# Simulation and Optimization for Decision Support (G54SOD)

Semester 2 of Academic Session 2017-2018
Dr Dario Landa-Silva
dario.landasilva@nottingham.ac.uk
http://www.cs.nott.ac.uk/~pszjds

## Lecture 8 – Optimization (with heuristics) in Practice

- Interaction Between Simulation and Optimization
  Understand the different ways in which these two methodologies can interact to tackle real-world problems
- Experimental Evaluation of Heuristic Algorithms
  Define sound experimental setting to evaluate the performance of heuristic algorithms

# Optimization Under Uncertainty of Parameter Values

Optimization seeks to find the best solution (or one of them) for a problem which is often specified by a formal model involving parameters, decision variables, objective function and constraints.

The optimization model is deterministic if all parameter values in the model are known and fixed.

Often, optimization models involve parameters that are uncertain or not fixed. Such uncertainty usually arises from aspects of the problem that cannot be controlled or calculated precisely.

The behaviour of uncertain parameters can be 'simulated' by sampling values from appropriate probability distributions.

# Example in a Workforce Scheduling Scenario

A Genetic Algorithm With Composite Chromosome for Shift Assignment of Part-time Employees. Ning Xue, Dario Landa-Silva, Isaac Triguero, Grazziela P. Figueredo. *2018 IEEE Congress on Evolutionary Computation*, July 2018.

This research arises from a current project in collaboration with business partner.

The challenge is to create weekly workforce schedules for a retail store, i.e. assign shifts to workers most of which are part-time.

This optimization problem was modelled in a deterministic manner. Several problems instances were generated based in real-world data collected from one food outlet.

Which characteristics of the problem can be stochastic in the model?

For each shift to be covered in the week, the following is given:

- shift length
- start/end times
- number of required workers
- required skills or roles

The characteristics of this shift assignment problem are:

- workers are paid according to their age and seniority
- each worker usually has multiple skills (roles)
- the total number of shifts in a week can be over 100
- the total number of workers can be up to 50
- the majority of workers are part-time, hence great variability in their availability and preferences
- start time of a shift is flexible
- shift length varies from 3 to 12 hours
- shifts are constructed (stage not part of this problem) following working regulations

The weekly schedules must obey regulations in respect of:

- workers availability in each day
- maximum working hours in a week
- maximum working hours in a day
- maximum working days in a week
- maximum consecutive working days
- required worker skills for given shift
- forbidden shift sequences, e.g. start before 12pm on day k immediately after ending after 8pm on day k – 1
- incompatible workers, e.g. workers that do not get along or workers that are too familiar with each other

Three criteria are used to evaluate the quality of schedules:

- total labour cost
- fair distribution of unpopular shifts among workers
- fair distribution of total weekly working hours among workers

Uncertainty is an intrinsic feature in many real-world scenarios.

Monte Carlo Simulation is usually used to model the behaviour of complex processes in a computer. It can be used to randomly choose representative values for uncertain parameters in an optimization model.

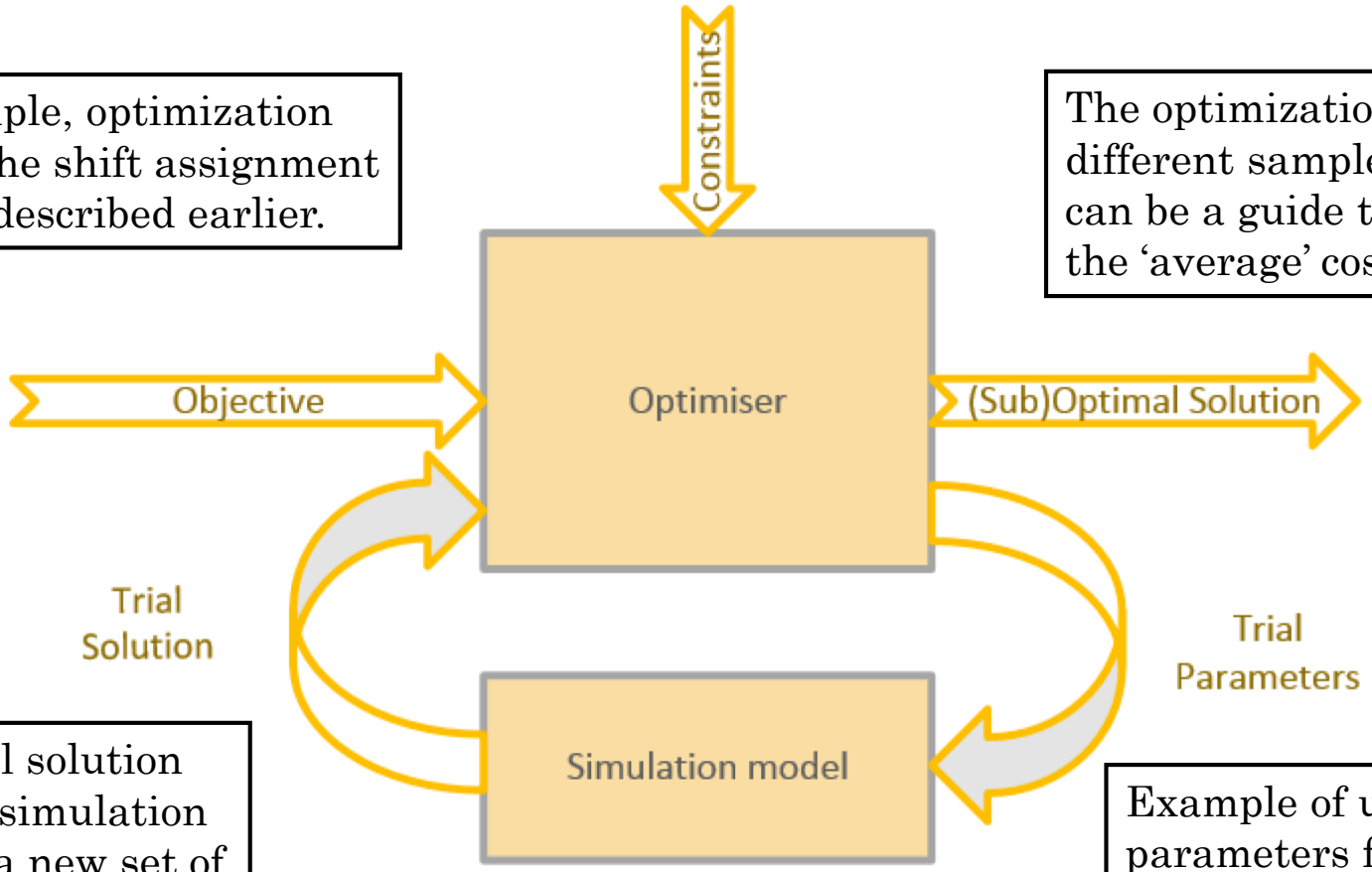
Simulation Optimization using Monte Carlo simulation involves the following steps:

1. identify the parameters that are 'uncertain'
2. replace the 'fixed values' for those parameters by a generator function
3. select a specific probability distribution for each uncertain parameter
4. define correlations between the uncertain parameters as appropriate
5. the optimization is executed by running a number of trials

# Simulation Optimization from Optimization Perspective

For example, optimization to solve the shift assignment problem described earlier.

The optimization of many different sample scenarios can be a guide to optimize the 'average' cost.




Each trial solution from the simulation model is a new set of parameter values for the optimization model.

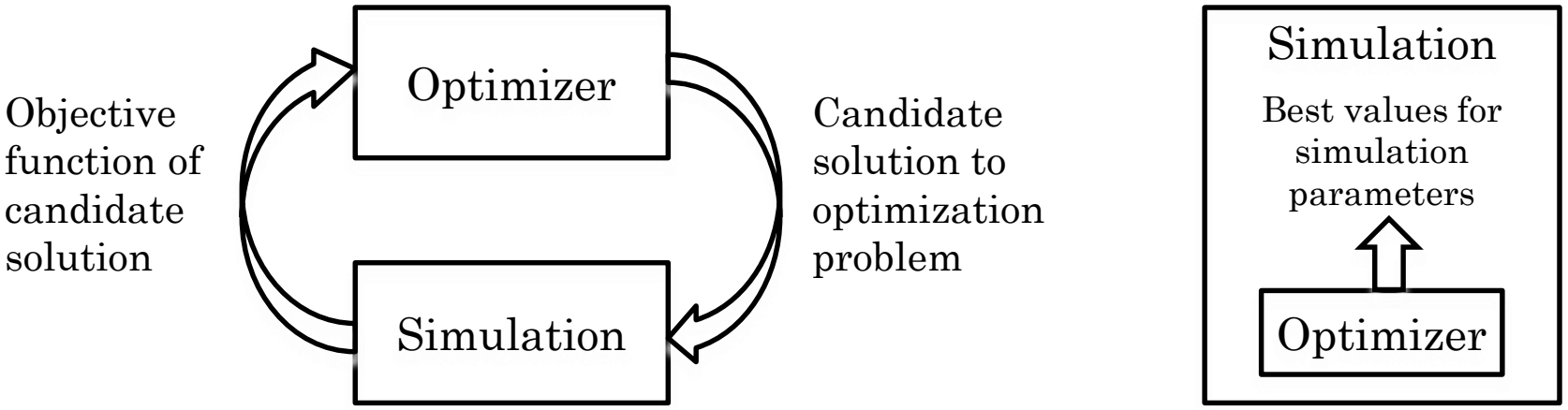
Example of uncertain parameters for trials: number of workers required in each shift, workers availability.

SimHeuristics is a research direction in which simulation models are used to improve the performance of heuristic algorithms.

SimHeuristics are an example of optimization driven approach to deal with uncertainty in the optimization model.

Other forms of interaction between optimization and simulation:

