University of Nottingham
UK | CHINA | MALAYSIA

Computational Optimisation & Learning Lab

COM2001/2011 Artificial Intelligence Methods

Lecture 3 Metaheuristics

Prof Ender Özcan
Computer Science, Office: C86
Ender.Ozcan@nottingham.ac.uk
www.nottingham.ac.uk/~pszeo/

Computational Optimisation & Learning Lab

1. Performance Comparison of Stochastic Search Algorithms

Which Stochastic Search Algorithm Performs Better for Solving Problem X?

- Algorithm A is new, B & C are previous approaches applied to the instance Inst1 of the minimising problem X
- Assume all algorithms are run for the same number of objective function evaluations, and experiments are fair
- Each experiment is repeated for 30 times, that is, an algorithm is run for 30 times, independently, on an instance

run/trial	A	B	C
1	1	8	20
2	1	6	11
3	0	2	15
4	1	1	12
5	0	3	11
6	2	4	13
7	0	8	3
8	1	6	19
9	1	1	15
10	0	8	3
11	1	8	11
12	2	11	9
13	1	3	14
14	2	11	12
15	1	5	6
⋮			

	Algorithm A				Algorithm B				Algorithm C			
Instance	avg.	std.	med.	min.	avg.	std.	med.	min.	avg.	std.	med.	min.
Inst1	0.9	0.7	1	0	5.7	3.3	6	1	11.6	4.9	12	3

So, which algorithm performs the best for solving Problem X?

- **avg.**: mean objective value computed by averaging the objective values of 30 solutions returned by an algorithm from 30 independent trials/runs
- **std.**: standard deviation associated with avg.
- **med.**: median objective value
- **min.**: objective value of the best solution found in all trials/runs

Which Stochastic Search Algorithm Performs Better for Solving Problem X?

- Algorithm A is new, B & C are previous approaches applied to the instance Inst1 of the minimising problem X
- Assume all algorithms are run for the same number of objective function evaluations, and experiments are fair
- Each experiment is repeated for 30 times, that is, an algorithm is run for 30 times, independently, on an instance

run/trial	A	B	C
1	1	8	20
2	1	6	11
3	0	2	15
4	1	1	12
5	0	3	11
6	2	4	13
7	0	8	3
8	1	6	19
9	1	1	15
10	0	8	3
11	1	8	11
12	2	11	9
13	1	3	14
14	2	11	12
15	1	5	6

⋮

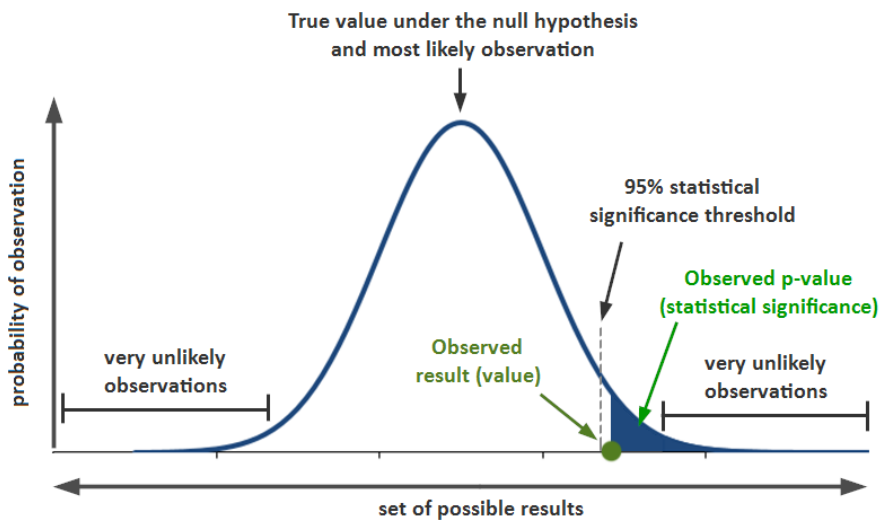
	Algorithm A				Algorithm B				Algorithm C			
Instance	avg.	std.	med.	min.	avg.	std.	med.	min.	avg.	std.	med.	min.
Inst1	0.9	0.7	1	0	5.7	3.3	6	1	11.6	4.9	12	3

So, which algorithm performs the best for solving Problem X?

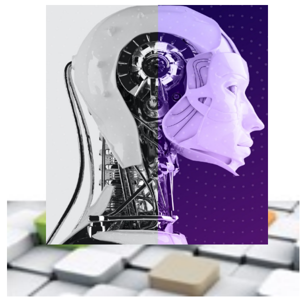
Any comment for 1 instance is valid for 1 problem instance. So, Algorithm A performs the best for the instance with the label Inst1 of the problem X based on mean, median, best (minimum) objective values.

Statistical tests, null hypothesis and p-values

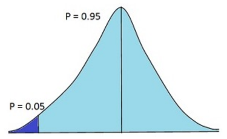
- The null hypothesis states the results are due to chance and are not significant in terms of supporting the idea being investigated.
- A p-value, or probability value, is a number describing how likely it is that your data would have occurred by random chance (i.e. that the null hypothesis is true).


