Move Acceptance in Local Search Metaheuristics and Parameter Setting Issues

Ender Özcan

Lecture 4

Computational Optimisation & Learning Lab

The University of Nottingham

UNITED KINGDOM · CHINA · MALAYSIA

Scheduling Notation – Examples

- 1 | prec | $C_{\max}$

  A single machine, general precedence constraints, minimising makespan (maximum completion time).

- $P3 \mid d_j, s_{jk} \mid \sum L_j$

  3 identical machines, each job has a due date and sequence dependent setup times between jobs, minimising total lateness of jobs.

- $R \| \sum C_j$

  variable number of unrelated parallel machines, no constraints, minimising total completion time.

Lecture 3
A Single Machine Scheduling Problem

$$1 \mid d_j \mid \sum w_j T_j$$

- Given *n* jobs to be processed by a single machine, each job (*j*) with a *due date* (d_j) (i.e. hard deadline), processing time (p_j), and a weight (w_j) (i.e., job with the highest weight, say more important and so needs to finish on time), **find the optimal sequencing of jobs** producing the minimal weighted *tardiness* (T_j).
- Tardiness is 0 if a job completes on time ($C_j \leq d_j$), otherwise it is the time spent after the due date to completion ($C_j - d_j$)

$T_j = \max(C_j - d_j, 0)$
tardiness of job *j*

Lecture 3

Example: Computing Weighted Tardiness

$1 \mid d_j \mid \sum w_j T_j$

jobs	1	2	3	4
p_j	10	10	13	4
d_j	4	2	1	12
w_j	14	12	1	12

- What would be weighted tardiness of the solution which uses the shortest processing time for sequencing the jobs? (E.g., <4, 1, 2, 3>)
- t: 0 4 14 24 37 **makespan**

p_4 C_4 d_4=12 p_1 C_1 d_1=4 p_2 C_2 d_2=2 p_3 C_3 d_3=1

- $\sum w_j T_j = w_4 \max(C_4 - d_4, 0) + w_1 \max(C_1 - d_1, 0) + w_2 \max(C_2 - d_2, 0) + w_3 \max(C_3 - d_3, 0)$

$= 12\max(-8,0)+14\max(10,0)+12\max(22,0)+1\max(36,0)$

$= 0 + 140 + 264 + 36 = 440$

Lecture 3 – Overall Summary

- Each metaheuristic has a mechanism for escaping from local optima
  - Balance between exploration and exploitation is important
  - Iterated Local Search enforces exploration and exploitation explicitly
  - Tabu Search uses flexible memory structures in conjunction with strategic restrictions and aspiration levels
- There are different types of metaheuristics embedding different components
  - The performance of a metaheuristic often varies based on the chosen design components – hence the need for empirical studies
- Scheduling contains many crucial NP hard real-world optimisation problems (α | β | γ) often dealt with in manufacturing/ production using heuristics/hyper-/metaheuristics.

1. Local Search Metaheuristics and Move Acceptance Methods

COM2001/2011: Artificial Intelligence Methods

Ender Özcan

Computational Optimisation & Learning Lab

The University of Nottingham

UNITED KINGDOM · CHINA · MALAYSIA

Lecture 4

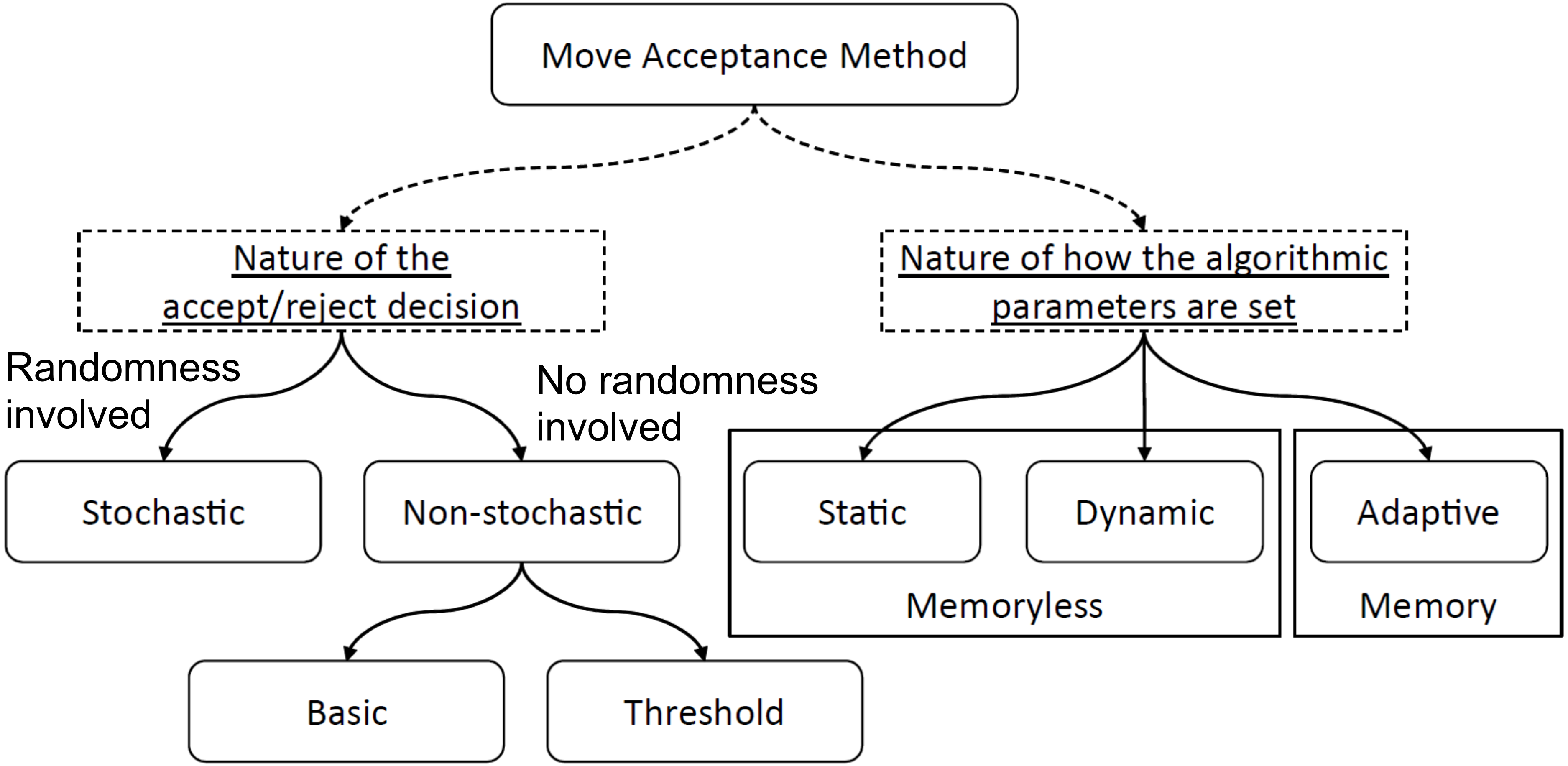
Stochastic Local Search –
Single Point Based Iterative Search
(Local Search Metaheuristics) - revisited

Algorithm 1: Outline of a Local Search Metaheuristic.

```
s ← generateInitialSolution();
s_best ← s;
while termination criteria not met do
    s' ← apply(h, s);
    process_1();
    s ← acceptRejectDecision(s, s');
    if f(s').isBetterThan(f(s_best)) then
        s_best ← s';
    end
    process_2();
end
return s_best;
```

- Move Acceptance decides whether to accept or reject the new solution considering its evaluation/quality, $f(s')$
- Accepting non-improving moves could be used as a mechanism to escape from local optimum

Move Acceptance Methods – A Taxonomy



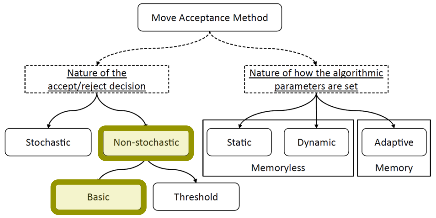
W. G. Jackson, E. Özcan and R. I. John, Move Acceptance in Local Search Metaheuristics for Cross-domain Search, Expert Systems with Applications, 109: 131-151, 2018 [original PDF]. [PDF]

Parameter Setting Mechanisms in Move Acceptance

- **Static**, either there is no parameter to set or parameters are set to a fixed value. E.g., IoM=5;
- **Dynamic**, parameter values vary with respect to time/iteration count. Given the same candidate and current solutions at the same current elapsed time or iteration count, the acceptance threshold or acceptance probability would be the same irrespective of search history.

  E.g., IoM = round(1+ ($iter_{current}$ / $iter_{max}$) * 4);
- **Adaptive**, Given the same candidate and current solutions at the same current elapsed time or iteration count, the acceptance threshold or acceptance probability is not guaranteed to be the same as one or more components depend on search history. E.g., if for 100 steps best solution found in the stage cannot be improved, then IoM++, and after any improvement, reset IoM=1;

Non-stochastic Basic Move Acceptance Methods



- Reuse the objective values of previously encountered solutions for the accept/reject decisions
- **Static**
  - All Moves: returns true regardless of $f(s')$
  - Improving Moves Only: $f(s') < f(s)$
  - Improving and Equal: $f(s') \leq f(s)$
- **Dynamic**: None
- **Adaptive**
  - Late Acceptance: compares the quality of the solution with that of the solution accepted/visited L iterations previously s_{t-L}, and accepts the move if and only if $f(s') \leq f(s_{t-L})$

Late Acceptance Algorithm – Pseudocode

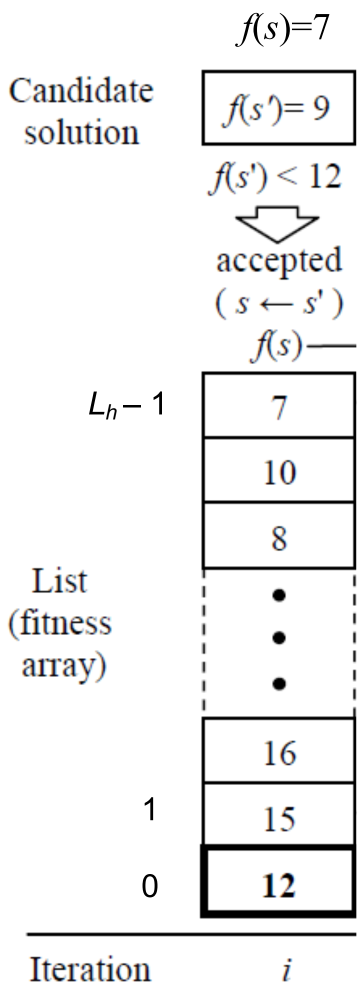
```
Produce an initial solution s
Calculate initial cost function C(s)   f(s)
Specify L_h
For all k∈{0...L_h-1} f_k:=C(s)
First iteration I:=0; I_idle:=0
Do until (I>100000) and (I_idle>I*0.02)
   Construct a candidate solution s*
   Calculate a candidate cost function C(s*)
   If C(s*)≥C(s)
      Then increment the idle iterations number I_idle:=I_idle+1
      Else reset the idle iterations number I_idle:=0
   Calculate the virtual beginning v:=I mod L_h
   If C(s*)<f_v or C(s*)≤C(s)
      Then accept the candidate s:=s*
      Else reject the candidate s:=s
   If C(s)<f_v
      Then update the fitness array f_v:=C(s)
   Increment the iteration number I:=I+1
```