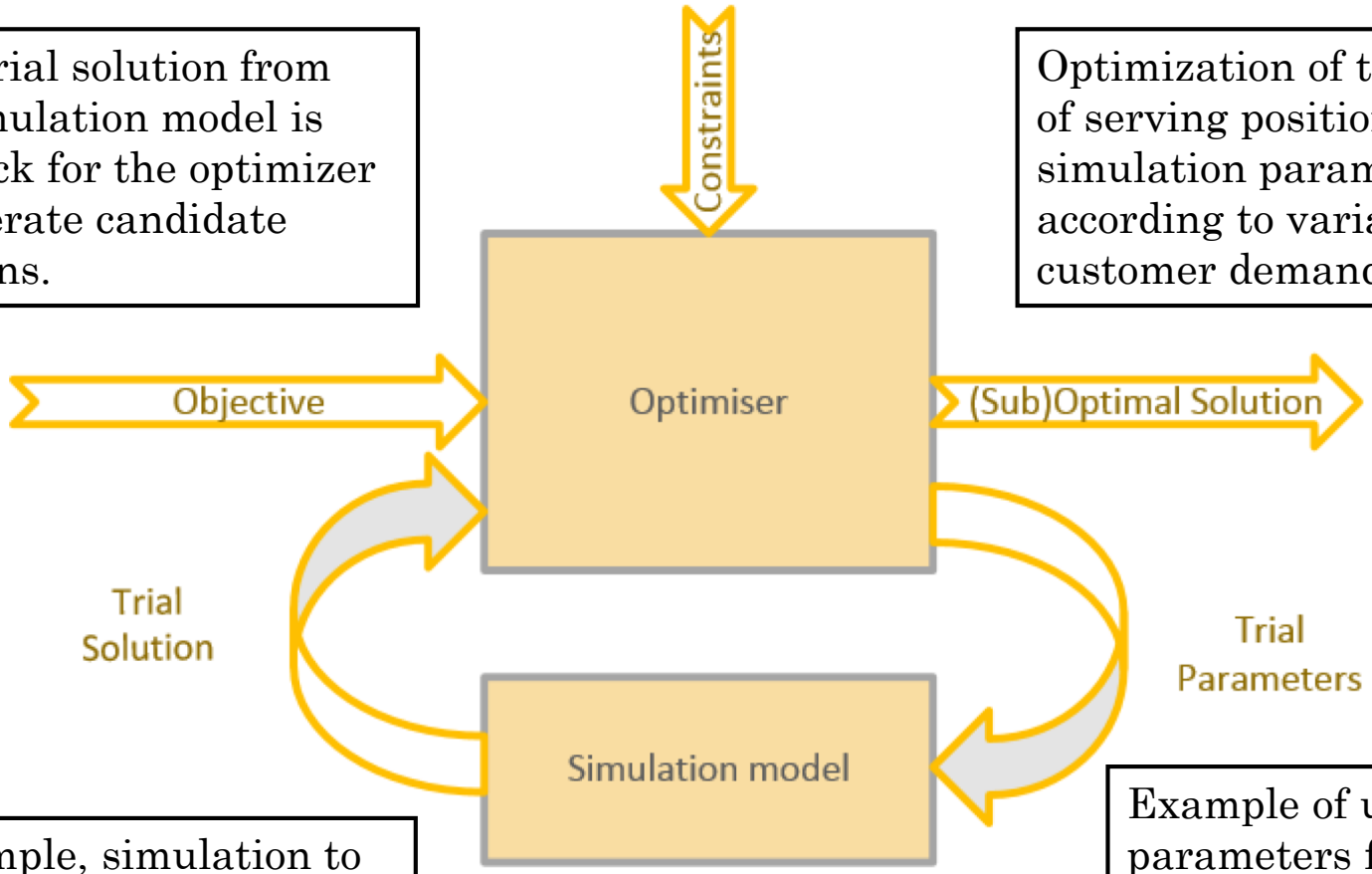
- optimizer uses a simulation to evaluate the quality of solutions
- optimization embedded in a simulation model to set the best values for some parameters in the simulation



# Simulation Optimization from Simulation Perspective

Each trial solution from the simulation model is feedback for the optimizer to generate candidate solutions.

Optimization of the number of serving positions (a simulation parameter) according to variations in customer demand.




For example, simulation to understand the operation of the retail store in terms of customer service.

Example of uncertain parameters for trials: number of workers required in each shift, workers availability.

# Evaluating Heuristic Algorithms

Typically, heuristic performance is evaluated with respect to three aspects: solution quality, computational effort and robustness.

- Quality of the best solution found?
- How long takes to find the best solution?
- How long it takes to find 'good' solutions?
- How robust is the heuristic method?
- How far is the best solution from other 'good' solutions easily found?
- Trade-off between feasibility and solution quality?

Solution Quality. Compare to the optimal solution (if possible), to a tight (upper or lower) bound or to the best known solutions.

Computational Effort. Several aspects can be measured, for example: time to find best solution, total execution time, time per stage (multi-stage method), convergence rate.

Robustness. Components and parameter values of the method should remain constant (or automatically set) for different problem instances.

When designing a computational experiment, it should be decided:

- Which factors to study?
- Which factors to fix?
- Which factors to ignore?

A good experimental design:

- Is unbiased
- Achieves the goals
- Clearly demonstrates performance of the method tested
- Has justifiable rationale
- Generates solid evidence to support conclusions
- Is reproducible

Real problems should be used when possible.

If artificially generated problems are used, these should be archived or its generation should be reproducible.

# Example 1 – Heuristic for Shelf Space Allocation

In this work, a local search heuristic method was developed to tackle the optimization problem of shelf space allocation in small retails shops.

The purpose of the experimental evaluation was to assess:

- the quality of solutions produced by the overall method and,
- the performance of the different phases in the heuristic method.

Heuristic Approach for Automated Shelf Space Allocation. Dario Landa-Silva, Fathima Marikar, Khoi Le. *Proceedings of the 24th ACM Symposium on Applied Computing (SAC 2009)*, Vol. 2, pp. 922-928, ACM Press, Hawaii, USA, March 2009. DOI: 10.1145/1529282.1529482. Paper available here.

# Shelf Space Allocation (SSA)

## Importance of Good Shelf Space Allocation

Maximise Space Utilisation

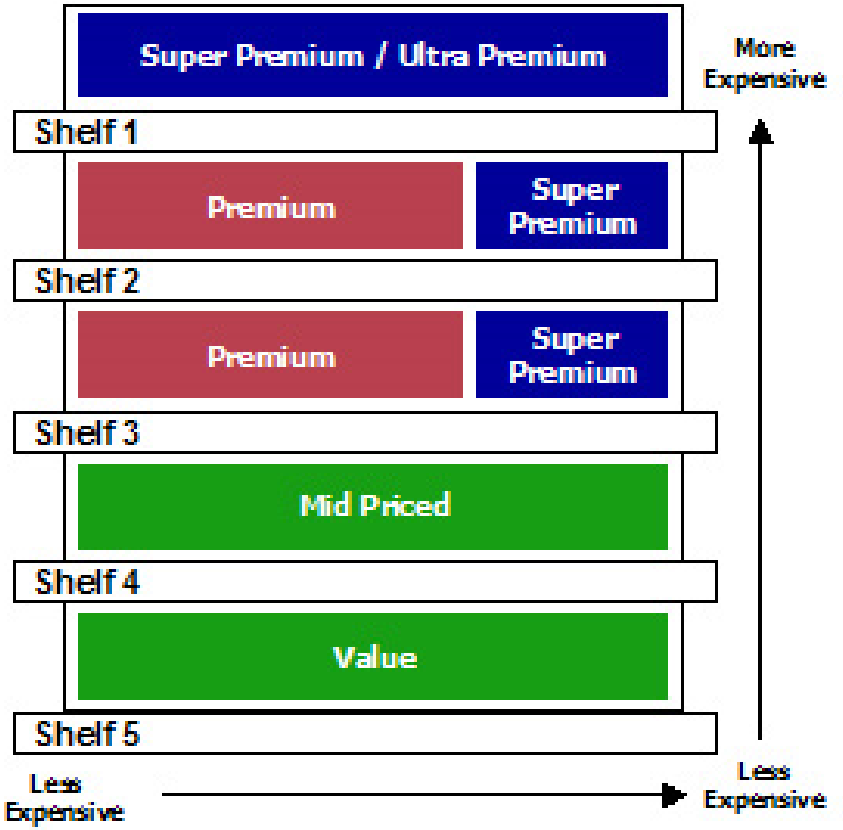
Increase Profit

Increase Sales and/or Reduce Costs

Improve Stock Control

Improve Customer Satisfaction

Through the Use of *Planograms*



# SSA Problem Specification

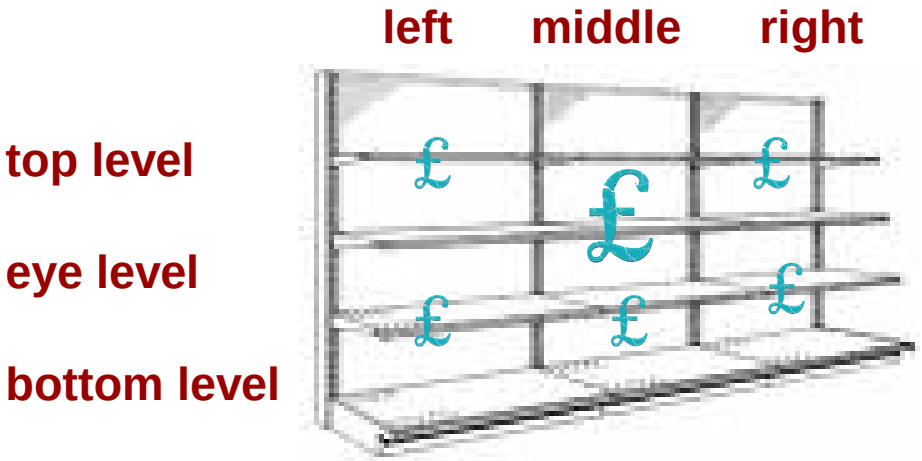
A **block** is a group of shelves

A **shelf** is a horizontal unit of space and can be: top level, eye-level or bottom level

A shelf is (virtually) **divided into 3 parts**: left, middle and right

**Priorities** are assigned to shelves and parts by the manager

Eye-level shelves and centre parts assigned the **highest profitability**



# Input Data

For shelves and parts:

- Number
- Length
- Priority



For products:

- Length of a facing
- Maximum facings required
- Minimum facings required
- Number of available units
- Profit per unit
- Profitability according to location


Additional considerations:

- Product selection stage is not required
- Height and depth of a product unit are ignored
- Generally, all units of the same product are allocated in contiguous space
- No elaborate product categorisation is used

# Problem Formulation

$$\text{Maximise} \quad Z = \sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{p} \varphi_{ijk} x_{ijk} \qquad (1)$$

$$\text{subject to} \quad \sum_{i=1}^{n} a_i x_{ijk} \le T_{jk} \quad \text{for} \quad j = 1\text{K}\ m, k = 1\text{K}\ p \qquad (2)$$

$$\sum_{j=1}^{m}\sum_{k=1}^{p} y_{ijk} = 1 \quad \text{for} \quad i = 1\text{K}\ n \qquad (3)$$

$$L_i \le \sum_{j=1}^{m}\sum_{k=1}^{p} x_{ijk} \le U_i \quad \text{for} \quad i = 1\text{K}\ n \qquad (4)$$

$$y_{ijk} \le x_{ijk} \le U_i y_{ijk} \quad \text{and} \quad x_{ijk} \in Z^{+} \qquad (5)$$

$n$ : products to allocate

$m$ : shelves availabe

$p$ : shelf parts ($p = 3$ here)

$a_i$ : length of facing

$L_i$ : minimum facings

$U_i$ : maximum facings

$x_{ijk}$ : facings $i$ in part $j$ of shelf $k$

$y_{ijk}$ : facings $i$ allocated to part $j$ of shelf $k$?