True value under the null hypothesis and most likely observation
95% statistical significance threshold
Observed p-value (statistical significance)
Observed result (value)
very unlikely observations
very unlikely observations
probability of observation
set of possible results

Which Stochastic Search Algorithm Performs Better for Solving Problem X?



- Apply non-parametric statistical test – one tailed:
  - E.g.: Given two algorithms; X versus Y, > (<) denotes that X (Y) is better than Y (X) and this performance difference is statistically significant within a confidence interval of 95% and X ≥ Y (X ≤ Y) indicates that X (Y) performs better on average than Y (X) but no statistical significance (Wilcoxon signed rank test – e.g., http://vassarstats.net/wilcoxon.html)



	Algorithm A					Algorithm B					Algorithm C			
Instance	avg.	std.	med.	min.	vs.	avg.	std.	med.	min.	vs.	avg.	std.	med.	min.
Inst1	0.9	0.7	1	0	>	5.7	3.3	6	1	>	11.6	4.9	12	3

run/trial	A	B	C
1	1	8	20
2	1	6	11
3	0	2	15
4	1	1	12
5	0	3	11
6	2	4	13
7	0	8	3
8	1	6	19
9	1	1	15
10	0	8	3
11	1	8	11
12	2	11	9
13	1	3	14
14	2	11	12
15	1	5	6
		⋮	

- **A stronger conclusion can be provided for one instance (Inst1)**
- **Important**: Always repeat the experiments more than or equal to 30 times for any given instance for a meaningful statistical comparison

Statistical Tests

Nonparametric statistical tests

Type of comparison	Procedures
Pairwise comparisons	Sign test Wilcoxon test
Multiple comparisons $(1 \times N)$	Multiple sign test Friedman test Friedman Aligned ranks Quade test Contrast Estimation
Multiple comparisons $(N \times N)$	Friedman test

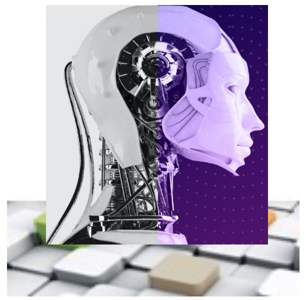
Post-hoc procedures

Type of comparison	Procedures
Multiple comparisons $(1 \times N)$	Bonferroni Holm Hochberg Hommel Holland Rom Finner Li
Multiple comparisons $(N \times N)$	Nemenyi Holm Shaffer Bergmann

Nonparametric statistics **are not based on any assumptions**, such as the sample following a specific distribution, and measures the central tendency with the median value

J. Derrac, S. García, D. Molina, F. Herrera, A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms, Swarm and Evolutionary Computation, vol. 1, issue 1, pp. 3-18, 2011. [PDF]

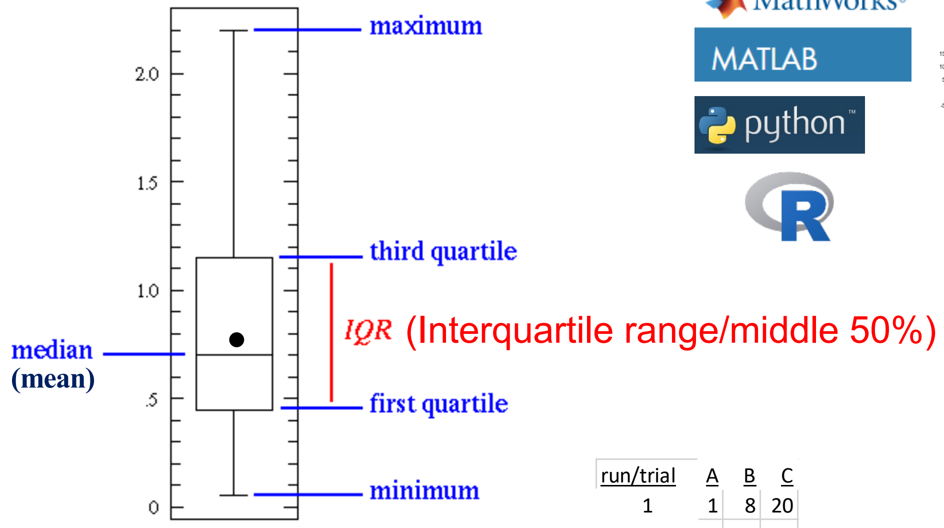
Which Stochastic Search Algorithm Performs Better for Solving Problem X? Boxplots



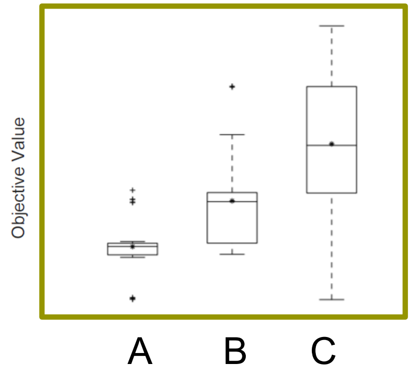
- Boxplots illustrates groups of numerical data through their quartiles.

