_{norm}(inst) = \frac{q_{orig}(inst)}{q_{ref}(inst)}$
    - q_{ref} can be the lower bound, or obtained by a reference rule (e.g., Min-Max normalization)
- Training can be time consuming
  - Use a lot of rules to generate a lot of solutions (simulations)
  - Speed up training
    - Batch learning (a small number of training instances per generation, and change the subset for each generation)
    - Surrogate: learn a fast approximate model for the evaluation

Summary

- GP can evolve heuristics
  - GP Hyper-heuristic
- GPHH for automatic heuristic design
  - Packing heuristic
  - Dispatching rule for scheduling
- Key:
  - Terminal and function sets (include the manual rules as special individuals)
  - Template/Framework for heuristics to work in
    - Simulation
    - Solution construction procedure
  - Fitness function
    - Training set
    - Normalisation

- **Next Lecture**: advanced GP

— Gradient Descent in GP
— Strongly Typed GP
— Grammar-based GP