$\rho_i$ : profit per unit

$\sigma_j$ : priority of shelf $j$

$\omega_k$ : priority of shelf part $k$

$\varphi_{ijk} = \rho_i \sigma_j \omega_k$ : estimated profitability

# Tailored Heuristic Method

## 1. Preparatory Phase

Is there enough space to allocate all products?

$$\sum_{i=1}^{n} L_i a_i \leq \sum_{j=1}^{m} \sum_{k=1}^{p} T_{jk}$$

## 2. Allocation Phase

Produce an initial arrangement

Product profit
Product size
Random choice
Existing arrangement

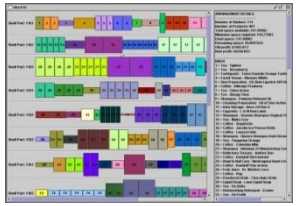
## 3. Adjustment Phase

Iterative changes to improve overall profit

Swap
AdjustIn
MultiShift
AdjustInter
RemoveLeastProfitable

## 4. Termination Phase

Compute quality of solution and display planogram

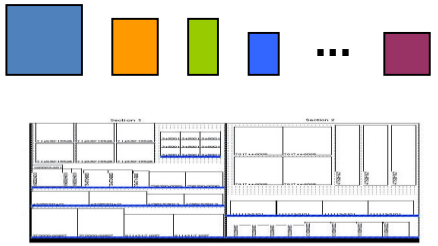


# Allocation Phase

## Prioritise

$P^S$ : sorted products, ↓ order $\rho_i/a_i$ or ↑ order $a_i$ or random

$S^S$ : sorted shelf parts, ↓ order of $\sigma_j$ and $\omega_k$

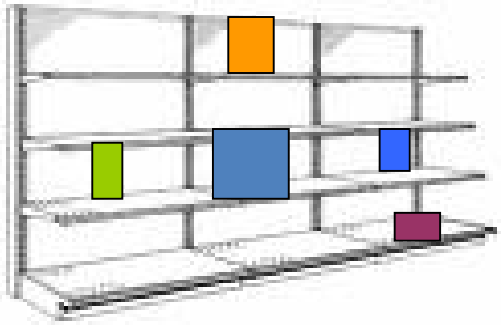


## Allocate Minimum

Given: $P^S$ and $S^S$ produce arrangement $\Lambda$

Take next product *i* and next shelf part *jk*

Allocate $L_i$ units to shelf part *jk*

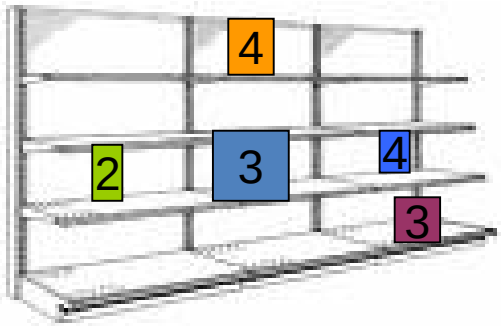


## Allocate More

Given: $P^S$ and arrangement $\Lambda$, improve $\Lambda$

Take next product *i*

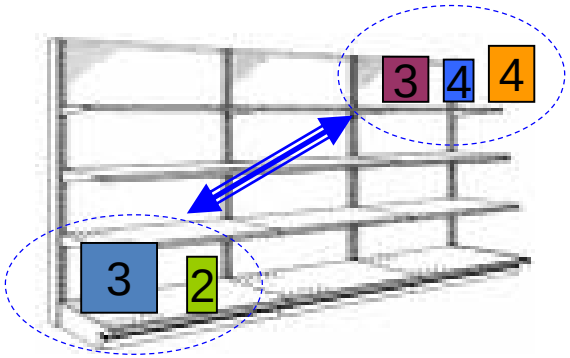
Increment $x_{ijk}$ by 1 ensuring feasibility ($U_i$ and $T_{jk}$)



# Adjustment Phase

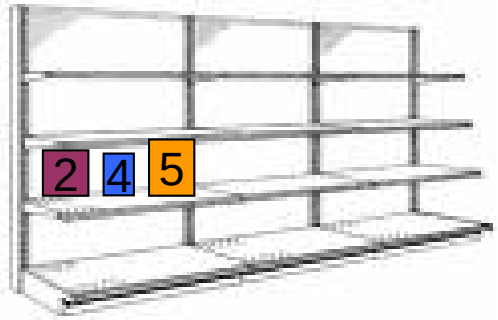
## Swap

All facings between products $i_1$ and $i_2$ located in different shelf parts $j_1k_1$ and $j_2k_2$



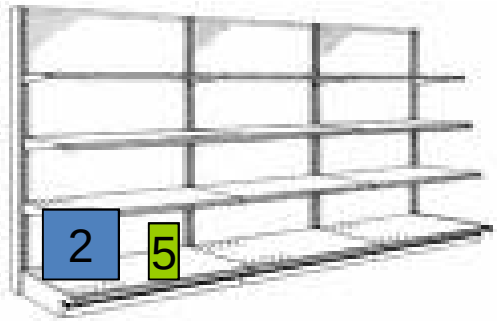

## AdjustIn

For products $i_1$ and $i_2$ both in shelf part $jk$ make $x_{i1jk} = x_{i1jk}+1$ and $x_{i2jk} = x_{i2jk} - 1$




## MultiShift

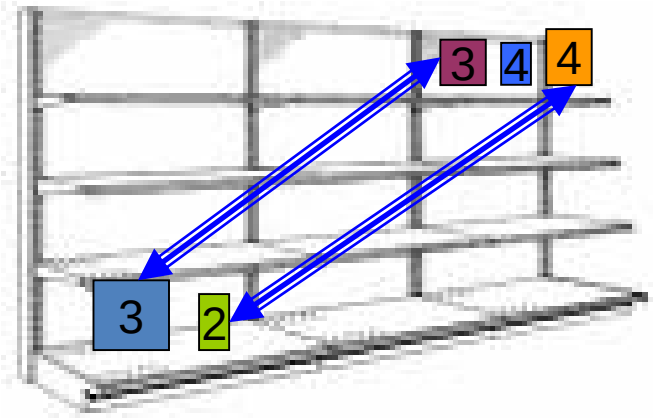
For products $i_1$ and $i_2$ both in shelf part $jk$ make $x_{i1jk} = x_{i1jk}+\alpha_1$ and $x_{i2jk} = x_{i2jk} - \alpha_2$



# Adjustment Phase

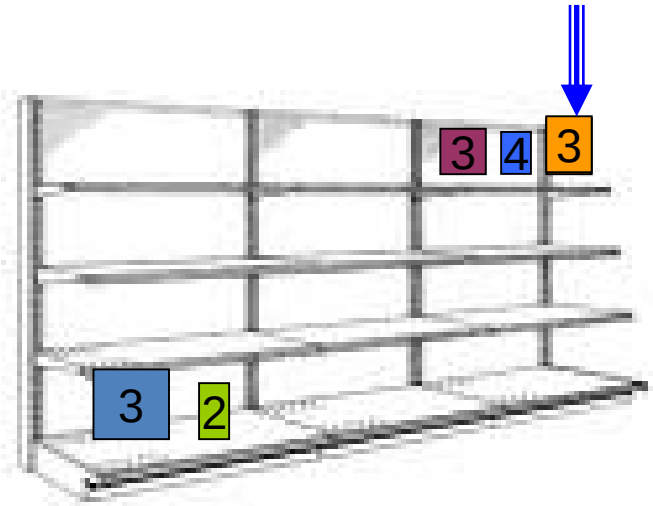
## AdjustInter

For products $i_1$ and $i_2$ both in shelf part $j_1k_1$ and products $i_3$ and $i_4$ both in shelf part $j_2k_2$
either:
- interchange $i_1$ on shelf $j_1k_1$ with $i_3$ on shelf $j_2k_2$
- interchange $i_2$ on shelf $j_1k_1$ with $i_4$ on shelf $j_2k_2$
- several checks made for the above interchanges




## RemoveLeastProfitable

Choose a random shelf part *jk* and with probability of 10%, find product *i* with $x_{ijk} > L_i$ contributing least profit $\varphi_{ijk}$ and make $x_{ijk} = x_{ijk} - 1$



# Experiments and Results

**Small Instance:**

$$n = 135, m = 5, p = 3, \text{average}(a_i) = 0.11 mts$$

$$\sum_{j=1}^{m}\sum_{k=1}^{p} T_{jk} = 33 mts, L_i = 1 \forall i, Ui \in \{2,3,\text{K},11\} \forall i$$

**Large Instance:**

$$n = 907, m = 38, p = 3, \text{average}(a_i) = 0.13 mts$$

$$\sum_{j=1}^{m}\sum_{k=1}^{p} T_{jk} = 248 mts, L_i = 1 \forall i, Ui \in \{2,3,\text{K},13\} \forall i$$

## Allocation Phase Results

| | Current | Profit | Size | Random |
|---|---|---|---|---|
| **Small** | 78.64 | 77.11 (0.07) | 71.36 (0.06) | 71.80 (0.07) |
| **Large** | 2463.05 | 2959.33 (3.55) | 2574.38 (3.33) | 2513.54 (4.07) |

# Experiments and Results

## Adjustment Phase Results

For each initial arrangement (current, profit, size, random)
Execute **Given Move** during 30 seconds x 30 times

**Small Instance**

| Current | Profit | Size | Random |
|---|---|---|---|
| AdjustInter | MultiShift | MultiShift | MultiShift |
| MultiShift | AdjustIn | AdjustInter | AdjustInter |
| AdjustIn | AdjustInter | AdjustIn | AdjustIn |
| Swap | Swap | Swap | Swap |

**Large Instance**

| Current | Profit | Size | Random |
|---|---|---|---|
| AdjustInter | AdjustIn | AdjustIn | MultiShift |
| MultiShift | MultiShift | MultiShift | AdjustIn |
| AdjustIn | AdjustInter | AdjustInter | AdjustInter |
| Swap | Swap | Swap | Swap |

Probabilities assigned for the local search:

MultiShift : 35%
AdjustInter : 35%
AdjustIn : 15%
Swap : 15%

# Experiments and Results

For each initial arrangement (current, profit, size, random)
Execute **Adjustment Phase** for *t* seconds x 15 times

**Small Instance**

| Profit Increase | | Overall Profit | |
|---|---|---|---|
| Mean | Std.Dev. | Initial | Final |
| 29.06 (37%) | 0.77 | 78.64 | 107.71 |
| 30.93 (40%) | 0.93 | 77.19 | 108.04 |
| 37.19 (52%) | 0.83 | 71.36 | 108.55 |
| 36.25 (50%) | 0.52 | 71.80 | 108.05 |

**Large Instance**

| Profit Increase | | Overall Profit | |
|---|---|---|---|
| Mean | Std.Dev. | Initial | Final |
| 2386.3 (97%) | 51.1 | 2463.0 | 4849.3 |
| 1869.7 (63%) | 27.5 | 2959.3 | 4829.1 |
| 2391.4 (93%) | 23.9 | 2574.3 | 4965.8 |
| 2191.7 (8%) | 36.0 | 2513.5 | 4705.2 |

# Example 2 – Heuristics for a UCTT Problem

In this work, various heuristic methods using constructive heuristics, local search and evolutionary computation mechanisms were developed to tackle benchmark instances of the University Course Timetabling (UCTT) Problem.

The purpose of the experimental evaluation was to assess the quality of solutions produced by the overall method but also to assess the performance of the different phases of the method.