  BoxPlotR: http://shiny.chemgrid.org/boxplotr/

- Example: boxplot of objective values obtained from 30 runs on the instance Inst1 from 3 algorithms
  - Outliers may be plotted as individual points.
  - **A stronger conclusion for Inst1 – next slide**



MathWorks MATLAB python gnuplot R X

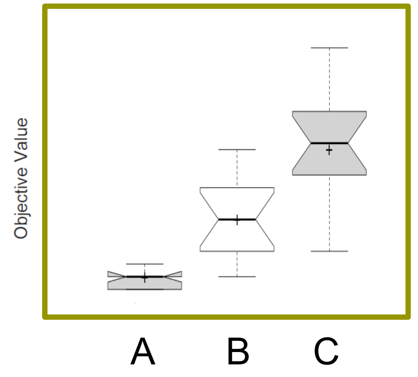


run/trial	A	B	C
1	1	8	20
2	1	6	11
3	0	2	15
4	1	1	12
5	0	3	11
6	2	4	13
7	0	8	3
8	1	6	19
9	1	1	15
10	0	8	3
11	1	8	11
12	2	11	9
13	1	3	14
14	2	11	12
15	1	5	6

⋮

Which Stochastic Search Algorithm Performs Better for Solving Problem X? <u>Notched Boxplots</u>

- The notched boxplot allows you to evaluate confidence intervals (by default 95% confidence interval) for the medians of each boxplot.



- Since the notches in the boxplots A vs B, A vs C and B vs C do not overlap, you can conclude that with 95% confidence, that the true medians do differ between each pair of those algorithms on Inst1:

  A performs significantly better than B as well as C, and B performs significantly better than C.

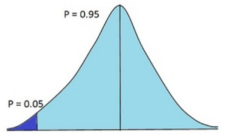
Which Stochastic Search Algorithm Performs Better for Solving Problem X?

- Algorithms are run on multiple instances from Problem X – now even a stronger conclusions can be made, e.g.,:
  - Algorithm A performs the best on (this benchmark/dataset) all problem X instances based on the mean/median objective values achieved over 30 trials.
  - Algorithm C is the worst performing approach based on all metrics.
  - Algorithm B provides the same best solution for 3 instances as Algorithm A.

	Algorithm A				Algorithm B				Algorithm C			
Instance	avg.	std.	med.	min.	avg.	std.	med.	min.	avg.	std.	med.	min.
Inst1	0.9	0.7	1	0	5.7	3.3	6	1	11.6	4.9	12	3
Inst2	3.1	3.9	2	1	21.3	13	12	3	44.9	9.8	30	18
Inst3	0.7	0.5	1	0	7.1	7.7	3	0	26.3	14	13	1
Inst4	1.7	1	1	1	5.7	4.3	3	1	20	4.6	15	12
Inst5	7.6	0.9	7	7	10.4	1.5	8	7	15.4	1.7	14	13

Which Stochastic Search Algorithm Performs Better for Solving Problem X?

- Apply non-parametric statistical test – one tailed:
  - E.g.: Given two algorithms; X versus Y, > (<) denotes that X (Y) is better than Y (X) and this performance difference is statistically significant within a confidence interval of 95% and X ≥ Y (X ≤ Y) indicates that X (Y) performs better on average than Y (X) but no statistical significance (Wilcoxon signed rank test – e.g., http://vassarstats.net/wilcoxon.html)



	Algorithm A					Algorithm B					Algorithm C			
Instance	avg.	std.	med.	min.	vs.	avg.	std.	med.	min.	vs.	avg.	std.	med.	min.
Inst1	0.9	0.7	1	0	>	5.7	3.3	6	1	>	11.6	4.9	12	3
Inst2	3.1	3.9	2	1	>	21.3	13	12	3	>	44.9	9.8	30	18
Inst3	0.7	0.5	1	0	>	7.1	7.7	3	0	>	26.3	14	13	1
Inst4	1.7	1	1	1	>	5.7	4.3	3	1	>	20	4.6	15	12
Inst5	7.6	0.9	7	7	>	10.4	1.5	8	7	>	15.4	1.7	14	13

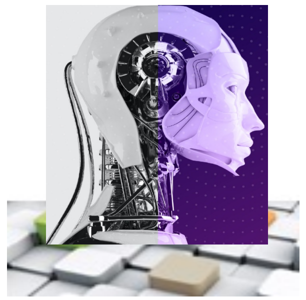
- **Important**: Always repeat the experiments more than or equal to 30 times for all instances for a meaningful statistical comparison

Which Stochastic Search Algorithm Performs Better for Solving Problem X?

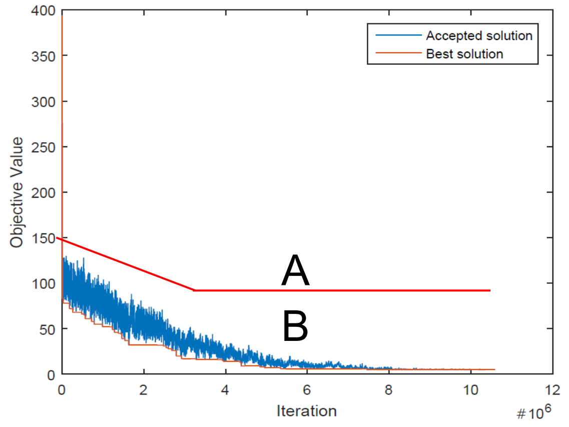
- Algorithms are run on multiple instances of Problem X –conclusions can be strengthen further, e.g.,:
  - Algorithm A is better than Algorithm B, while Algorithm B is better than C on (this benchmark/dataset or) all problem X instances on average and all those performance differences between the pair of algorithms are statistically significant within a confidence interval of 95%.