- **Initilisation**: Assign all elements of the list to be equal to the initial cost (objective value)
- List for the history of the objective values of the recent solutions is implemented as a circular queue

Edmund K. Burke, Yuri Bykov, [The late acceptance Hill-Climbing heuristic](). EJOR 258(1): 70-78 (2017)

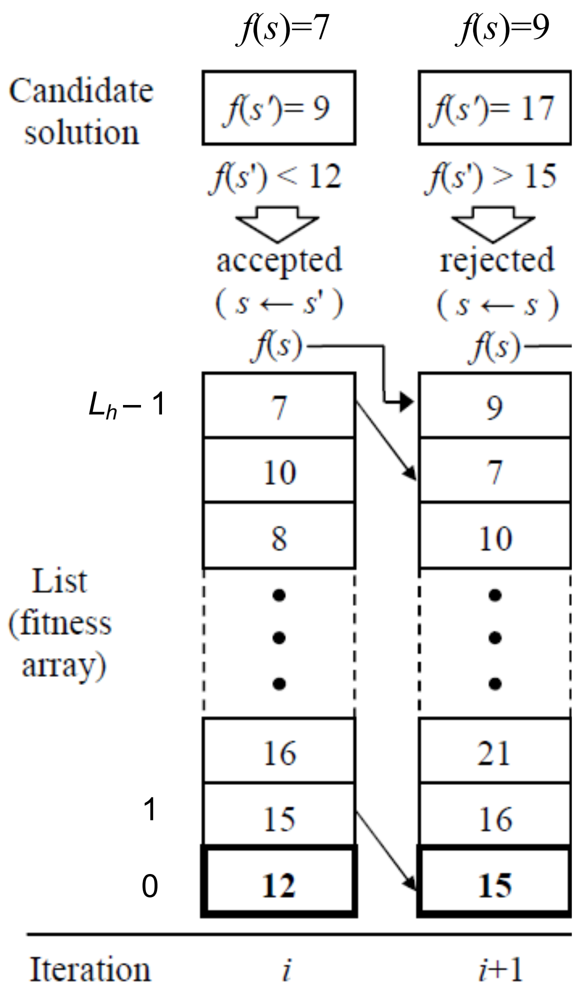
A Sample Run –
Late Acceptance Algorithm

linear queue implementation



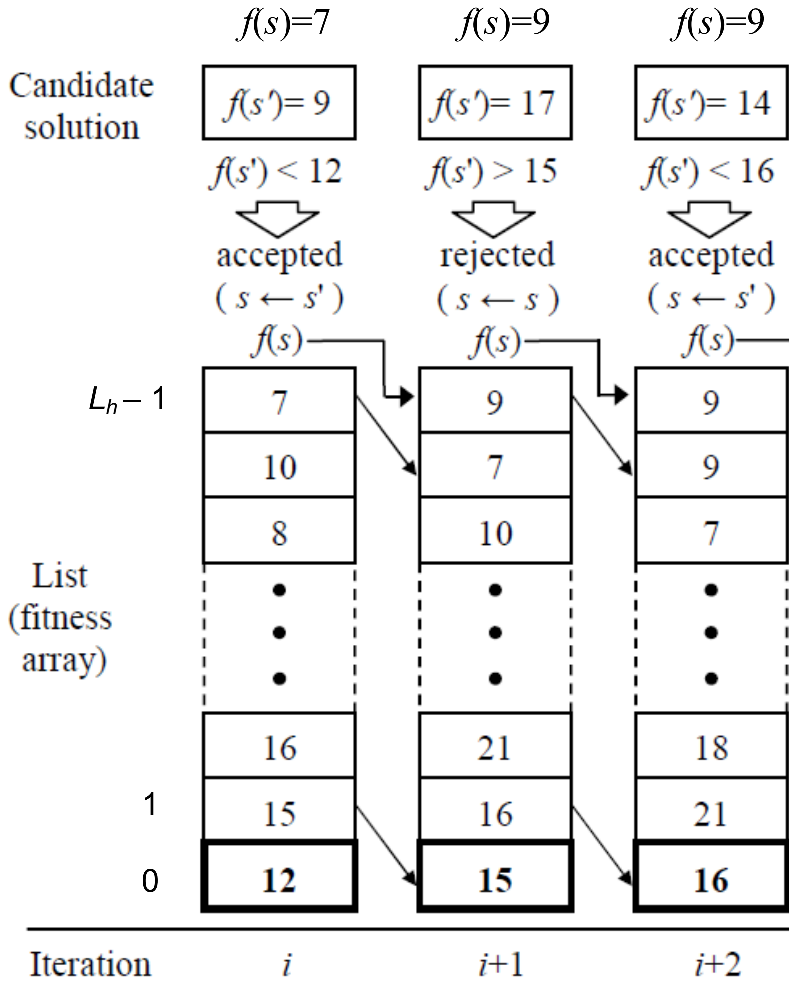
A Sample Run –
Late Acceptance Algorithm

linear queue implementation



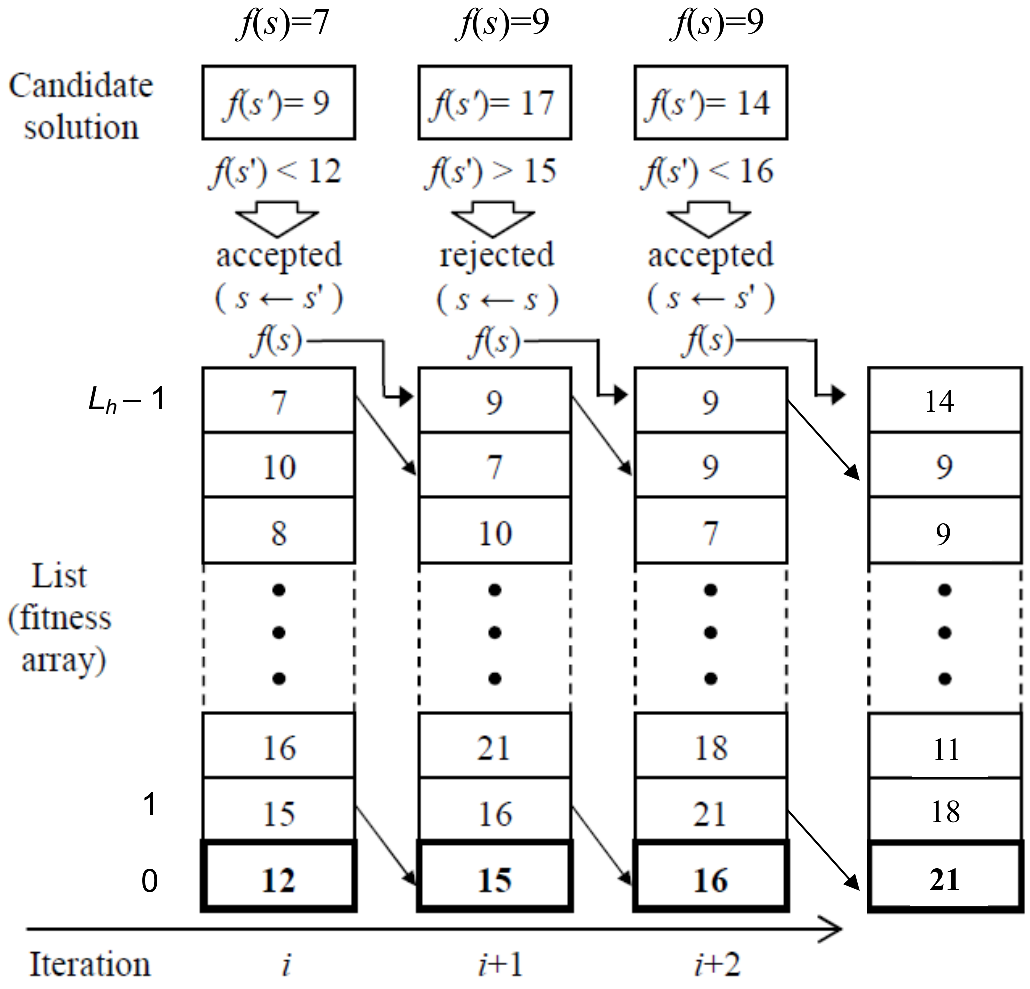
A Sample Run –
Late Acceptance Algorithm

linear queue implementation

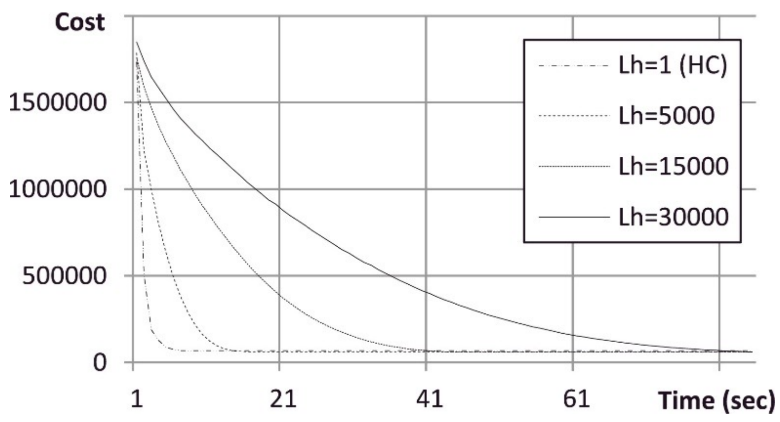


A Sample Run –
Late Acceptance Algorithm

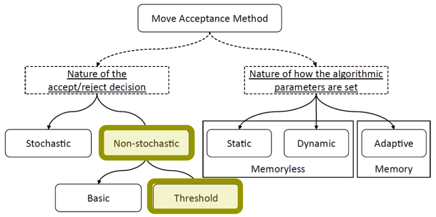
linear queue implementation



- Simple approach with a single parameter, yet the setting of the list length parameter influences the overall performance of the algorithm