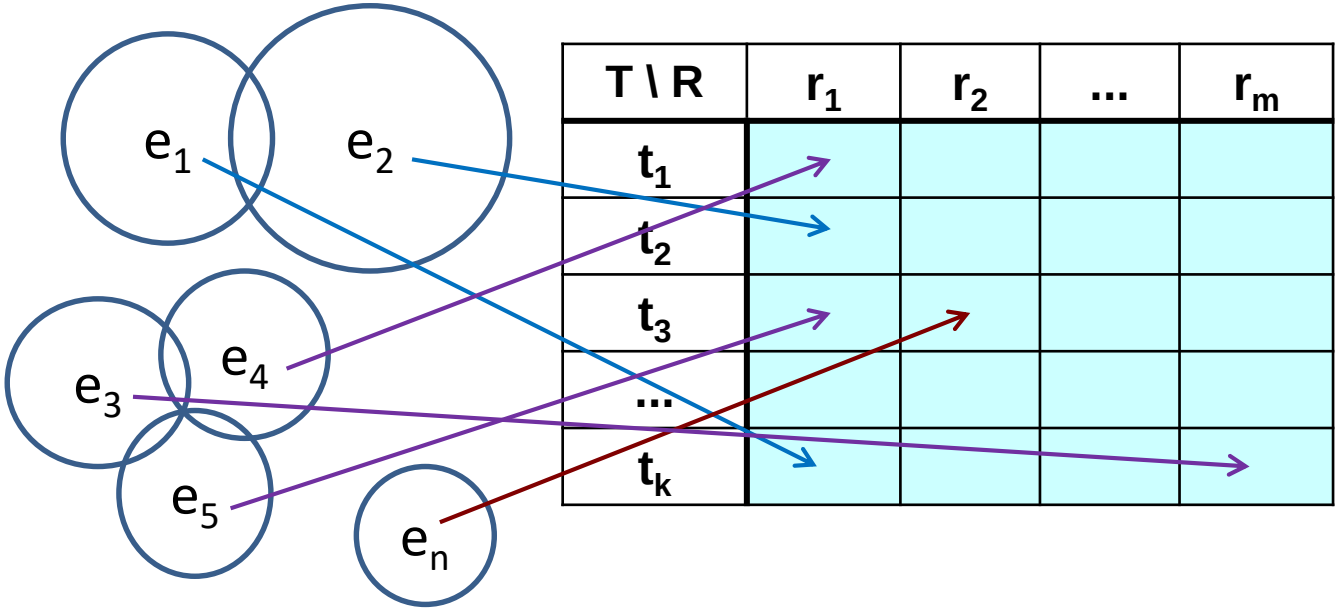
Computational Study of Non-Linear Great Deluge for University Course Timetabling. Joe Henry Obit, Dario Landa-Silva. *Intelligent Systems: From Theory to Practice*. Sgurev, Vassil and Hadjiski, Mincho and Kacprzyk, Janusz (eds), Studies in Computational Intelligence, Vol. 299, pp. 309-328, Springer-Verlag, 2010. DOI: 10.1007/978-3-642-13428-9_14

Evolutionary Non-Linear Great Deluge for University Course Timetabling. Dario Landa-Silva, Joe Henry Obit. *Hybrid Artificial Intelligent Systems*, E. Corchado, X. Wu, E. Oja, A. Herrero, B. Baruque (eds.), Lecture Notes in Computer Science, Vol. 5572, pp. 269-276, Springer, 2009. DOI: 10.1007/978-3-642-02319-4

Non-Linear Great Deluge with Learning Mechanism for Solving the Course Timetabling Problem. Joe H. Obit, Dario Landa-Silva, Djamilah Ouelhadj, Marc Sevaux. *8th Metaheuristics International Conference (MIC 2009)*, Hamburgh, Germany, July 2009.

Comparing Hybrid Constructive Heuristics for University Course Timetabling. Dario Landa-Silva, Joe Henry Obit. *VII ALIO/EURO Workshop on Applied Combinatorial Optimization*, Porto, Portugal, pp. 222-225, May 2011.

University Course Timetabling (UCTT) is the problem of assigning events to timeslots and rooms while satisfying a set of hard constraints and a set of additional requirements (soft constraints) associated to the quality of the timetable. There is a number of students enrolled in each event and events with students in common are considered conflicting events. The limited available resources (timeslots and rooms) makes the construction of a conflict-free timetable a difficult problem.



# Timetabling as Graph Colouring

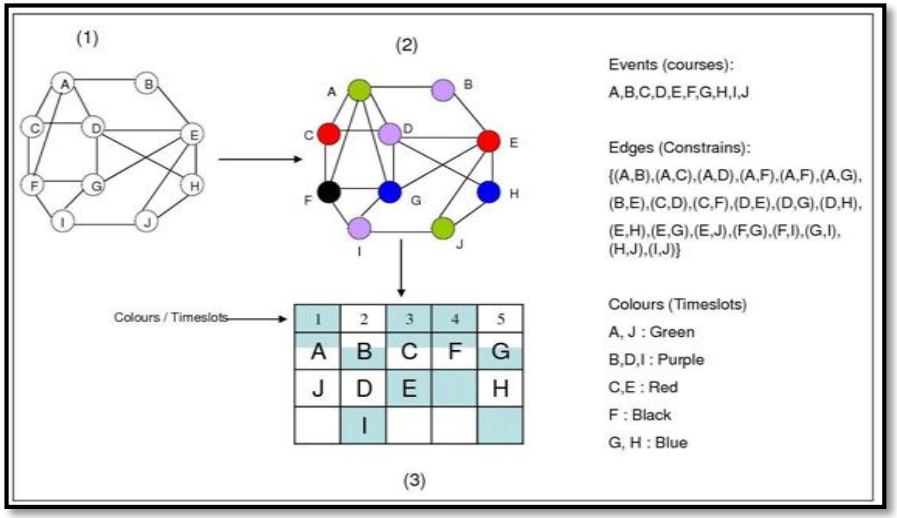
Conflict Matrix

| E \ E | $e_1$ | $e_2$ | $e_3$ | $e_4$ | $e_5$ | $e_n$ |
|---|---|---|---|---|---|---|
| $e_1$ | X | 1 | 0 | 0 | 0 | 0 |
| $e_2$ | 1 | X | 0 | 0 | 0 | 0 |
| $e_3$ | 0 | 0 | X | 1 | 1 | 0 |
| $e_4$ | 0 | 0 | 1 | X | 1 | 0 |
| $e_5$ | 0 | 0 | 1 | 1 | X | 0 |
| $e_n$ | 0 | 0 | 0 | 0 | 0 | X |

Some additional conditions:

- Room capacity
- Room features
- Spread of events
- Other requirements

Classroom assignment often tackled as sub-problem.




Graph Colouring Problem

- Vertices represent events
- Edges represent conflicts between events
- Assign colours to the nodes so that no two adjacent nodes have the same colour
- The chromatic number is the minimum number of colours needed which is related to finding cliques in a graph.

# UCTT Benchmark Instances

Problem described in (Socha et al. 2002) and the ITC-2002 web site. There are 45 slots (5 days, 9 hours), set of events, set of rooms of given size and features, set of students, students/events attendance matrix.

Four (Hard) Constraints

- A student cannot attend two events simultaneously.
- Only one event can be assigned per timeslot in each room.
- The room capacity must not be exceeded in any timeslot.
- The room features must be suitable for the assigned events.

Three Requirements (Soft Constraints)

- Students should not have exactly one event timetabled on a day.
- Students should not attend more than two consecutive events on a day.
- Students should not attend an event in the last timeslot of the day.

Socha, K., Knowles, J., Sampels, M.: A Max-min Ant System for the University Course Timetabling Problem. In: Dorigo, M., Di Caro, G.A., Sampels, M. (eds.) Ant Algorithms 2002. LNCS, vol. 2463, pp. 1–13. Springer, Heidelberg (2002).

ITC-2002: http://sferics.idsia.ch/Files/ttcomp2002/

# UCTT Benchmark Instances

Features of 11 problem instances by (Socha et al. 2002).

| Features | Small Size<br>5 Instances | Medium Size<br>5 Instances | Large Size<br>1 Instance |
|---|---|---|---|
| Number of events (*E*) | 100 | 400 | 400 |
| Number of rooms (*R*) | 5 | 10 | 10 |
| Number of features (*F*) | 5 | 5 | 10 |
| Number of students (*S*) | 80 | 200 | 400 |
| Number of timeslots (*T*) | 45 | 45 | 45 |
| Maximum events per student | 20 | 20 | 20 |
| Maximum students per event | 20 | 50 | 100 |
| Approximate features per room | 3 | 3 | 5 |
| Percent feature use | 70 | 80 | 90 |

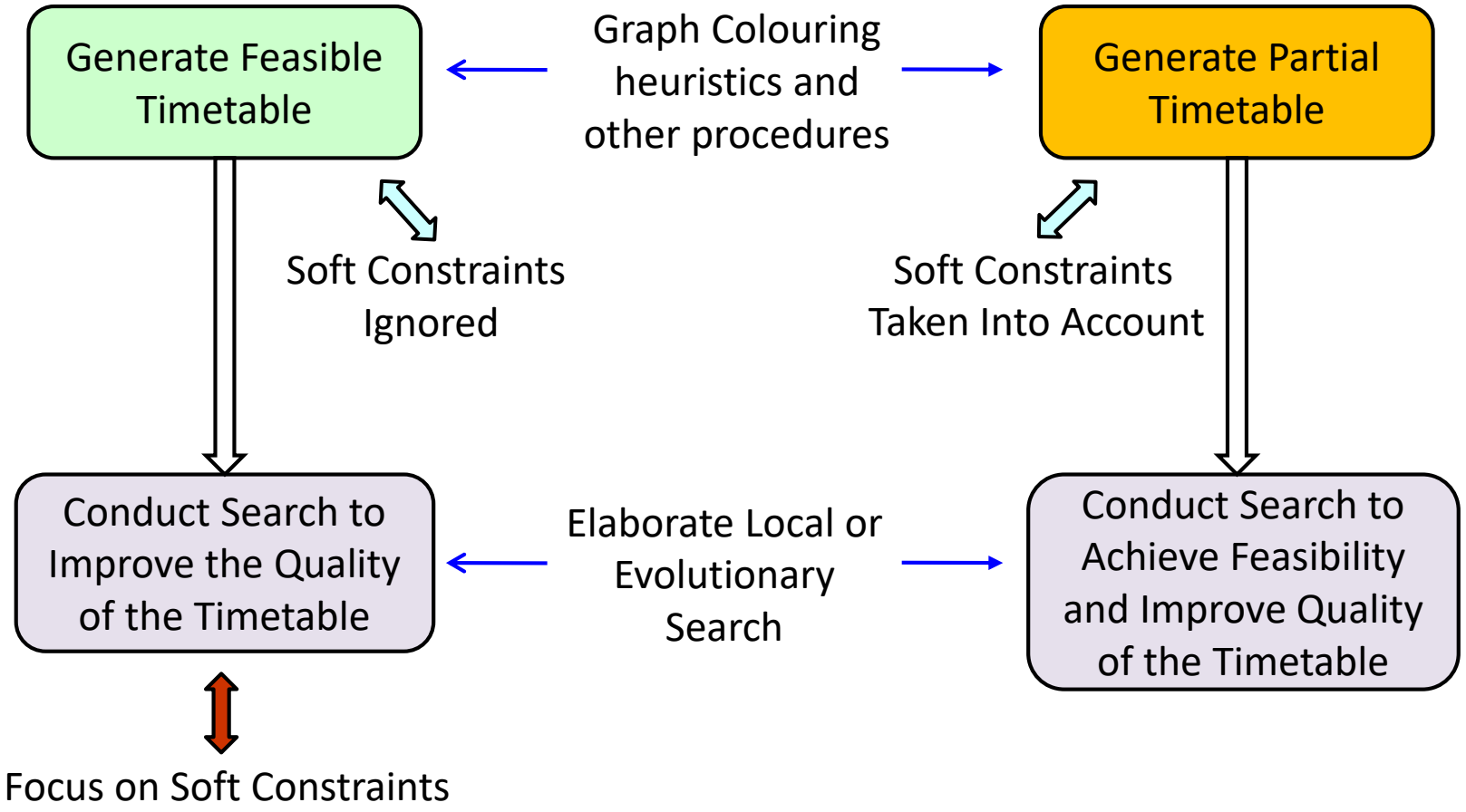
# UCTT Benchmark Instances

Features of the 20 problem instances for the ITC-2002.