	Algorithm A					Algorithm B					Algorithm C			
Instance	avg.	std.	med.	min.	vs.	avg.	std.	med.	min.	vs.	avg.	std.	med.	min.
Inst1	0.9	0.7	1	0	>	5.7	3.3	6	1	>	11.6	4.9	12	3
Inst2	3.1	3.9	2	1	>	21.3	13	12	3	>	44.9	9.8	30	18
Inst3	0.7	0.5	1	0	>	7.1	7.7	3	0	>	26.3	14	13	1
Inst4	1.7	1	1	1	>	5.7	4.3	3	1	>	20	4.6	15	12
Inst5	7.6	0.9	7	7	>	10.4	1.5	8	7	>	15.4	1.7	14	13

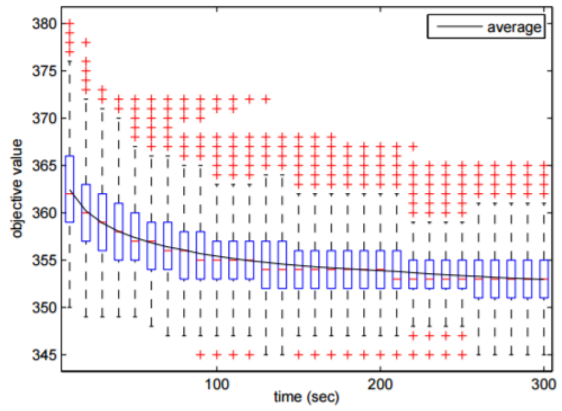
Performance Analysis Using Plots – Other Methods



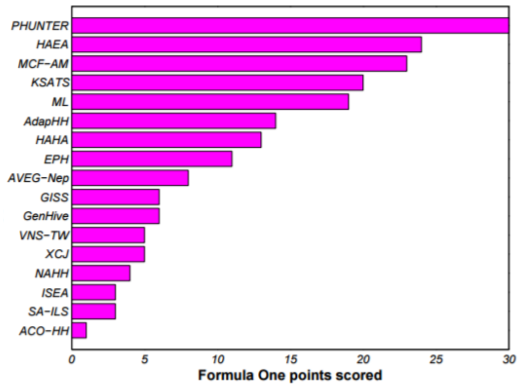
- **Progress plot** – per instance: Objective value from a run or mean of objective values from multiple runs per iteration/time unit
- Ranking



Objective value of best and accepted solutions versus iteration from Algorithms A and B for the same instance from a sample run



Box plot of objective values collected at each time step (every 10 sec.) from multiple runs of the Algorithm A on an instance



(higher score is better)

$$f_{norm}(s) = \frac{f(s) - f(s_{best})}{f(s_{worst}) - f(s_{best})}$$

	Cross-domain
IE	17.67
AILLA	15.91
TA	20.04
GD	13.15
AILTA	19.05
NA	29.61
SA	**10.10**
SARH	10.22

(lower score is better)

Formula 1 vs normalised objective value ranking of algorithms on multiple instances

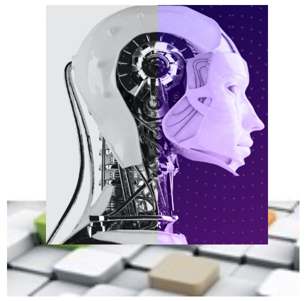
2. Metaheuristics

What is a Metaheuristic?

A **metaheuristic** is a high-level problem independent algorithmic framework that provides a set of guidelines or strategies to develop heuristic optimization algorithms

K. Sörensen and F. Glover. Metaheuristics. In S.I. Gass and M. Fu, editors, Encyclopedia of Operations Research and Management Science, pp 960–970. Springer, New York, 2013. [PDF]

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)

Main Components of a Metaheuristic Search Method (r.v.)

- ☑ Representation of candidate solutions
- ☑ Evaluation function
- ☑ Initialisation: E.g., initial candidate solution may be chosen
  - randomly
  - use a constructive heuristic
  - according to some regular pattern
  - based on other information (e.g. results of a prior search), and more
- ☑ Neighbourhood relation (move operators)
- ● Search process (guideline)
- ● Stopping conditions
- ● *Mechanism for escaping from local optima*

Mechanisms for Escaping from Local Optima I

- Search process (guideline)
- Stopping conditions
- *Mechanism for escaping from local optima*

- Iterate with different solutions, or restart (re-initialise search whenever a local optimum is encountered).
  - Initialisation could be costly
  - E.g., Iterated Local Search, GRASP
- Change the search landscape
  - Change the objective function (E.g., Guided Local Search)
  - Use (mix) different neighbourhoods (E.g., Variable Neighbourhood Search, Hyper-heuristics)

Mechanisms for Escaping from Local Optima II

- Search process (guideline)
- Stopping conditions
- *Mechanism for escaping from local optima*