Non-stochastic Threshold Move Acceptance – minimising *F*



$s_0 = generateInitialSolution(\,);$
s, s_{best}= s_0;
// initialise other elevant parameters if there is any
REPEAT
    s' = makeMove(s, memory); // choose a neighbour of s*
    threshold-ε = moveAcceptance-getThreshold(s, s', memory);
    if (*f*(s')≤ threshold-ε) s =s'; // else reject new solution s'
    $\boldsymbol{s_{best} = updateBest}$**(s, s'); // keep track of** $\boldsymbol{s_{best}}$
UNTIL(termination conditions are satisfied);
RETURN s_{best}

- Determine a threshold which is in the vicinity of a chosen solution quality, e.g. the quality of the best solution found so far or current solution, and accept all solutions below that threshold

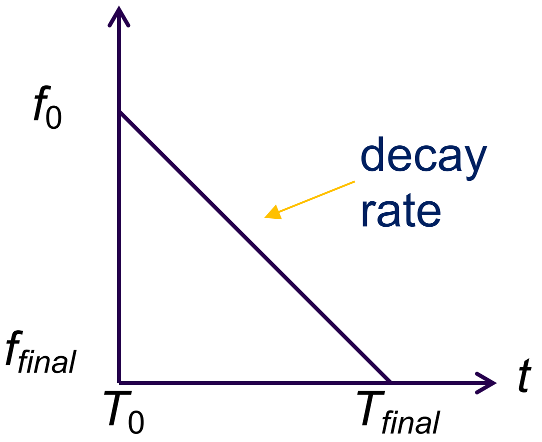
Non-stochastic Threshold Move Acceptance – Examples

- **Static**
  - Accept a worsening solution if the worsening of the objective value is no worse than a fixed value
- **Dynamic**
  - Great Deluge
  - Flex Deluge
- **Adaptive**
  - Record to record travel (RRT)
  - Extended Great Deluge
  - Modified Great Deluge

New Optimization Heuristics. The Great Deluge Algorithm and the Record-to-Record Travel. Dueck, Gunter, Journal of Computational Physics, Volume 104, Issue 1, January 1993, Pages 86-92

Great Deluge – minimisation

Example:



THE GREAT DELUGE ALGORITHM FOR MAXIMIZATION.

Choose an initial configuration
choose the "rain speed" DOWN>0
choose the initial WATER-LEVEL > 0
Opt: choose a new configuration which is a stochastic
 small
 perturbation of the old configuration
 compute E := quality (new configuration) → (E is f(.))
 IF E< WATER-LEVEL
 THEN old configuration := new configuration
 WATER-LEVEL := WATER-LEVEL – DOWN
 IF a long time no increase in quality or too many
 iterations
 THEN stop
GOTO Opt

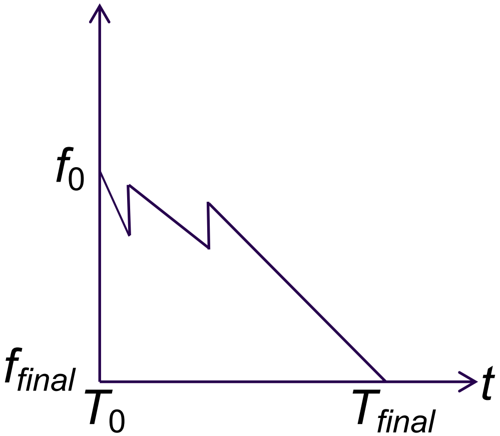
decay
rate

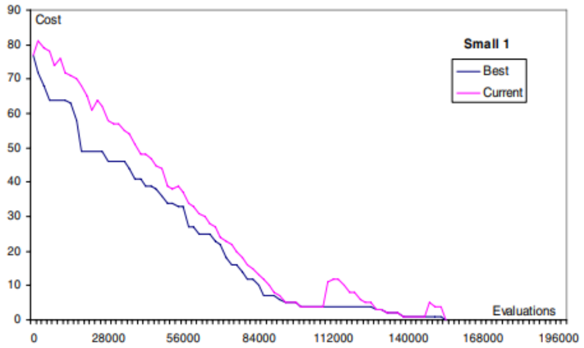
Gunter Dueck, Tobias Scheuer, [Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing], Journal of Computational Physics, Volume 90, Issue 1, September 1990, Pages 161-175

Extended Great Deluge – minimisation

- Feedback is received during the search and decay-rate is updated/reset accordingly whenever there is no improvement for a *long time*.

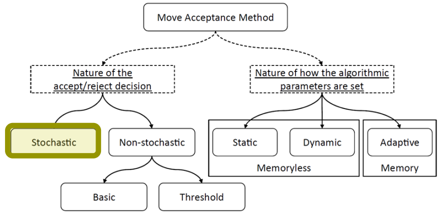
```
Set the initial solution s using a construction heuristic;
Calculate initial cost function f(s)
Set Initial Boundary Level B0 = f(s)
Set initial decay Rate ΔB based on Cooling Parameter
While stopping criteria not met do
Apply neighbourhood Heuristic S* on S
      Calculate f(s*)
        If f(s*) <= f(s) or (f(s*) <= B Then
          Accept s = s*
        Lower Boundary B = B - ΔB
        If no improvement in given time T Then
          Reset Boundary Level B0 = f(s)
          Set new decay rate ΔB based on Secondary
               Cooling Parameter
```





[1] An Extended Great Deluge Approach to the Examination Timetabling Problem, B. McCollum, P.J. McMullan, A. J. Parkes, E.K. Burke, S. Abdullah [PDF]
[2] An Extended Implementation of the Great Deluge Algorithm for Course Timetabling, Paul McMullan [PDF]

Stochastic Move Acceptance



$s_0 = generateInitialSolution(\,);$

s, s_{best}= s_0;

REPEAT

  s' = makeMove(s, memory); // choose a neighbour of s

  P = moveAcceptance-getAcceptanceProbability(s, s', memory);

  r = getRandomValue(); // e.g., a uniform random value $\in$[0,1)

  if (*f*(s').isBetterThan(*f*(s)) || (r < P))

    s = s'; // else reject new solution s'

  $s_{best} \leftarrow updateBest$(s, s′); // keep track of s_{best}

UNTIL (termination conditions are satisfied);

RETURN $s_{best};$

Stochastic Move Acceptance – Examples

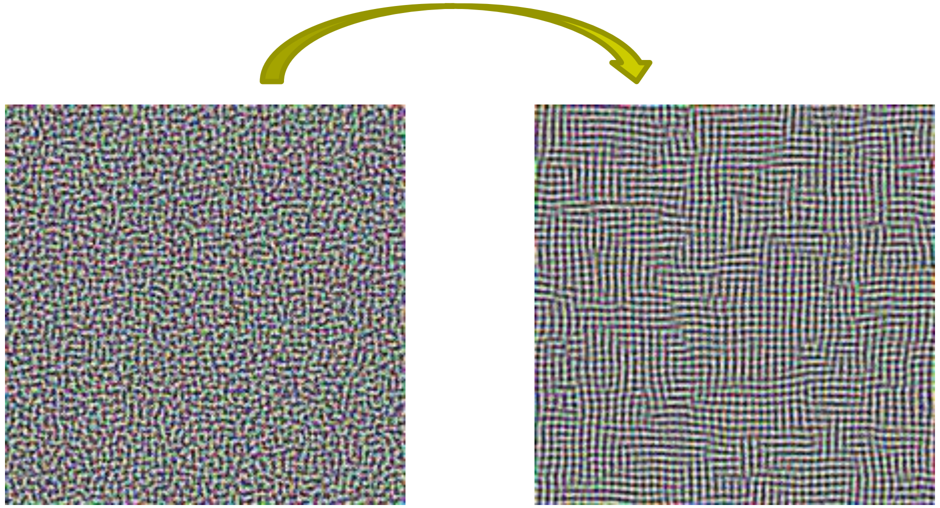
- **Static**
  - E.g., Naive Acceptance: P is fixed, e.g. if improving P=1.0, else P=0.5
- **Dynamic**
  - E.g., Simulated Annealing: P changes in time with respect to the difference in the quality of current and previous solutions (see the next slides). Temperature parameter is changes dynamically.
- **Adaptive**
  - E.g., Simulated Annealing with reheating: P is modified via increasing temperature time to time causing partial restart – increasing the probability of acceptance of non-improving solutions

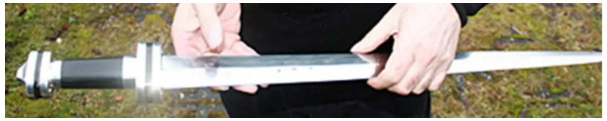
Simulated Annealing

- A stochastic local search algorithm inspired by the physical process of annealing (Kirkpatrick et al. 1983)
- Easy to implement
- Achieves good performance given sufficient running time
- Requires a good parameter setting for improved performance
- Has interesting theoretical properties (convergence), but these are of very limited practical relevance

Simulated Annealing – Analogy

- If a liquid material **cools and anneals** too quickly, then the material will solidify into a **sub-optimal** configuration.
- If the liquid material **cools slowly**, the crystals within the material will solidify **optimally into a state of minimum energy** (i.e. ground state). – This ground state corresponds to the minimum of the cost function in an optimisation problem.





Simulated Annealing
Local Search Metaheuristic

INPUT: T_0, T_{final}
$s_0 \leftarrow generateInitialSolution(\,);$
$T \leftarrow T_0;$ // initialise temperature to T_0
$s_{best} \leftarrow s_0; s \leftarrow s_0;$ // set s and s_{best} to initial solution
REPEAT
    $s' \leftarrow perturbation(s);$ // choose a neighbouring solution of s
    $\Delta = f(s') - f(s);$
    $r \leftarrow random \in [0,1];$ // get a uniform random number in the range [0,1)
    if($\Delta < 0 \,||\, r < P(\Delta, T)$) { // accept s' if solution is non-worsening or with Boltzmann probability
        $s \leftarrow s';$
    }
    $\mathrm{s}_{\mathrm{best}} \leftarrow updateBest(\,);$ // keep track of best solution
    $T \leftarrow coolTemperature(\,);$ // decrease the temperature according to cooling schedule
UNTIL (Termination conditions are satisfied);
Return $s_{best};$

Accepting Moves using SA

- Improving moves are accepted
- Worsening moves are accepted using the Metropolis criterion at a given temperature T

$\Delta = F(S_{new}) - F(S_{old})$ minimise{ F }

An inferior solution S_{new} would yield $\Delta > 0$

Accept it with a Boltzman probability of

$$P(\Delta, T) = e^{\frac{-\Delta}{T}}$$

U(0,1) generates a random number in [0,1)

Accept if U(0,1) < $P(\Delta, T)$

Cooling/Annealing

$$P(\Delta,T)=e^{\frac{-\Delta}{T}}$$

- Temperature T is slowly decreased
  - T is initially high - many inferior moves are accepted
  - T is decreasing - inferior moves are nearly always rejected
- As the temperature T decreases, the probability of accepting worsening moves decreases.