| Instance | Events | Students | Rooms | Rooms/Event | Events/Student | Students/Event |
|---|---|---|---|---|---|---|
| com01 | 400 | 200 | 10 | 1.96 | 17.75 | 8.88 |
| com02 | 400 | 200 | 10 | 1.92 | 17.23 | 8.62 |
| com03 | 400 | 200 | 10 | 3.42 | 17.7 | 8.85 |
| com04 | 400 | 300 | 10 | 2.45 | 17.43 | 13.07 |
| com05 | 350 | 300 | 10 | 1.78 | 17.78 | 15.24 |
| com06 | 350 | 300 | 10 | 3.59 | 17.77 | 15.23 |
| com07 | 350 | 350 | 10 | 2.87 | 17.48 | 17.48 |
| com08 | 400 | 250 | 10 | 2.93 | 17.58 | 10.99 |
| com09 | 440 | 220 | 11 | 2.58 | 17.36 | 8.68 |
| com10 | 400 | 200 | 10 | 3.49 | 17.78 | 8.89 |
| com11 | 400 | 220 | 10 | 2.06 | 17.41 | 9.58 |
| com12 | 400 | 200 | 10 | 1.96 | 17.57 | 8.79 |
| com13 | 400 | 250 | 10 | 2.43 | 17.69 | 11.05 |
| com14 | 350 | 350 | 10 | 3.08 | 17.42 | 17.42 |
| com15 | 350 | 300 | 10 | 2.19 | 17.58 | 15.07 |
| com16 | 440 | 220 | 11 | 3.17 | 17.75 | 8.88 |
| com17 | 350 | 300 | 10 | 1.11 | 17.67 | 15.15 |
| com18 | 400 | 200 | 10 | 1.75 | 17.56 | 8.78 |
| com19 | 400 | 300 | 10 | 3.94 | 17.71 | 13.28 |

# Generating Good Quality Timetables

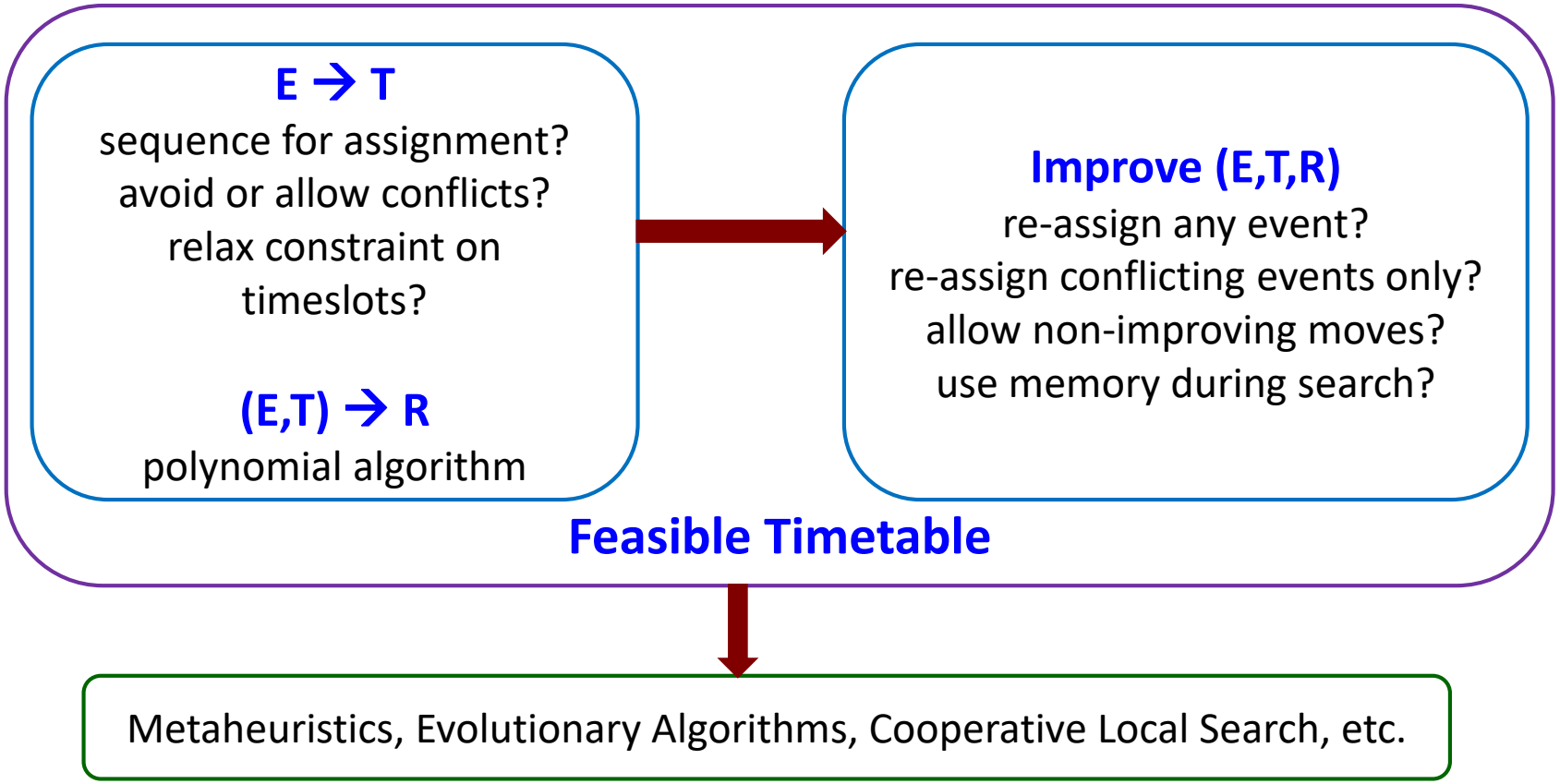
Many approaches in the literature do not start with a feasible timetable but instead seek feasibility while conducting the search for optimal solutions.



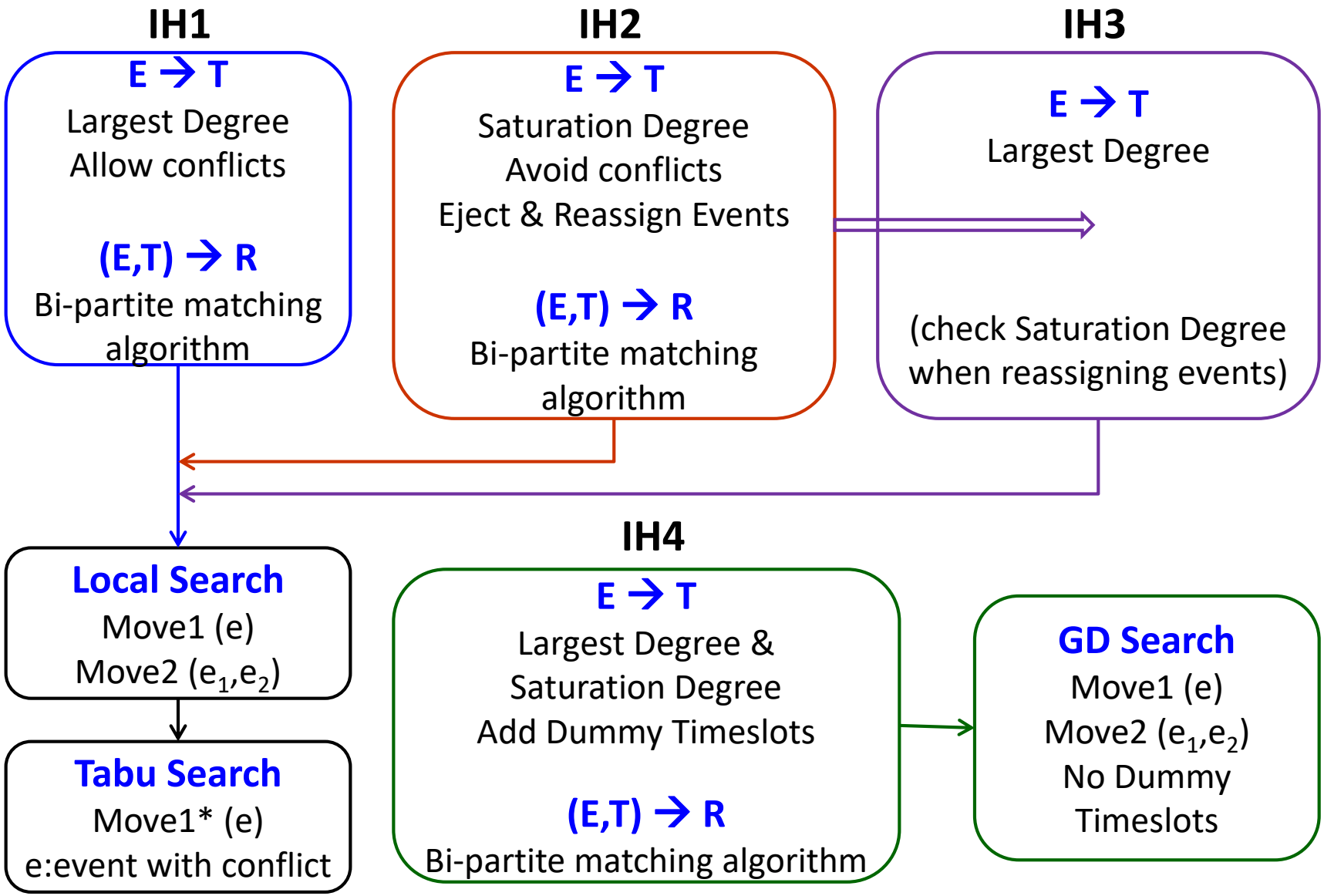
# Generating Feasible Timetables

## (Perhaps Poor Quality in Soft Constraints Satisfaction)

Solving the underlying graph colouring problem is very difficult.



# Several Tailored Heuristics



# Results From Heuristics on Socha et al. Instances

| Problem | IH1 | IH2 | IH3 | IH4 | Time (s) |
|---|---|---|---|---|---|
| S1 | **173** | 198 | 207 | 200 | 0.078 – 0.125 |
| S2 | 211 | 217 | **189** | 208 | 0.093 – 0.115 |
| S3 | **176** | 190 | 188 | 209 | 0.068 – 0.110 |
| S4 | 250 | **174** | 203 | 192 | 0.078 – 0.093 |
| S5 | 229 | 238 | 226 | **217** | 0.078 – 0.109 |
| M1 | 817 | **772** | 802 | 774 | 6.046 – 9.53 |
| M2 | 793 | **782** | 784 | 802 | 6.342 – 14.952 |
| M3 | **795** | 867 | 828 | 817 | 13.437 – 21.702 |
| M4 | **735** | 785 | 811 | 795 | 6.891 – 7.766 |
| M5 | 773 | 771 | 784 | **769** | 115.946 – 358.561 |
| L | **1340** | 1345 | 1686 | 1670 | 300 – 3000 |

Best results indicated in bold.