- Use Memory (e.g., tabu search)
- Accept non-improving moves: allow search using candidate solutions with equal or worse evaluation function value than the one in hand
  - Could lead to long walks on plateaus (neutral regions) during the search process, potentially causing cycles – visiting of the same states
- None of the mechanisms is guaranteed to always escape effectively from local optima

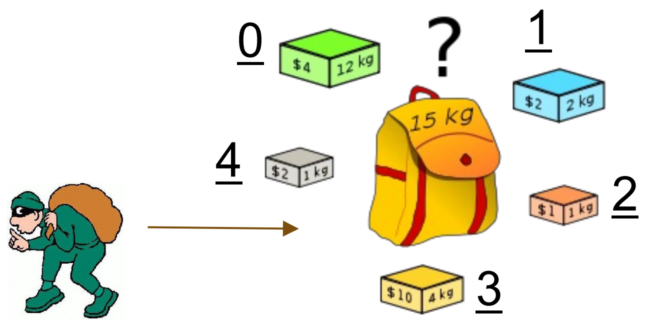
Termination Criteria (Stopping Conditions) – Examples

- Search process (guideline)
- Stopping conditions
- *Mechanism for escaping from local optima*

- Stop if
  - a fixed maximum number of iterations, or moves, objective function evaluations), or a fixed amount of CPU time is exceeded.
  - consecutive number of iterations since the last improvement in the best objective function value is larger than a specified number.
  - evidence can be given than an optimum solution has been obtained. (i.e. optimum objective value is known)
  - no feasible solution can be obtained for a fixed number of steps/time. (a solution is *feasible* if it satisfies all constraints in an optimisation problem)

Feasibility Example: 0/1 Knapsack Problem

- Fill the knapsack with as much value in goods as possible (i.e., maximise "profit") **without exceeding the capacity** (as a constraint) – which items to take?



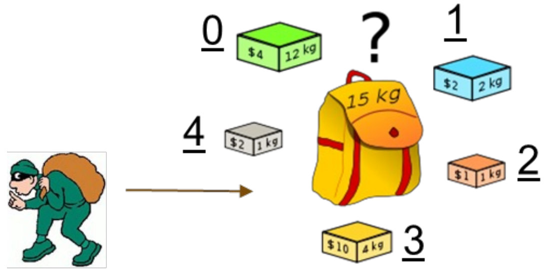
$f(s)$

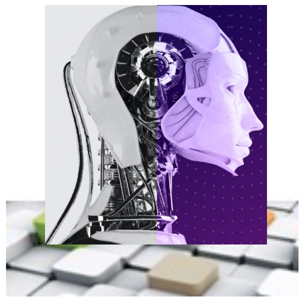
0 1 2 3 4 profit

- 1 1 0 0 1: $7 (15 kg) $\longrightarrow$ feasible solution
- 0 1 1 1 1: $15 (8 kg) $\longrightarrow$ feasible solution
- 1 0 0 1 0: ~~$14~~ (**16 kg**) $\longrightarrow$ not a feasible solution
- 1 1 0 1 0: ~~$16~~ (**18 kg**) $\longrightarrow$ not a feasible solution

How to Deal with Infeasible Solutions

maximisation problem





p_i is the profit from the ith item

- Use a problem domain specific repair operator
  - E.g. randomly flip a bit to 0 until the solution in hand feasible: 1 **1** 0 1 0: $16 (**18 kg**) → 1 **0** 0 $\underline{1}$ 0: ~~$14~~ (**16 kg**) → 1 0 0 $\underline{0}$ 0 $4 (12 kg)
- Penalise **each constraint violation** for the infeasible solutions such that they can't be better than the worst feasible solution for a given instance
  - Set a fixed (death) penalty value poorer than the worst, e.g.,

    $f'(s)$=if s is infeasible, then min$\{p_i, \forall i\}$/2 (that is $1 for the example)

    1 1 0 1 0: $0.5 (**18 kg**), 1 0 0 1 0: $0.5 (**16 kg**)
  - Distinguish the level of infeasibility of a solution with the penalty: e.g.,

    $f'(s)$= if s is infeasible, then min$\{p_i, \forall i\}$/(2*($total\_weight$-**capacity**))

    1 1 0 1 0: $0.167 (**18 kg**), 1 0 0 1 0: $0.5 (**16 kg**)

3. Local Search Metaheuristics and Iterated Local Search

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

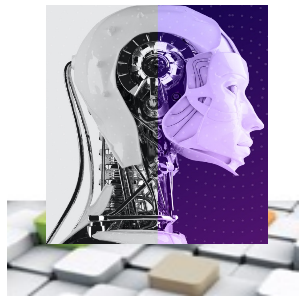
```
$s_0$ ;  // starting solution
s* = initialise($s_0$); // e.g., improve $s_0$ or use the same
repeat
   // generate a new solution
    s'  = makeMove(s*, memory); // choose a neighbour of s*
    **accept = moveAcceptance(s*, s', memory);**  // remember $s_{best}$
    **if  (accept) s* = s'; // else reject new solution s'**
until (termination conditions are satisfied);
```