	Probability of acceptance $P(\Delta,T)$		
T	Δ=1	Δ=2	Δ=3
10	0.9	0.82	0.74
1	0.37	0.14	0.05
0.25	0.018	0.0003	0.000006
0.1	0.00005	2×10^{-9}	9×10^{-14}

$\Delta > 0$ inferior solution

$-\Delta < 0$ $\quad T \searrow \quad -\frac{\Delta}{T} \searrow \quad e^{\frac{-\Delta}{T}} \searrow$

SA – Cooling Schedule

- Starting Temperature
- Final Temperature
- Temperature Decrement
- Iterations at each temperature

SA – Cooling Schedule II

- Starting Temperature (T_0)
  - *hot enough*: to allow almost all neighbours
  - *not so hot*: random search for sometime
  - Estimate a suitable starting temperature:
    - Reduce quickly to 60% of worse moves are accepted
    - Use this as the starting temperature
- Final Temperature
  - Usually 0, however in practise, not necessary
  - T is low: accepting a worse move is almost the same as T=0
  - The stopping criteria: either be a suitably low T, or “frozen” at the current T (i.e. no worse moves are accepted)

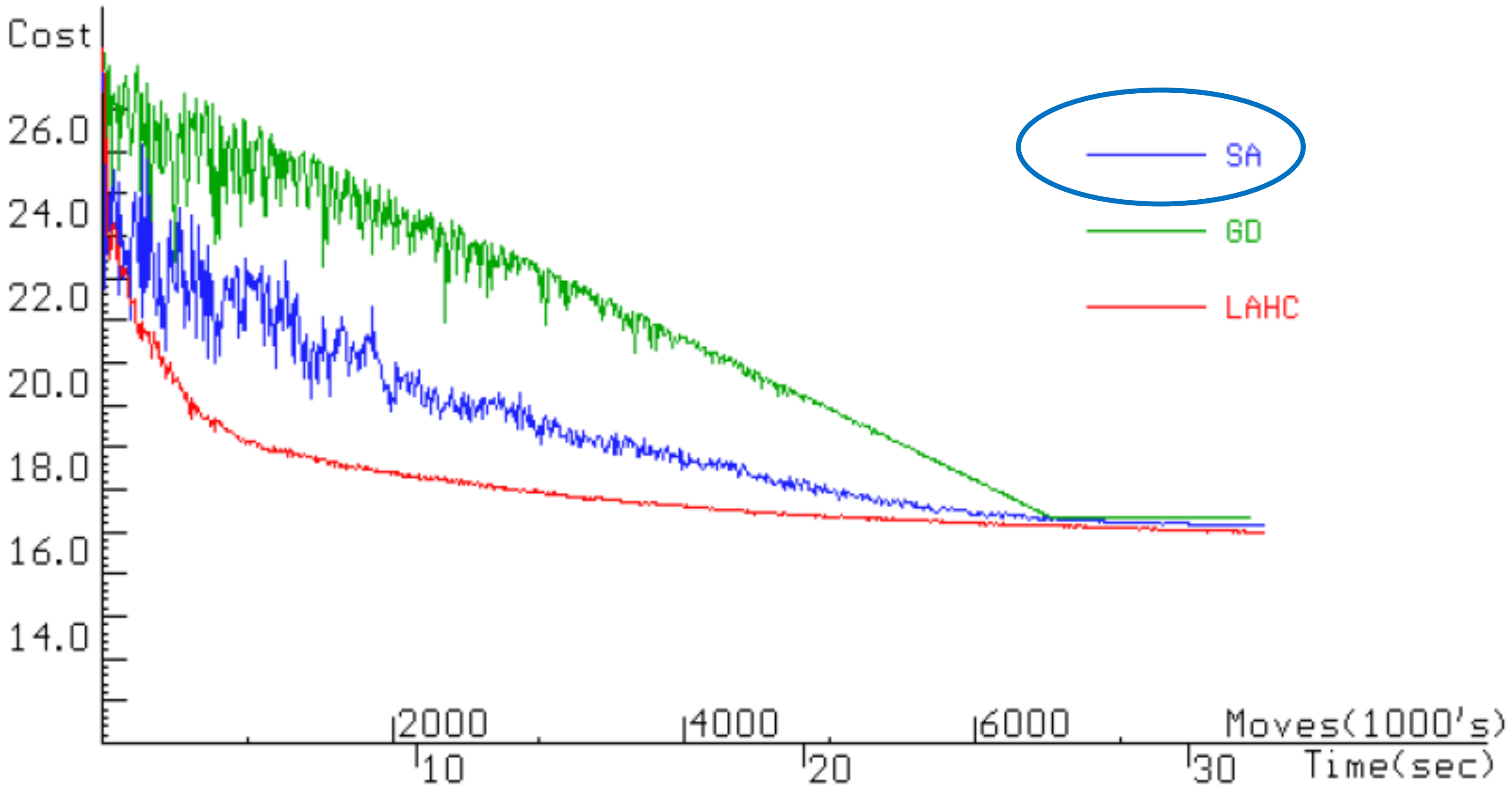
SA – Cooling Schedule III

- Temperature Decrement
  - **Linear**: $T = T - x$
  - **Geometric**: $T = T * \alpha$
    - Experience: α is typically in the interval [0.9, 0.99]
  - **Lundy Mees**: $T = \dfrac{T}{1 + \beta T}$
    - One iteration at each T, but decrease T very slowly. Experience: β is typically a very small value, that is close to 0 (e.g., 0.0001)

SA – Cooling Schedule IV

- Iterations at each temperature
  - One iteration at each *T*
  - A constant number of iterations at each *T*
  - Compromise
    - Either: a large number of iterations at a few *T*s, or
    - A small number of iterations at many *T*s, or
    - A balance between the two
  - Dynamically change the no. of iterations
    - At higher *T*s: less no. of iterations
    - At lower *T*s: large no. of iterations, local optimum fully exploited
- Reheating, if stuck at a local optimum for a while, increase the current temperature with a certain rate

Behaviour of Simulated Annealing – An Example



Simulated Annealing with Geometric Cooling

INPUT: T_0 (> 0), α (cooling rate<1.0)
Generate an initial solution S_0 using some heuristics
Set $S_k = S_{best} = S_0$, k=0;

REPEAT
Select $S_{new} \in \mathcal{N}(S_k)$ // Make a move from S_k to S_{new} based on $\mathcal{N}$
If $F(S_{new}) < F(S_k)$ **then** $S_{k+1} = S_{new}$ // an improving move is made
else // A worsening solution is obtained
generate a random uniform number in (0,1], U(0,1)
If U(0,1) < $e^{-\frac{F(S_{new})-F(S_k)}{T_k}}$ **then** $S_{k+1} = S_{new}$ // Accept worsening move
else $S_{k+1} = S_k$ // Reject worsening move
// Keep track of the best solution found so far
If $F(S_{new}) < F(S_{best})$ **then** $S_{best} = S_{new}$
$T_{k+1} = \alpha T_k$; // *geometric cooling:* multiply previous temp with α (value<1.0)
$k = k+1$; // one iteration at each T
UNTIL (*stopping condition* = true)

Exercise

- Consider the MAX-SAT problem instance:
  - $(a \vee b) \wedge (\neg d \vee f) \wedge (\neg a \vee c) \wedge (b \vee \neg f) \wedge (\neg b \vee c) \wedge (c \vee e)$
- Objective function: number of unsatisfied clauses
- Apply the simulated annealing to the instance starting out with the 100100 as an initial solution for 3 SA steps/iterations.
- **Neighbourhood operator**: perform random 1-bit flip
- Choose α = 0.9 and T_0 = 0.9
- Use the following numbers in the given order as **random numbers where appropriate**
  - for choosing a random literal i in [1..6]: <2, 3, 1, 6, …>to apply 1-bit flip of (i)th literal’s truth asssigment in the candidate solution
  - for $U(0,1)$: <0.47, 0.089, ...>

Running of SA

0 1 2

$$\overset{0}{(a \vee b)} \wedge \overset{1}{(\neg d \vee f)} \wedge \overset{2}{(\neg a \vee c)}$$
$$\wedge \underset{3}{(b \vee \neg f)} \wedge \underset{4}{(\neg b \vee c)} \wedge \underset{5}{(c \vee e)}$$

3 4 5

$S_{best} = S_0 = \overset{a\,b\,c\,d\,e\,f}{100100}$, $T_0 = 0.9$

$F(S_0) = F(100100)=3$ (Clause SAT: 012345) $= F(S_{best})$

<u>k=0</u>

$S_{new} \leftarrow \mathcal{N}(S_0)$:1**1**0100 ➢ Rand[1..6]: <**2**, 3, 1, 6, …>

$S_{new} = 110100$, Clause SAT: 012345: $F(S_{new}) = 4 > F(S_0) = 3$

$U(0,1) = 0.47 > e^{-\frac{4-3}{0.9}} = 0.33$ ➢ U(0,1) : <**0.47**, 0.089, ...>

$S_1 = 100100$ ➢ reject S_{new} : $S_1 = S_0$

$F(S_{new}) = 4 > F(S_{best})=3$ ➢ S_{best} does not change

$T_1 = \alpha\, T_0 = 0.9 \cdot 0.9 = 0.81$ ➢ update temperature

<u>k=1</u>

$S_{new} \leftarrow \mathcal{N}(S_1)$:10**1**100 ➢ Rand[1..6]: <~~2~~, **3**, 1, 6, …>

$S_{new} = 101100$, Clause SAT: 012345: $F(S_{new}) = 1 < F(S_1) = 3$

$S_2 = 101100$ ➢ accept S_{new} : $S_2 = S_{new}$

$F(S_{new}) = 1 < F(S_{best})$

$S_{best} = 101100$ ➢ update best solution

$F(S_{best}) = 1$

$T_2 = \alpha\, T_1 = 0.9 \cdot 0.81 = 0.729$ ➢ update temperature

Running of SA (cont.)