# Results From Heuristics on the ITC-2002 Instances

| Instance | IH1 | IH2 | IH3 | IH4 | Time (s) |
|---|---|---|---|---|---|
| com01 | 805 | **786** | 805 | 805 | 4.696 – 6.301 |
| com02 | **731** | 776 | **731** | 778 | 2.141 – 4.347 |
| com03 | **760** | 812 | **760** | 777 | 1.34 – 16.155 |
| com04 | 1201 | **1178** | 1201 | 1236 | 5.015 – 46.930 |
| com05 | 1246 | 1243 | 1246 | **1135** | 24.546 – 287.202 |
| com06 | 1206 | 1219 | 1206 | **1133** | 35.39 – 163.608 |
| com07 | 1391 | 1388 | 1391 | **1265** | 33.656 – 444.843 |
| com08 | 1001 | **968** | 1001 | 1006 | 1.810 – 13.749 |
| com09 | **841** | 859 | **841** | 843 | 1.464 – 11.086 |
| com10 | **786** | 816 | **786** | 799 | 34.678 – 85.61 |
| com11 | 852 | 877 | 852 | **839** | 121.968 – 536.796 |
| com12 | 814 | 831 | 814 | **788** | 102.515 – 426.609 |
| com13 | 1008 | 1010 | **1008** | 1009 | 2.26 – 6.976 |
| com14 | 1040 | **1032** | 1040 | 1355 | 3.716 – 51.952 |
| com15 | 1165 | 1162 | 1165 | **1161** | 135.14 – 227.89 |
| com16 | **887** | 911 | 887 | 888 | 2.592 – 6.415 |
| com17 | 1227 | **1032** | 1227 | 1199 | 1.136 – 14.952 |
| com18 | 793 | **724** | 793 | 763 | 1.892 – 4.249 |
| com19 | **1184** | 1212 | **1184** | 1209 | 3.928 – 20.753 |
| com20 | **1137** | 1161 | **1137** | 1205 | 1.072 – 1.804 |

Best results indicated in bold.

# Direct Solution Representation

Events are assigned to pairs <timeslot,room>

| t1 | | t2 | | … | T | |
|---|---|---|---|---|---|---|
| r1 | e04 | r1 | e14 | … | r1 | e34 |
| r2 | e21 | r2 | e52 | … | r2 | e18 |
| r3 | e09 | r3 | e23 | … | r3 | e56 |
| … | … | … | … | … | … | … |
| R | e12 | R | e32 | … | R | e12 |

# Neighbourhood Moves for UCTT

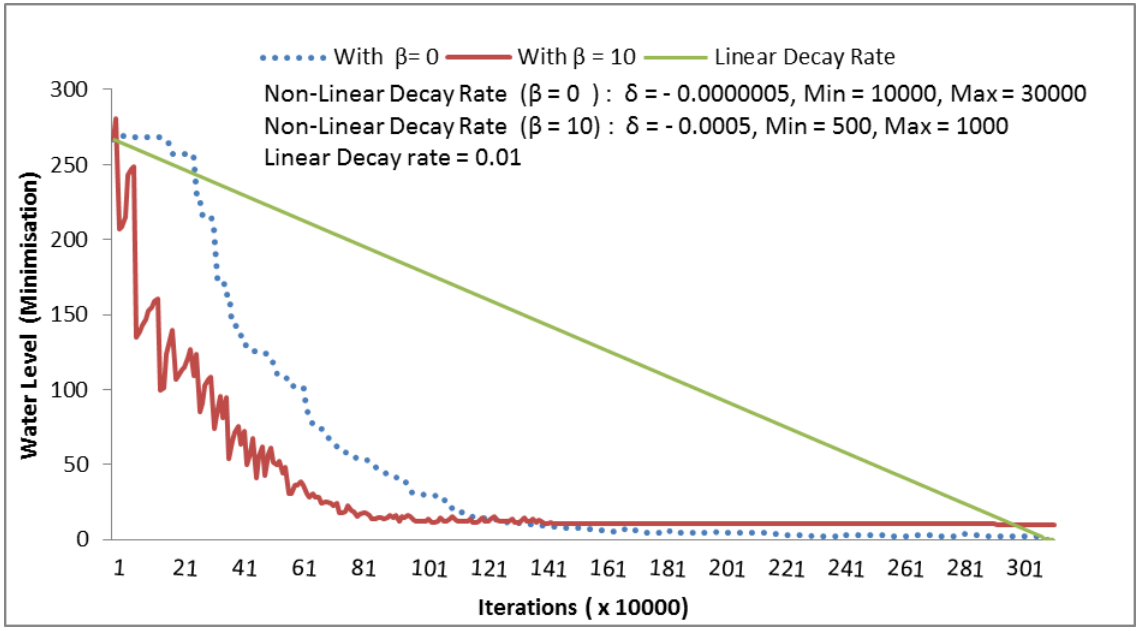
Moves to seek the improvement of solutions.

- Move N1: selects one event at random and assigns it to a feasible pair timeslot-room also chosen at random ensuring feasibility.
- Move N2: selects two events at random and swaps their timeslots and rooms ensuring feasibility.
- Move N3: identifies an event that violates soft constraints and then moves it to another pair timeslot-room selected at random ensuring feasibility.

# Great Deluge With Non-linear Decay Rate

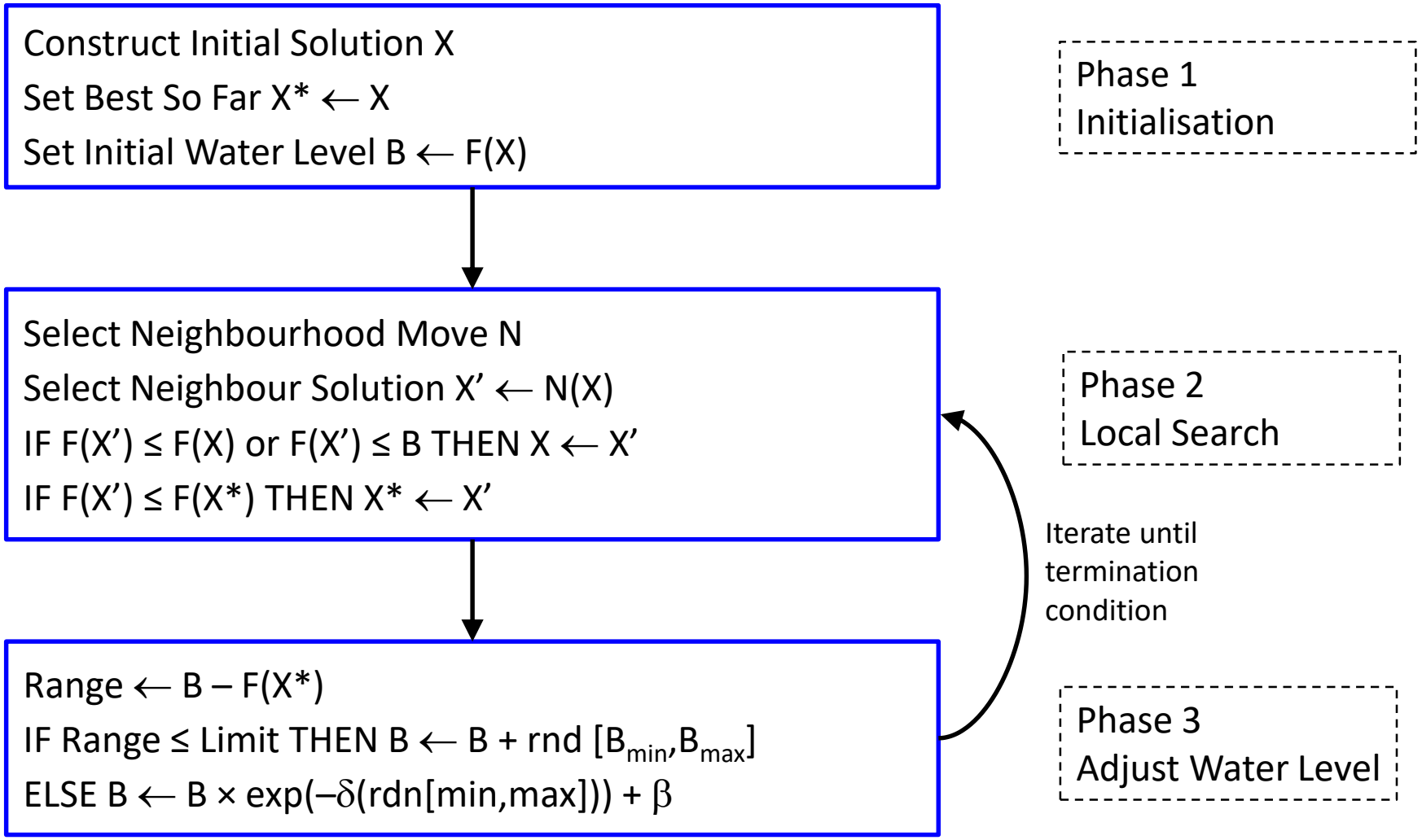
- Set initial water level B = F(current).
- A non-linear decay rate is proposed: $B = B \cdot \exp^{-\delta(rnd[\min,\max])} + \beta$

- $\beta$ is the best expected value in the optimal solution, i.e. lower bound.
- $\delta$, min and max control the speed of the decay rate for the water level.
- $\delta$, min, should be tuned accordingly.
- A candidate solution is accepted if F(candidate) ≤ F(current) or F(candidate) ≤ B.

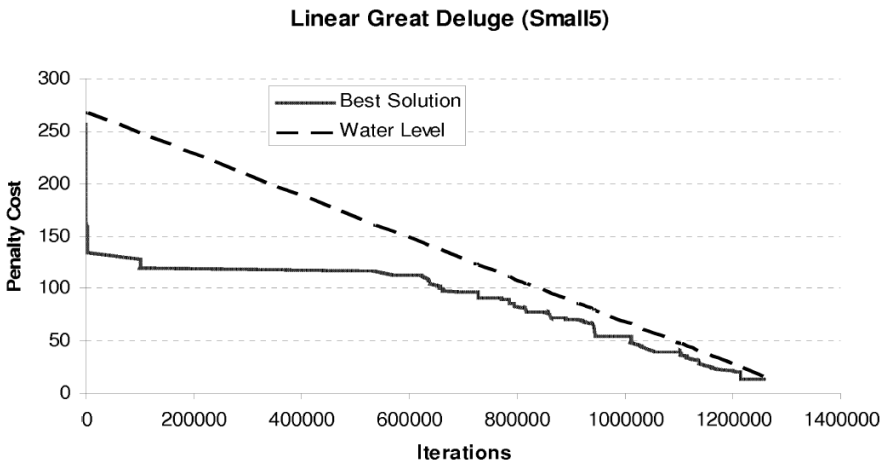


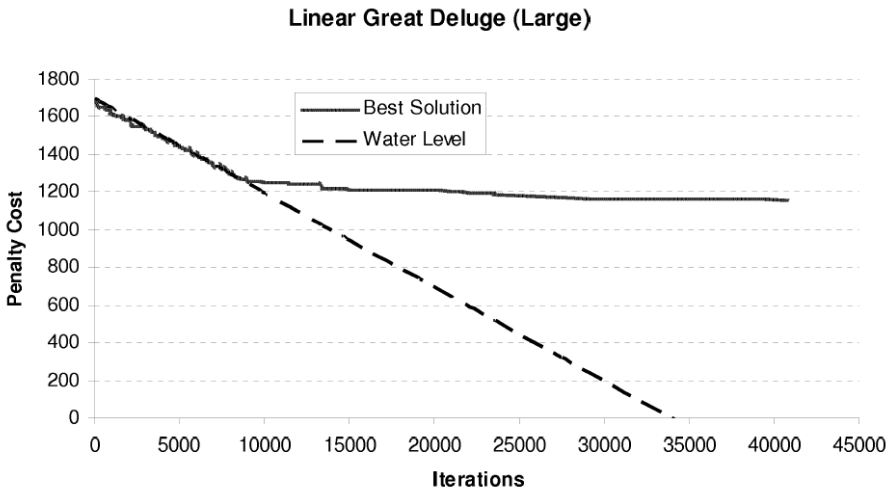

- The water level B rises again when its value and F(candidate) are about to converge, it increases by a random number within $[B_{min}, B_{max}]$, with $B_{min} \approx 2$ and $B_{max} \approx 5$, should be tuned accordingly.