- 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

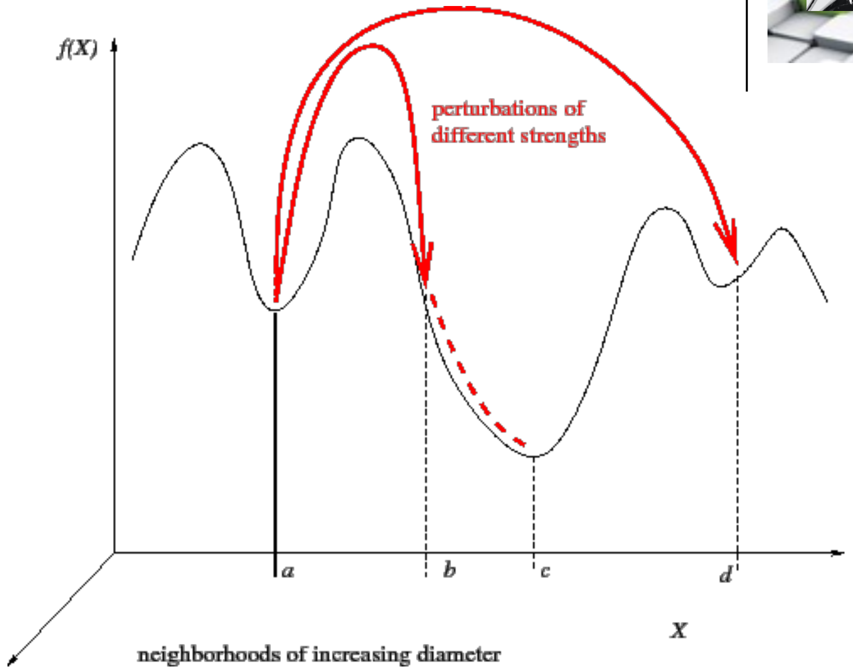
The Art of Searching

- Effective search techniques provide a mechanism to balance *exploration* and *exploitation*
  - Exploration aims to prevent stagnation of search process getting trapped at a local optimum
  - Exploitation aims to greedily increase solution quality or probability, e.g., by exploiting the evaluation function
- Aim is to design search algorithms/metaheuristics that can
  - escape local optima
  - balance exploration and exploitation
  - make the search independent from the initial configuration

Iterated Local Search (ILS)



```
$s_0$= GenerateInitialSolution()
//random or construction heuristic
**s* = LocalSearch($s_0$)** // hill climbing - not always used
Repeat
    s' = **Perturbation(s*, memory)**
    **// random move**
    s' = **LocalSearch(s' )**
    **// hill climbing**
    s* = **AcceptanceCriterion(s*, s', memory)** // remember $s_{best}$
    // the conditions that the new local optimum
    // must satisfy to replace the current solution
Until (termination conditions are satisfied)
return s*
```



f(X)
perturbations of different strengths
a
b
c
d
X
neighborhoods of increasing diameter

Iterated Local Search II

- Based on visiting a sequence of locally optimal solutions by:
  - perturbing the current local optimum (exploration);
  - applying local search/hill climbing (exploitation) after starting from the modified solution.
- A perturbation phase might consist of one or more steps
- The perturbation strength is crucial
  - Too small (weak): may generate cycles
  - Too big (strong): good properties of the local optima are lost.

Iterated Local Search III

- Acceptance criteria
  - Extreme in terms of exploitation: accept only improving solutions
  - Extreme in terms of exploration: accept any solution
  - Other: deterministic (like threshold), probabilistic (like Simulated Annealing)
- Memory
  - Very simple use: restart search if for a number of iterations no improved solution is found

Exercise 1a – Designing an ILS Algorithm for MAX-SAT

- GenerateInitialSolution: Random
- Perturbation: A number of random bit flips (random walk)
- LocalSearch: Use 1-bit-flip neighbourhood, Steepest Descent Hill Climbing
- AcceptanceCriterion: accept *improving and equal moves* (non-worsening): accept *s' if and only if* $f(s') \leq f(s^*)$

Applying an Iterated Local Search Algorithm to a MAX-SAT Problem Instance – Exercise

- Problem:
  - $(a \lor b) \land (\neg d \lor f) \land (\neg a \lor c) \land (b \lor \neg f) \land (\neg b \lor c) \land (c \lor e)$
- Representation: Binary string
- Initialisation: random binary string
- Objective function: number of unsatisfied clauses

- Step 1: Initialise Solution
  - $s_0 \leftarrow \overset{a}{1}\overset{b}{0}\overset{c}{0}\overset{d}{1}\overset{e}{0}\overset{f}{0}$
  - $(a \lor b) \land (\neg d \lor f) \land (\neg a \lor c) \land (b \lor \neg f) \land (\neg b \lor c) \land (c \lor e)$
  - $c_0 c_1 c_2 c_3 c_4 c_5$
  - $f(s_0) = 3$
  - (No local search after initialisation)