***k*=2**

$S_{new} \leftarrow \mathcal{N}(S_2)$: **0**01100 ➢ Rand[1..6]: <~~2~~, ~~3~~, **1**, 6, …>

S_{new} = 001100, 012345, $F(S_{new}) = 2 > F(S_2) = 1$

$U(0,1) = 0.089 < e^{-\frac{2-1}{0.729}} = 0.253$ ➢ U(0,1) : <~~0.17~~, **0.089**, ...>

S_3= 001100 ➢ accept S_{new} : $S_3 = S_{new}$

$F(S_{new}) = 2 > F(S_{best}) =1$ ➢ S_{best} does not change (101100)

$T_3 = \alpha\, T_2 = 0.9 \cdot 0.729 = 0.6561$

...

2. Parameter Setting Issues and Tuning Methods

COM2001/2011: Artificial Intelligence Methods

Ender Özcan

Lecture 4

Computational Optimisation & Learning Lab

The University of Nottingham

UNITED KINGDOM · CHINA · MALAYSIA

Metaheuristics

Local Search	[Kirkpatrick, 1983]	Simulated Annealing (SA)
	[Glover, 1986]	Tabu Search (TS)
	[Voudouris, 1997]	Guided Local Search (GLS)
	[Stutzle, 1999]	Iterated Local Search (ILS)
	[Mladenovic, 1999]	Variable Neighborhood Search (VNS)
Population-based	[Holland, 1975]	Genetic Algorithm (GA)
	[Smith, 1980]	Genetic Programming (GP)
	[Goldberg, 1989]	Genetic and Evolutionary Computation (EC)
	[Moscato, 1989]	Memetic Algorithm (MA)
	[Kennedy and Eberhart, 1995]	Particle Swarm Optimisation (PSO)
Constructive	[Dorigo, 1992]	Ant Colony Optimisation (ACO)
	[Resende, 1995]	Greedy Randomized Adaptive Search Procedure (GRASP)

Metaheuristics

<table>
<tr><th></th><th></th><th><u>Examples of Parameters</u></th></tr>
<tr><td rowspan="5">Local Search</td><td>SA</td><td>T_0 (initial temperature), α (cooling rate)</td></tr>
<tr><td>TS</td><td>Tabu list size (tabu tenure)</td></tr>
<tr><td>GLS</td><td>λ (intensification control), a (coefficient)</td></tr>
<tr><td>ILS</td><td><u>perturbation, perturbation strength</u></td></tr>
<tr><td>VNS</td><td>k_{min}, k_{max} (smallest, largest neighbourhood size)</td></tr>
<tr><td rowspan="5">Population-based</td><td>GA</td><td rowspan="4">population size, mutation probability, mutation</td></tr>
<tr><td>GP</td></tr>
<tr><td>EC</td></tr>
<tr><td>MA</td></tr>
<tr><td>PSO</td><td>number of particles, V_{max} (maximum velocity)</td></tr>
<tr><td rowspan="2">Constructive</td><td>ACO</td><td>weight of pheromone, evaporation rate</td></tr>
<tr><td>GRASP</td><td>restricted candidate list parameter</td></tr>
</table>

Parameter Types

- Categorical/symbolic/structural parameters
  - Choice of initialisation method, choice of mutation,…
- Ordinal parameters
  - Neighbourhoods (e.g., small, medium, large),..
- Numerical/behavioural parameters
  - integer, real-valued, …
  - population sizes, evaporation rates,…
  - values may depend on the setting of categorical or ordinal parameters

Parameter Setting Methods

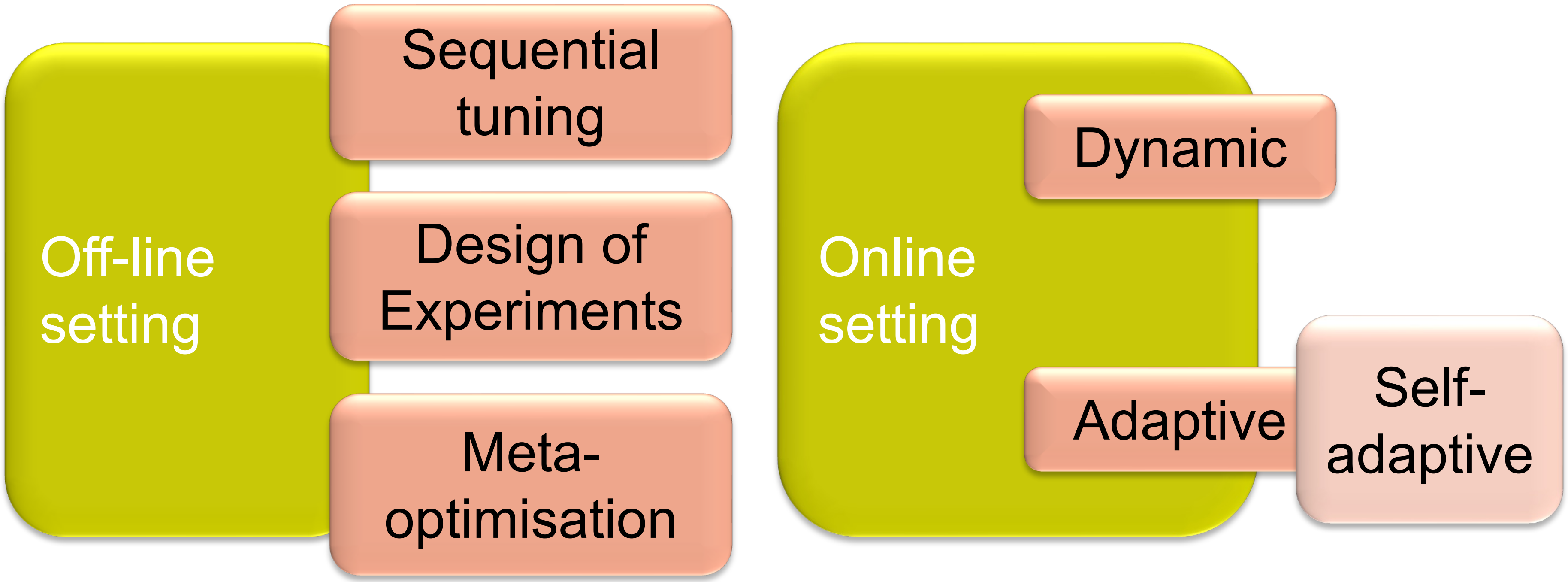
- **Parameter tuning**: Finding the best initial settings for a set of parameters before the search process starts (*off-line*). E.g., fixing the mutation strength in ILS, mutation probability in genetic algorithms, etc.
  - The initial parameter setting influences the performance of a metaheuristic
- **Parameter control**: Managing the settings of parameters during the search process (*online*) (dynamic, adaptive, self-adaptive). E.g., changing the mutation strength in ILS, changing the mutation probability in genetic algorithms during the search process
  - Controlling parameter setting could yield a system which is not sensitive to its initial setting

A Classification

Parameter Setting

Parameter Tuning

Parameter Control

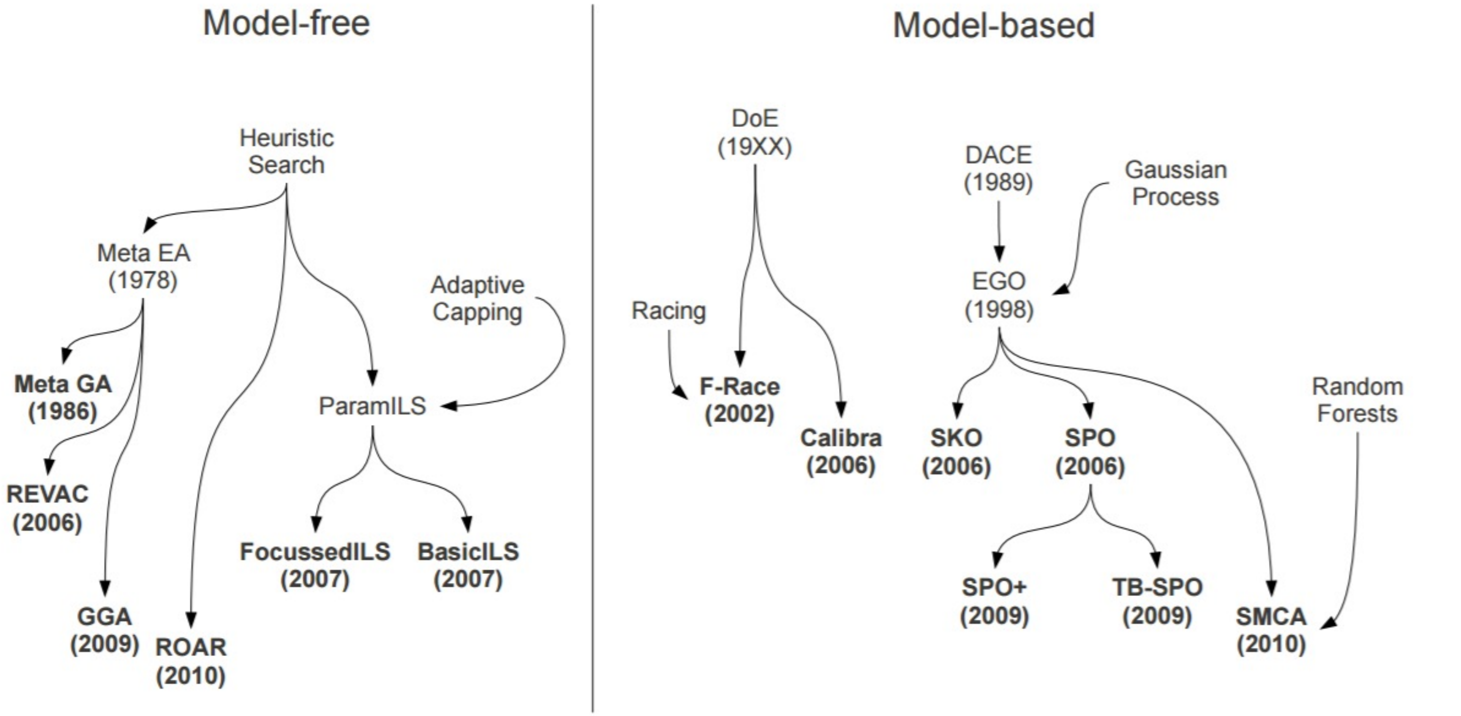


Parameter Tuning Methods

Example: Assume
$\phi \in \{0, 0.3, 0.5, 0.8, 1.0\}$ and
$\delta \in \{20, 40, 50, 60, 80\}$