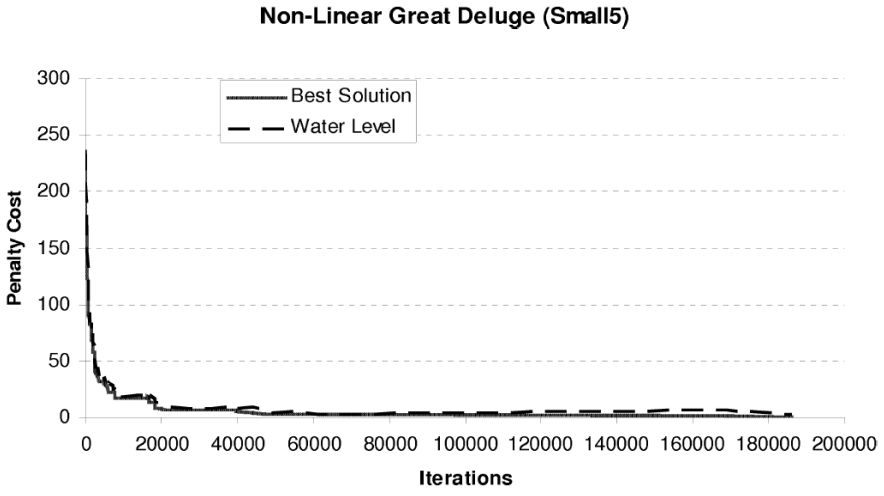
# Great Deluge With Non-linear Decay Rate

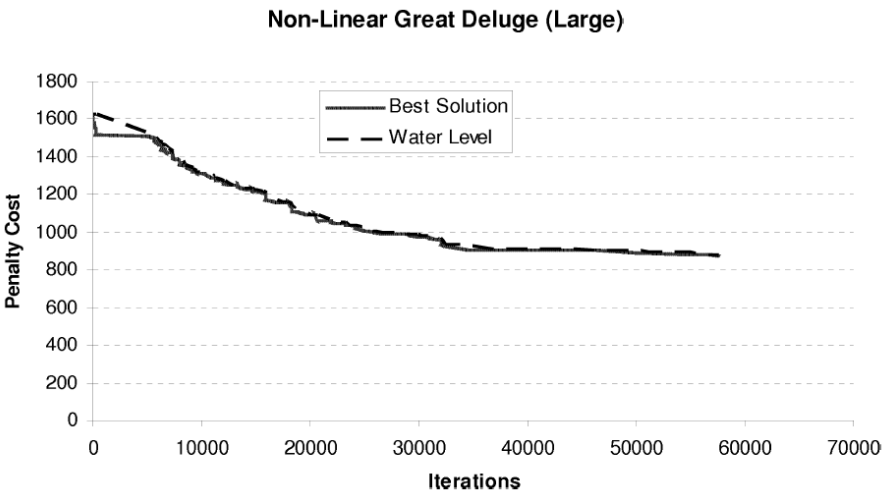


# Comparing Linear vs Non-linear Great Deluge



# Comparing Linear vs Non-linear Great Deluge

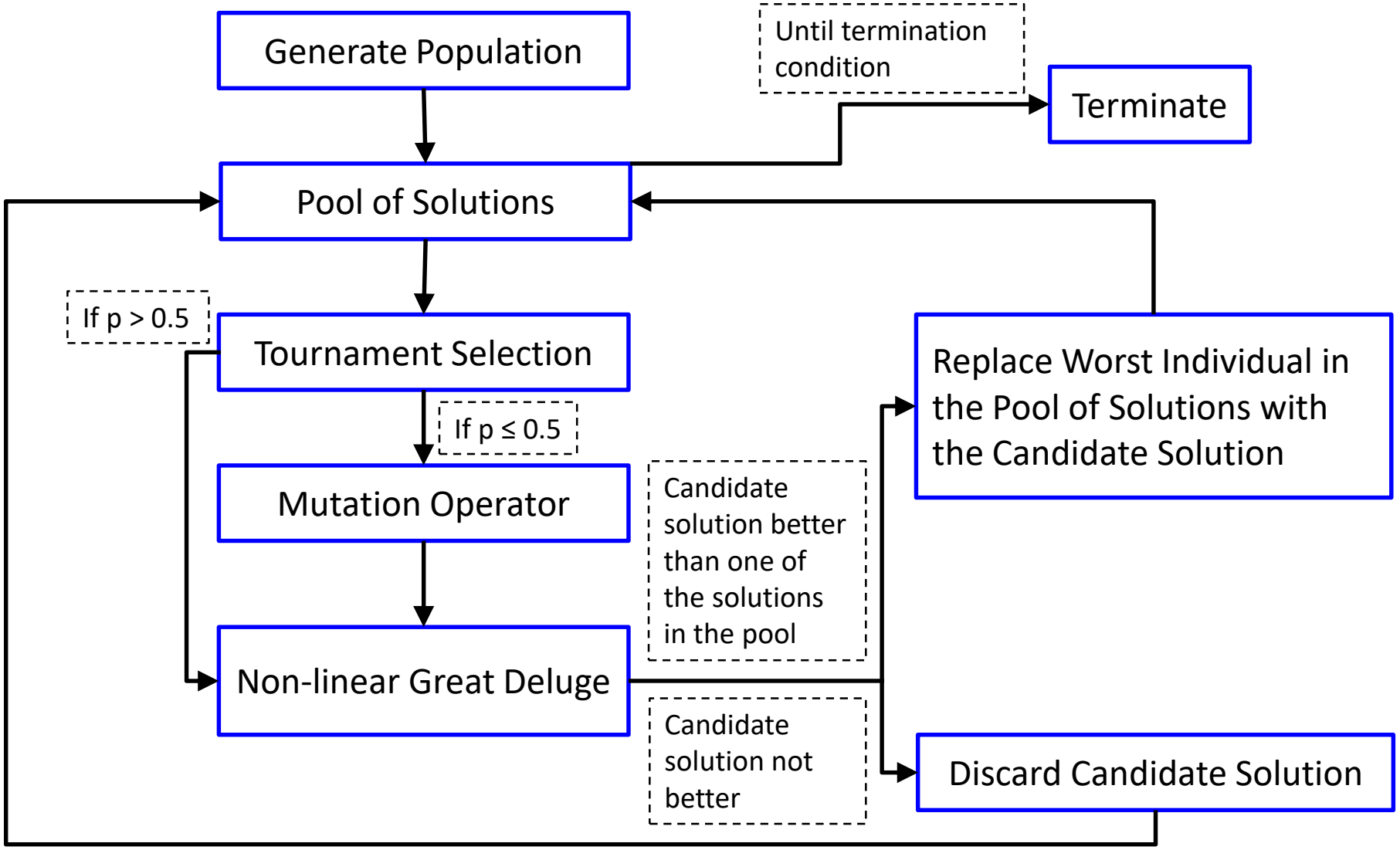
## Linear Great Deluge

- The decay rate is pre-determined and fixed
- Mainly, the search is driven by the water level
- When the best solution and water level converge the algorithm accepts only improving solutions

## Non-Linear Great Deluge

- The decay rate changes at every iteration
- Mainly, the water level is driven by the search
- This algorithm never becomes an only improving search

# Evolutionary Non-linear Great Deluge

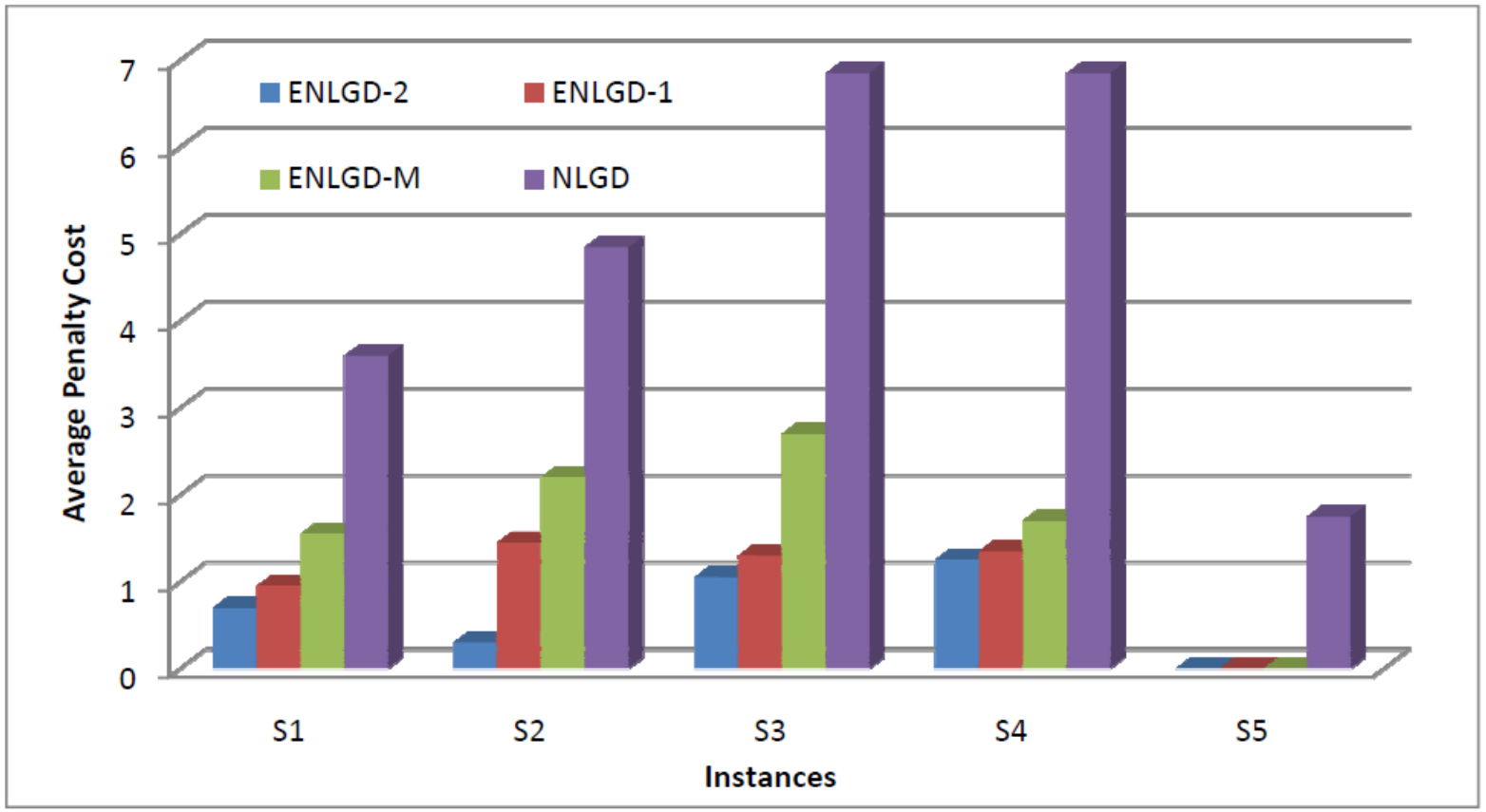


# Mutation Operator for ENGD

Selects one of these three moves at random:

- M1: selects one event at random and assigns it to a feasible pair timeslot-room also chosen at random.
- M2: selects two events at random and swaps their timeslots and rooms while ensuring feasibility.
- M3: select three events at random and exchanges the position of the events at random and ensuring feasibility.