$s^* = s_0 = 100100; f(s_0) = 3$

- Problem:
  - $(a \vee b) \wedge (\neg d \vee f) \wedge (\neg a \vee c) \wedge (b \vee \neg f) \wedge (\neg b \vee c) \wedge (c \vee e)$
- Representation: Binary string
- Initialisation: random binary string
- Objective function: number of unsatisfied clauses
- Perturbation Operator: Random bit flip
- Local Search Operator: Steepest (Gradient) Descent with IE (≤) acceptance (SDHC)
- Move Acceptance: Improving or equal (≤)

- Step 2: Iterated Local Search (Loop 1)
  - Random Bit Flip
    - $s' \leftarrow 110100$
    - Clause SAT: 012345
    - $f(s') = 4$
  - SDHC
    - $f(\overset{a\,b\,c\,d\,e\,f}{010100}) = 012345 = 3$
    - $f(100100) = 012345 = 3$
    - $\underline{f(111100) = 012345 = 1}$
    - $f(110000) = 012345 = 3$
    - $f(110110) = 012345 = 3$
    - $f(110101) = 012345 = 3$
    - $s' \leftarrow 111100$
  - $1 < 3\ \left(f(s') \leq f(s^*)\right)$
    - $\therefore s^* \leftarrow s'$

$s^* = s_1 = 111100; f(s_1) = 1$

- Problem:
  - $(a \vee b) \wedge (\neg d \vee f) \wedge (\neg a \vee c) \wedge (b \vee \neg f) \wedge (\neg b \vee c) \wedge (c \vee e)$

- Step 2: Iterated Local Search (Loop 2)
  - Random Bit Flip
    - $s' \leftarrow 101100$
    - Clause SAT: 012345
    - $f(s') = 1$
  - SDHC
    - a b c d e f
    - $f(001100) = 012345 = 2$
    - $f(111100) = 012345 = 1$
    - $f(100100) = 012345 = 3$
    - $\underline{f(101000) = 012345 = 0}$
    - $f(101110) = 012345 = 1$
    - $f(101101) = 012345 = 1$
    - $s' \leftarrow 101000$
  - $0 < 1 \; \big(f(s') \leq f(s^*)\big)$
    - $\therefore s^* \leftarrow s'$
  - return $f(s^*) = 0;$

Example 1b – Designing Another ILS Algorithm for MAX-SAT

- GenerateInitialSolution: Random
- Perturbation: *intensityOfMutation* times random bit flips
- LocalSearch: Davis's Bit Hill Climbing for *depthOfSearch* (noOfSteps) times
- AcceptanceCriterion: accept *improving and equal moves* (non-worsening): accept *s' if and only if* $f(s') \leq f(s^*)$

Example 2 – Designing an ILS Algorithm for TSP

- GenerateInitialSolution: Nearest-neighbor
- Perturbation: a number of exchange moves

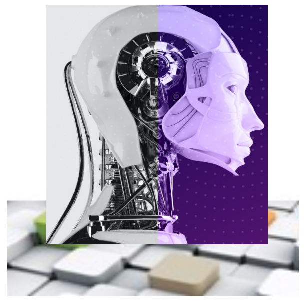
  E.g.:1 2 3 4 5 6 → 1 4 3 2 5 6, then 6 4 3 2 5 1
- LocalSearch: first improvement using insertion neighborhood moves

  E.g.: 1 2 3 4 5 6 → 1 3 2 4 5 6, 1 2 3 4 5 6, 1 3 4 2 5 6, 1 3 4 5 2 6, 1 3 4 5 6 2, 2 1 3 4 5 6
- AcceptanceCriterion: accept *improving moves only*: accept *s' if and only if* $f(s') < f(s^*)$

ILS – Some Guidelines I

- Initial solution should be to a large extent irrelevant for longer runs.
- The interactions among perturbation strength and acceptance criterion can be particularly important
  - it determines the relative balance of exploration and exploitation
  - large perturbations are only useful if they can be accepted.

ILS – Some Guidelines II



- Advanced acceptance criteria may take into account search history,e.g., by occasionally reverting to incumbent solution.
- Advanced ILS algorithms may change nature and/or **strength of perturbation** (e.g., number of bit flips) adaptively during search.
- Local search should be as effective and as fast as possible. Better local search generally leads to better ILS performance
- Choose a perturbation operator whose steps cannot be easily undone by the local search

Computational Optimisation & Learning Lab

4. Tabu Search

Tabu Search

- Proposed by Fred W. Glover in 1986 and formalised in 1989
- Uses history (memory structures) to escape from local minima, inspired by ideas from artificial intelligence in the late 1970s.
- Applies hill climbing/local search
  - Some solution elements/moves are regarded as tabu (forbidden)
  - Proceeds according to the assumption that there is no point in accepting a new (poor) solution unless it is to avoid a path already investigated

Glover F 1986 Future Paths for Integer Programming and Links to Artificial Intelligence. Computers and Operations Research. Vol. 13, pp. 533-549. [PDF]

Tabu Search Algorithm

Tabu Search (TS):

```
determine initial candidate solution $s$
While *termination criterion* is not satisfied:
    determine set $N'$ of non-tabu neighbours of $s$
    choose a best improving candidate solution $s'$ in $N'$

    update tabu attributes based on $s'$
    $s := s'$
```