- Traditional approaches
  - Use of an arbitrary setting (e.g., ϕ =*0* and δ =80)
  - Trial&error with settings based on intuition
  - Use of theoretical studies
  - A mixture of above
- Sequential tuning: fix parameter values successively (e.g., fix fixing δ =20 and tune ϕ that is try {0, 0.3, 0.5, 0.8, 1.0}, then fixing the best setting for ϕ from the previous trials and tune δ that is try {20, 40, 50, 60, 80})
- Design of experiments
- Meta-optimisation: use a metaheuristic to obtain "optimal" parameter settings

Automated Parameter Tuning



Source: http://www.mff.cuni.cz/veda/konference/wds/proc/pdf10/WDS10_109_i1_Dobslaw.pdf

I-race (download: http://iridia.ulb.ac.be/irace/)
ParamILS (download: http://www.cs.ubc.ca/labs/beta/Projects/ParamILS/)
SPOT (download: https://cran.r-project.org/web/packages/SPOT/index.html)

Design of Experiments (DoE)

- A systematic method (controlled experiments) to determine the relationship between controllable and uncontrollable *factors* (inputs to the process, variables) affecting a process (e.g., running of an algorithm), their *levels* (settings) and the *response* (output) of that process (e.g., quality of solutions obtained – performance of an algorithm). (Fisher 1926, 1935)
- Important outcomes are measured and analysed to determine the factors and their settings that will provide the best overall outcome
  - E.g., Two factors; $\phi \in \{0, 0.3, 0.5, 0.8, 1.0\}$ and $\delta \in \{20, 40, 50, 60, 80\}$, each with 5 levels – at least 5^2 runs required

Fractional Factorial Designs

- Assuming the number of factors is *k* in an *n* level factorial design, then number of runs for even a single replicate of the n^k design becomes very large.
    - E.g, a replicate of an 8 factor two level experiment would require 2^8 =256 runs. If a run consists of 31 trials each taking 5 min, such an experiment would take ~28 days (1 instance)
- Fractional factorial designs can be used in these cases to draw out valuable conclusions from fewer runs.
- Key observation: Responses are often affected by a small number of main effects and lower order interactions, while higher order interactions are relatively unimportant.

Design of Experiments – Sampling

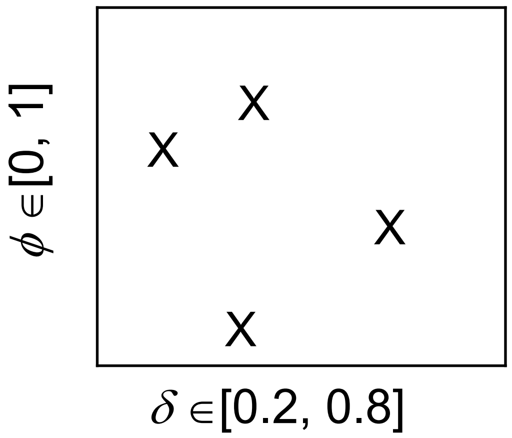
- Whenever factorial design is not possible, sampling is performed:
  - **Random**
  - **Latin Hyper-cube**
  - **Orthogonal**
- Example: Assume that we have two parameters,

$$\phi \in [0, 1] \text{ and } \delta \in [0.2, 0.8]$$

Random Sampling

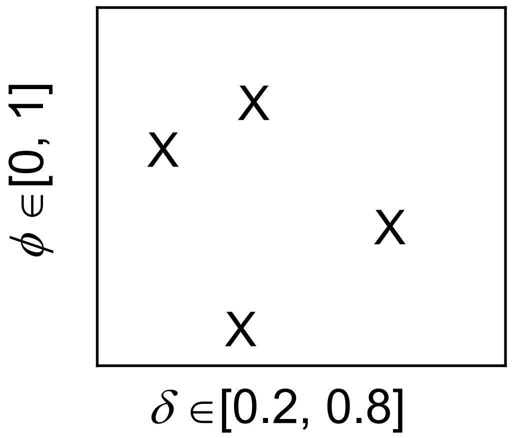
- Generate each sample point independently (M)

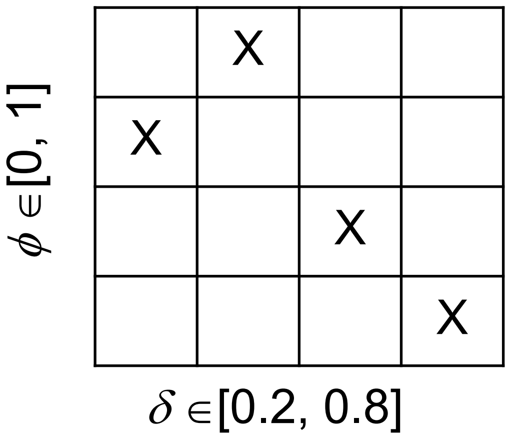
- Example:



Latin Hypercube Sampling

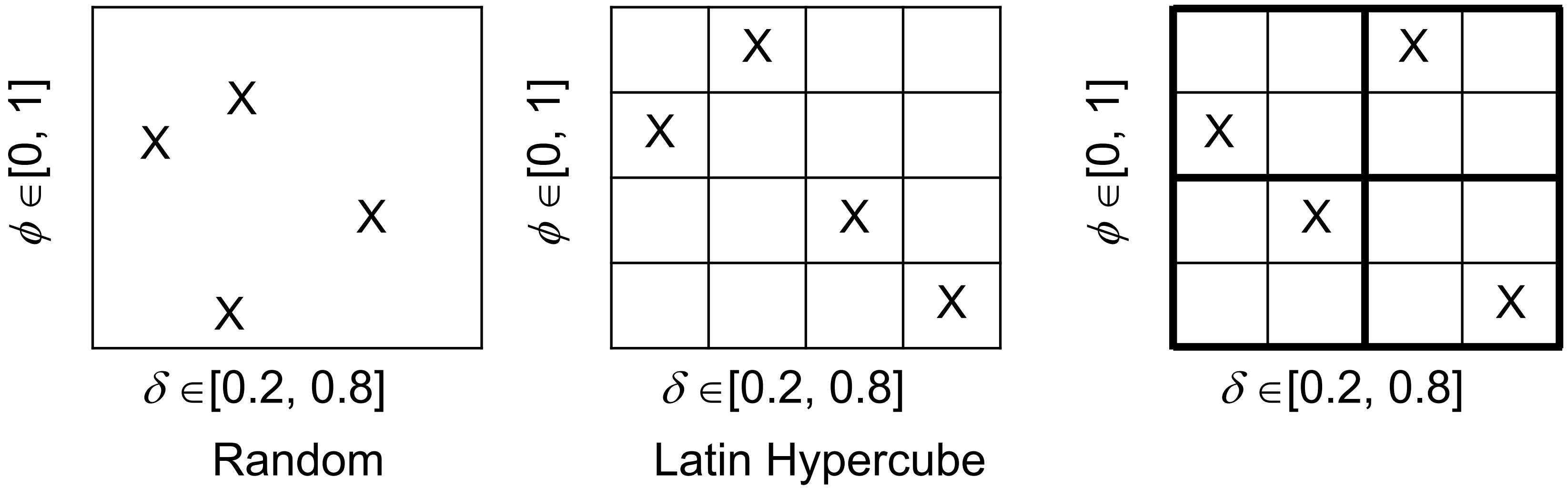
- Decide the number of sample points (M) for N variables and for each sample point remember in which row and column the sample point was taken
- Example:





Orthogonal Sampling

- The sample space is divided into equally probable subspaces. Sample points simultaneously, ensuring they form an ensemble of Latin Hypercube sample
- Example:



Taguchi Orthogonal Arrays Method for Parameter Tuning

- Developed by Genichi Taguchi to improve the quality of manufactured goods initially, then applied to problems from the other fields
- Aim: make a “product” or “process” less variable (more *robust*) in the face of variation over which we have little or no control.
- Taguchi method is a structured statistical (experimental design) method for determining the *best* combination of parameter settings to achieve certain objective(s)

Taguchi Orthogonal Arrays Method for Parameter Tuning II

- Best when there are an intermediate number of parameters/variables/ factors (3 to 50), few interactions between variables, and when only a few variables contribute significantly.
  - E.g., 3 factors and 2 levels (settings) per factor: 2^3 combinations
- Taguchi's orthogonal arrays are highly fractional orthogonal designs, which can be used to estimate main effects using only a few experimental runs (which can consist of multiple trials). E.g.,

L4 (2^3)

Run	Columns		
	1	2	3
1	1	1	1
2	1	2	2
3	2	1	2
4	2	2	1

L8 (2^7)

Run	Columns						
	1	2	3	4	5	6	7
1	1	1	1	1	1	1	1
2	1	1	1	2	2	2	2
3	1	2	2	1	1	2	2
4	1	2	2	2	2	1	1
5	2	1	2	1	2	1	2
6	2	1	2	2	1	2	1
7	2	2	1	1	2	2	1
8	2	2	1	2	1	1	2

L9 (3^4)

Run	Columns			
	1	2	3	4
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	2	1	2	3
5	2	2	3	1
6	2	3	1	2
7	3	1	3	2
8	3	2	1	3
9	3	3	2	1

Taguchi Orthogonal Arrays Method for Parameter Tuning – Main Steps

1. Selection of control parameters (independent variables/factors)
2. Selection of number of level settings for each parameter
3. Select a suitable orthogonal array based on the number of parameters and levels