# Comparing Evolutionary NGD

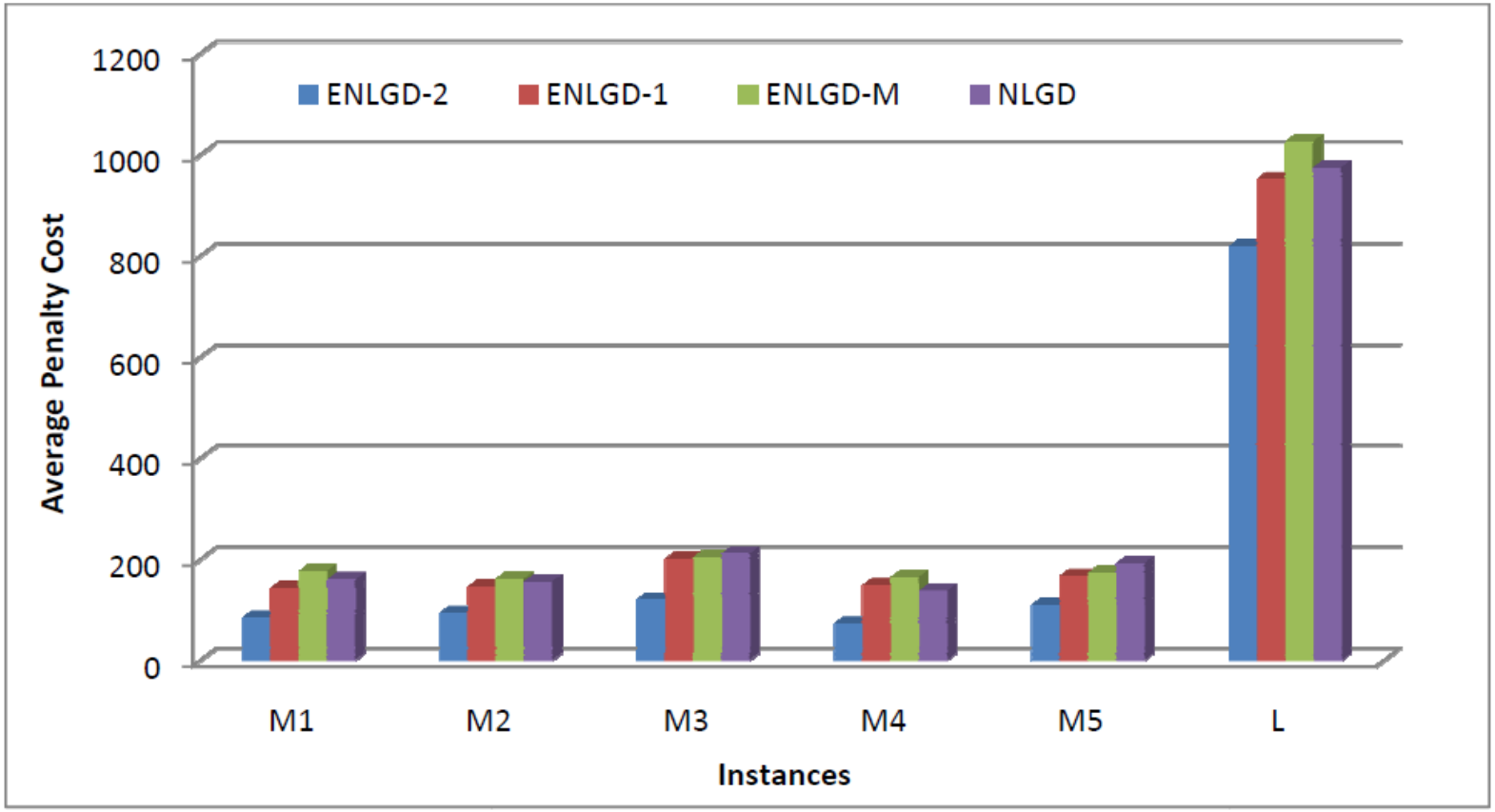



ENGD-1 runs the local search (NGD) for a fixed time or number of idle iterations.

ENGD-2 runs the local search (NGD) for 3 seconds each time.

ENGD-M is like ENGD-1 without the mutation.

# Comparing Evolutionary NGD




ENGD-1 runs the local search (NGD) for a fixed time or number of idle iterations.

ENGD-2 runts the local search (NGD) for 3 seconds each time.

ENGD-M is like ENGD-1 without the mutation.

# Non-linear Great Deluge With Reinforcement Learning

A reinforcement learning mechanism is used to guide the selection of the moves.

Initially, all moves have the same probability to be selected.

The learning mechanism tunes the probabilities of selecting moves as the search progresses so that the algorithm tries to learn which moves to use.

Each move is assigned a weight $W_i$. At every learning period, the weight $W_i$ of the move is updated.

Two types of learning mechanisms:

- Learning with static memory length
- Learning with dynamic memory length

# Learning With Static Memory Length

In each iteration, moves are selected with probability: $p_i = W_i / \sum_{i=1}^{n} W_i$

Initially, every weight is set to $W_i = 0.01$ for each of the n moves.

If the chosen move improves the current solution, a reward of 1 point is given to the move, i.e. $W_i = W_i + 1$, otherwise no rewards is given.

The weights are updated at every learning period (lp) given by lp = max(K/500,n), where K is the total number of feasible moves explored.

The weights $W_i$ are normalised by the ratio new(i)/total(i) when range ≤ 1 and by the ratio accept(i)/total(i) when range > 1.

- total(i) is the number of times that move i is called
- new(i) is the number of times that move i generates solution with different objective function value
- accept(i) is the number of times that solution generated wit move i meets the NGD acceptance criterion

# Learning With Dynamic Memory Length

In each iteration, moves are selected with probability: $p_i = W_i + W_{\min} / \sum_{i=1}^{n} (W_i + W_{\min})$

Initially, every weight is set to $W_i = 0.01$ for each of the n moves and $W_{\min} = \min(0, W_i)$.

When the selected move improves the current solution, the move is rewarded, otherwise the move is punished:

$$\Re_{ij} \begin{cases} r & \text{if } \Delta < 0 \\ -r & \text{if } \Delta > 0 \\ \Im & \text{if } \Delta = 0 \text{ and new solution} \\ \Im & \text{if } \Delta = 0 \text{ and no new solution} \\ 0 & \text{if not selected} \end{cases}$$

where r = 1, $\Im$ = 0.1 and Δ is the difference between the best solution so far and the current solution.

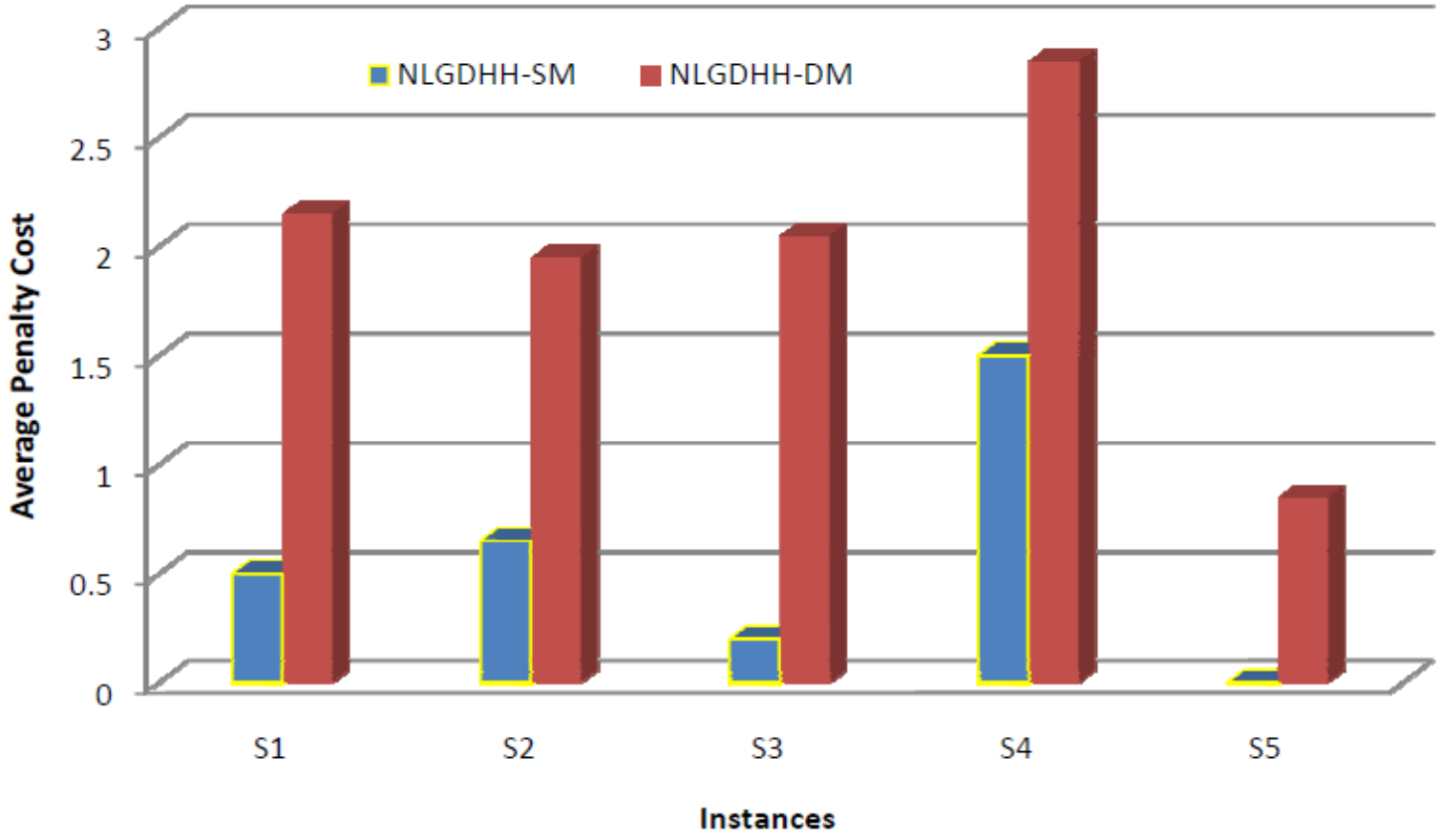
# Learning With Dynamic Memory Length

Weights are updated every time the algorithm performs a feasible move. $W_i = \sum_{j=k}^{h} \sigma^i \Re_{ij}$

Parameter σ is the length of the dynamic memory.

In every learning period lp, the algorithm updates σ with a random value in (0.5, 1.0]. Like before, lp = max(K/500,n).

# Comparing Static and Dynamic Memory Length



# Comparing Static and Dynamic Memory Length