Planning

4. Conduct the experiments using the algorithm on the selected subset of test instances

Conduct

5. Analyse the results
6. Determine the optimum levels for the individual parameters

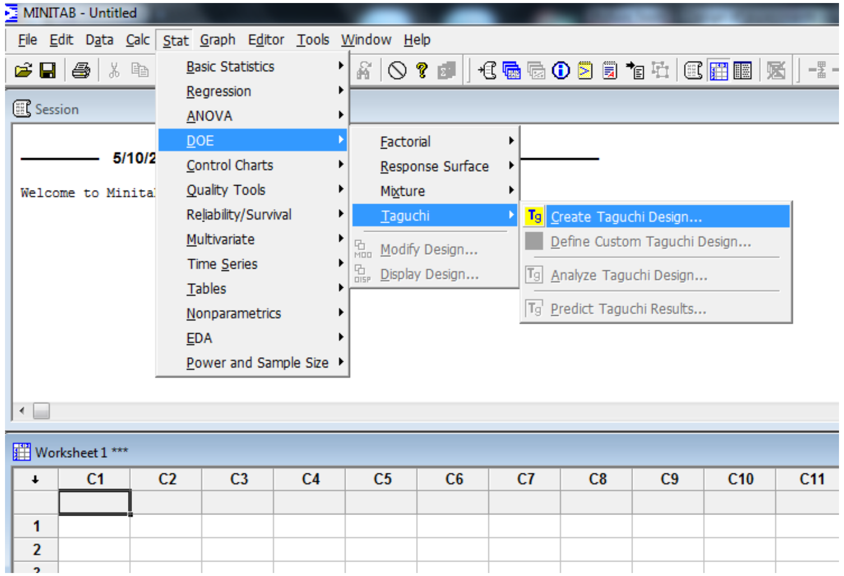
Analysis

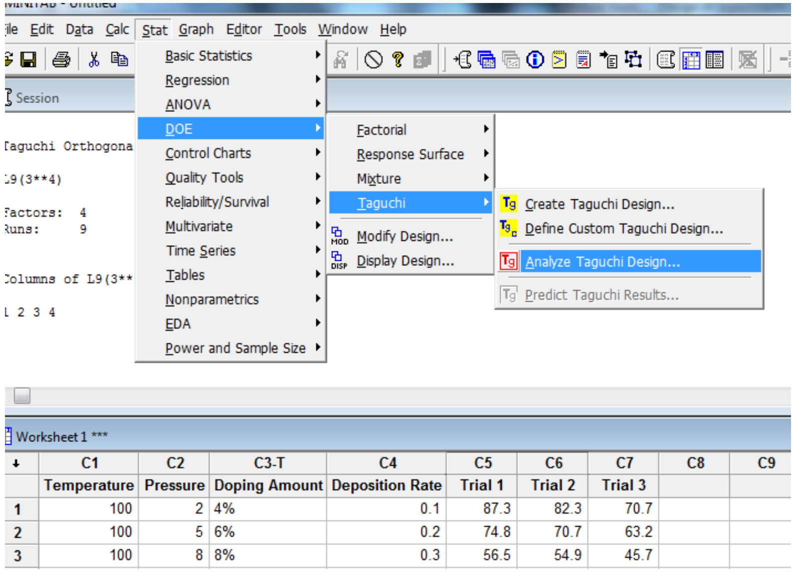
7. Confirmation experiment

Validation

<u>Use the same configuration for the rest of the experiments</u>

Taguchi Orthogonal Arrays Method – Software Packages





Minitab

- MATLAB, Minitab, R, Octave, JMP, DOE++, …

1. Selection of control parameters (independent variables)
2. Selection of number of level settings for each parameter
3. Select a suitable orthogonal array based on the number of parameters and levels

Planning

- Assume that we have a metaheuristic with 4 parameters (factors):
  - param1 $\in [0,1.0]$ ($\mathbb{R}$)
  - param2 $\in [0,1.0]$ ($\mathbb{R}$)
  - param3 $\in [1..80]$ ($\mathbb{Z}^+$)
  - param4 $\in [1..5]$ ($\mathbb{Z}^+$)

Parameter levels

		Value Options
[5 levels]	param1	{0.2, 0.4, 0.6, 0.8, 1.0}
[5 levels]	param2	{0.2, 0.4, 0.6, 0.8, 1.0}
[5 levels]	param3	{5, 10, 20, 40, 80}
[4 levels]	param4	{2, 3, 4, 5}

- Choosing a suitable Taguchi orthogonal array design:
  - We have 4 factors with a maximum of 5 levels for each factor

- L4: Three two-level factors
- L8: Seven two-level factors
- L9 : Four three-level factors
- L12: Eleven two-level factors
- L16: Fifteen two-level factors
- L16b: Five four-level factors
- L18: One two-level and seven three-level factors
- L25: Six five-level factors
- L27: Thirteen three-level factors
- L32: Thirty-two two-level factors

⋮

X1	X2	X3	X4	X5	X6
-	-	-	-	-	-
1	1	1	1	1	1
1	2	2	2	2	2
1	3	3	3	3	3
1	4	4	4	4	4
1	5	5	5	5	5
2	1	2	3	4	5
2	2	3	4	5	1
2	3	4	5	1	2
2	4	5	1	2	3
2	5	1	2	3	4
3	1	3	5	2	4
3	2	4	1	3	5
3	3	5	2	4	1
3	4	1	3	5	2
3	5	2	4	1	3
4	1	4	2	5	3
4	2	5	3	1	4
4	3	1	4	2	5
4	4	2	5	3	1
4	5	3	1	4	2
5	1	5	4	3	2
5	2	1	5	4	3
5	3	2	1	5	4
5	4	3	2	1	5
5	5	4	3	2	1

Taguchi orthogonal array

Experiment number	param1	param2	param3	param4
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	1	4	4	4
5	1	5	5	2
6	2	1	2	3
7	2	2	3	4
8	2	3	4	1
9	2	4	5	1
10	2	5	1	2
11	3	1	3	4
12	3	2	4	1
13	3	3	5	2
14	3	4	1	3
15	3	5	2	4
16	4	1	4	2
17	4	2	5	3
18	4	3	1	4
19	4	4	2	2
20	4	5	3	1
21	5	1	5	4
22	5	2	1	3
23	5	3	2	1
24	5	4	3	2
25	5	5	4	3

Planning

Parameter levels

		Value Options
[5 levels]	param1	{0.2, 0.4, 0.6, 0.8, 1.0}
[5 levels]	param2	{0.2, 0.4, 0.6, 0.8, 1.0}
[5 levels]	param3	{5, 10, 20, 40, 80}
[4 levels]	param4	{2, 3, 4, 5}

1 2 3 4

Taguchi orthogonal array

Configuration ID		param1	param2	param3	param4
1	e.g.;	1 (0.2)	1 (0.2)	1 (5)	1(2)
2		1	2	2	2
3		1	3	3	3
4		1	4	4	4
5		1	5	5	2
6		2	1	2	3
7		2	2	3	4
8		2	3	4	1
9		2	4	5	1
10		2	5	1	2
11		3	1	3	4
12		3	2	4	1
13		3	3	5	2
14		3	4	1	3
15		3	5	2	4
16		4	1	4	2
17		4	2	5	3
18		4	3	1	4
19		4	4	2	2
20		4	5	3	1
21		5	1	5	4
22		5	2	1	3
23		5	3	2	1
24		5	4	3	2
25	e.g.;	5 (1.0)	5 (1.0)	4(40)	3(4)

Conduct Experiments

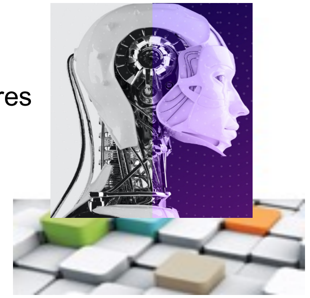
- Run experiments, say for 30 times using the algorithm with each setting (potentially on multiple ‘training’ instances)
- Use a performance metric, e.g., record Formula 1 score for each run/trial (higher the better), where the top 8 algorithms score **10, 8, 6, 5, 4, 3, 2** and **1** point(s).

Configuration ID	param1	param2	param3	param4
1	0.2	0.2	5	2
2	0.2	0.4	10	3
3	0.2	0.6	20	4
4	0.2	0.8	40	5
5	0.2	1.0	80	3
6	0.4	0.2	10	4
7	0.4	0.4	20	5
8	0.4	0.6	40	2
9	0.4	0.8	80	2
10	0.4	1.0	5	3
11	0.6	0.2	20	5
12	0.6	0.4	40	2
13	0.6	0.6	80	3
14	0.6	0.8	5	4
15	0.6	1.0	10	5
16	0.8	0.2	40	3
17	0.8	0.4	80	4
18	0.8	0.6	5	5
19	0.8	0.8	10	3
20	0.8	1.0	20	2
21	1.0	0.2	80	5
22	1.0	0.4	5	4
23	1.0	0.6	10	2
24	1.0	0.8	20	3
25	1.0	1.0	40	4

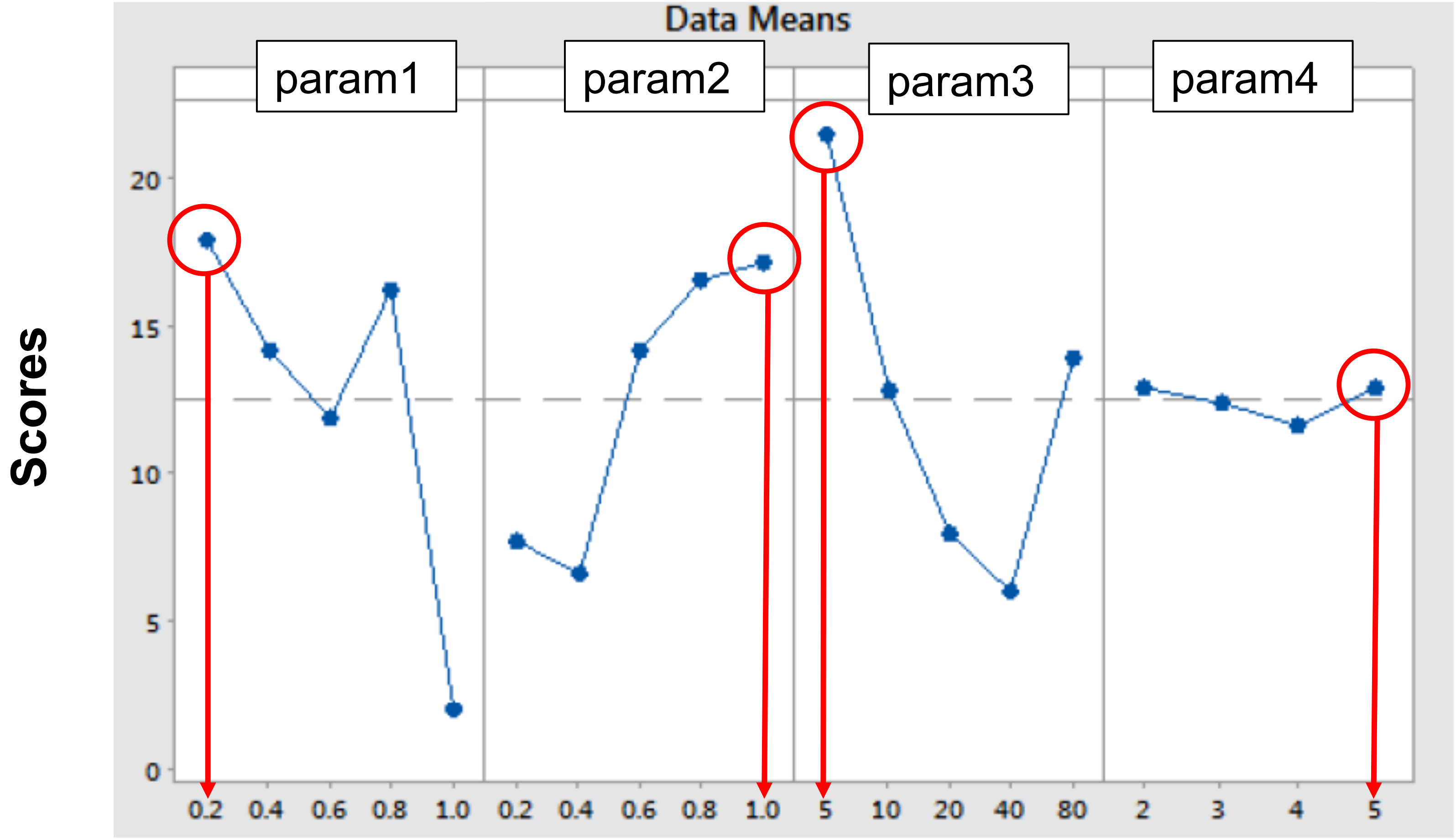
Conduct Experiments

- Collect results using an appropriate performance indicator: e.g., obtain mean F1 *score* per run for each algorithm with a specific parameter combination setting per instance, and sum those scores up from all instances
- For analysis: Main effect of param3 for the setting of 20 is the *mean score* for all experiments with that setting: (17.5+2.5+0+18.38+1.88)/5 =8.052

Configuration ID	param1	param2	param3	param4	Performance Indicator/Scores
1	0.2	0.2	5	2	24.88
2	0.2	0.4	10	3	14
3	0.2	0.6	**20**	4	**17.50**
4	0.2	0.8	40	5	17.88
5	0.2	1.0	80	3	15.50
6	0.4	0.2	10	4	9.88
7	0.4	0.4	**20**	5	**2.50**
8	0.4	0.6	40	2	8.88
9	0.4	0.8	80	2	22.50
10	0.4	1.0	5	3	27.25
11	0.6	0.2	**20**	5	**0**
12	0.6	0.4	40	2	1
13	0.6	0.6	80	3	12
14	0.6	0.8	5	4	24.38
15	0.6	1.0	10	5	22.25
16	0.8	0.2	40	3	0
17	0.8	0.4	80	4	25.88
18	0.8	0.6	5	5	30.88
19	0.8	0.8	10	3	16.25
20	0.8	1.0	**20**	2	**18.38**
21	1.0	0.2	80	5	4
22	1.0	0.4	5	4	0
23	1.0	0.6	10	2	1.88
24	1.0	0.8	**20**	3	**1.88**
25	1.0	1.0	40	4	2.50

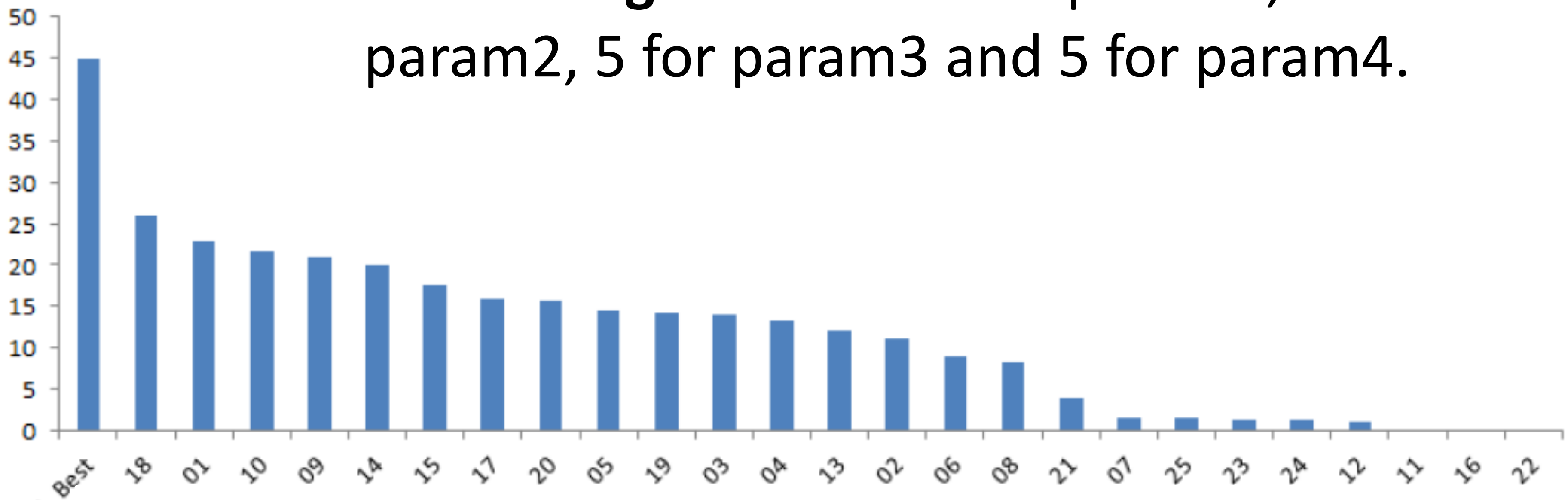


Analysis – Main Effects Plot



Validation: Confirmation Run

Best configuration: 0.2 for param1, 1.0 for param2, 5 for param3 and 5 for param4.



If failure, then chosen levels could be problematic needs to be reviewed.

Summary I

- It is not trivial which (meta)heuristic optimisation/search algorithm will perform better than the other one on a given problem domain
  - Experiments using a single or only small instances might not be realistic or true performance indicator of an algorithm
- There is a variety of statistical tools for the performance analyses of algorithms.
- Move acceptance methods as a part of local search metaheuristics can be used for escaping from the local optima, enabling acceptance of non-improving solutions.
  - Great Deluge and Simulated Annealing are well-known and well-studied such local search metaheuristics, and recently Late Acceptance. These are easy to implement methods and there are more elaborate variants.
  - Move acceptance methods vary depending on the mechanism and components they have for accepting a non-improving solution.

Summary II

- Many (meta)heuristic optimisation/search algorithms come with parameters which often require an initial setting (tuning)
  - Parameter tuning is possible, however it is time consuming.
  - There is a range of different techniques varying from manual/semi-automated experimental design methods to automated tuning, such as, Taguchi method, I-race.
  - Parameter control as an alternative to parameter tuning changes parameter values during the run of the algorithm
  - There is no guidance indicating which method is the best, however many studies show that parameter tuning/control often does improve the performance of an algorithm as compared to the variant where it is not used

Summary III

- Move acceptance methods can be hybridised:
  - Single stage strategy:
    - Use a decision mechanism to choose which move acceptance to employ at each step.
    - Use group decision making. E.g, combine simulated annealing, great deluge and late acceptance and apply majority vote.
  - Multi-stage
    - Use a different move acceptance at different stages of the search process. For example, use improving and equal until the search process gets stuck, then switch to simulated annealing.

HOME EXERCISE (SEE THE FORUM)

Soft Drink Bottling

- How would you represent this scheduling problem using the standard notation?
- single machine
- 4 flavours
- each flavour has its own filling time
- cleaning and changeover time between the bottling of successive flavours

  aim: to minimise cycle time,
  sufficient: to minimise the total changeover time

$$\begin{array}{c c} & \begin{array}{cccc} f_1 & f_2 & f_3 & f_4 \end{array} \\ \begin{array}{c} f_1 \\ f_2 \\ f_3 \\ f_4 \end{array} & \begin{pmatrix} - & 2 & 70 & 50 \\ 6 & - & 3 & 4 \\ 8 & 3 & - & 2 \\ 50 & 5 & 6 & - \end{pmatrix} \end{array}$$

$f_1 \to f_2 \to f_3 \to f_4 \to f_1$
2+3+2+50 = 57

$f_3 \to f_4 \to f_2 \to f_1 \to f_3$
2+5+6+70 = 83

$f_2 \to f_3 \to f_4 \to f_1 \to f_2$
3+2+50+2 = 57

$f_4 \to f_2 \to f_3 \to f_1 \to f_4$
5+3+8+50 = 66

optimal: $f_1 \to f_2 \to f_4 \to f_3 \to f_1$
2+4+6+8 = 20

Illustration of a Run of Tabu Search on a Scheduling Problem

Example:

jobs	1	2	3	4
p_j	10	10	13	4
d_j	4	2	1	12
w_j	14	12	1	12

$1 \mid d_j \mid \sum w_j T_j$

Schedule four jobs on a machine

$T_j = \max(C_j - d_j, 0)$
tardiness of job j

Neighbourhood operator: go through all schedules that can be obtained through adjacent pairwise interchanges, choose the best.
Tabu-list: pairs of jobs (j, k) that were swapped within the last two moves
<u>Run the algorithm on the following slide for 2 while loop iterations, starting with the initial solution</u>: S_0 = <2, 1, 4, 3>

Applying Tabu Search Algorithm to a Scheduling Problem – Example

```
l=0; s_l ← initialize, maxTabuSize = 2;
S_best ← s0
tabuList ← []
while (not stoppingCondition())
    candidateList ← []
    bestCandidate ← null
    for (sCandidate in sNeighborhood) // any configuration in the neighbourhood of s: sCandidate ∈ N(s_l)
        if ( (not tabuList.contains(sCandidate)) and ( F(sCandidate) < F(bestCandidate) ) )
            bestCandidate ← sCandidate; bestMove ← (i, j) // swapped adjacent jobs
        end
    end
    l++; s_l ← bestCandidate
    if ( F(bestCandidate) < F(S_best) )
         S_best ← bestCandidate
    end
    tabuList.push( reverse(bestMove) ); // save (j, i) into tabu list
    if (tabuList.size > maxTabuSize)
        tabuList.removeFirst()
    end
end
return S_best
```

Q&A

Thank you.

Ender Özcan

ender.ozcan@nottingham.ac.uk

University of Nottingham, School of Computer Science
Jubilee Campus, Wollaton Road, Nottingham
NG8 1BB, UK
http://www.cs.nott.ac.uk/~pszeo