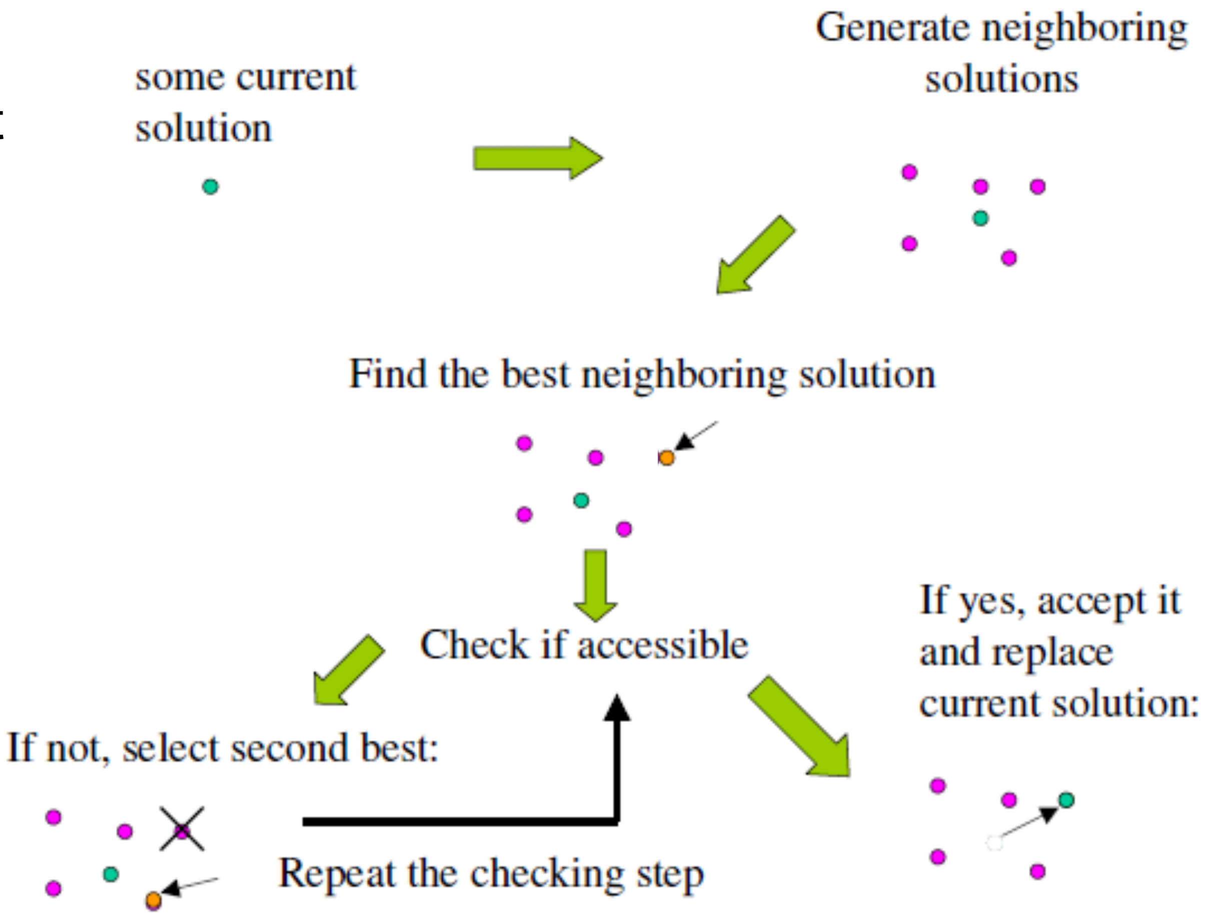
Tabu Search – Overview

3 main components:

– Forbidding strategy: control what enters the tabu list

– Freeing strategy: control what exits the tabu list and when

– Short-term strategy: manage interplay between the forbidding strategy and freeing strategy to select trial solutions

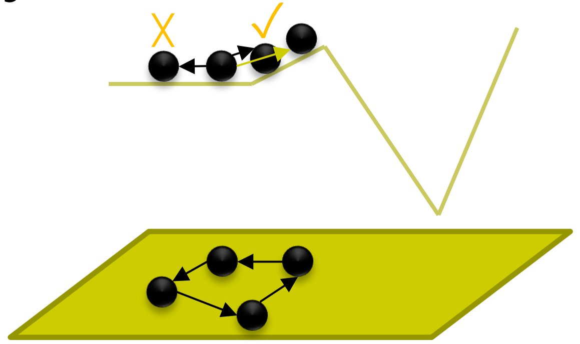

some current solution
Generate neighboring solutions
Find the best neighboring solution
Check if accessible
If not, select second best:
If yes, accept it and replace current solution:
Repeat the checking step

Tabu Search – Fundementals I

- A stochastic local search algorithm which heavily relies on the use of an explicit memory of the search process
  - systematic use of memory to guide search process
  - memory typically contains only specific attributes of previously seen solutions
  - simple tabu search strategies exploit only short term memory
  - more complex tabu search strategies exploit long term memory

Tabu Search – Fundementals II

- In each step, move to ‘non-tabu’ best neighbouring solution (*admissible neighbours*) although it may be worse than current one
- To avoid cycles, tabu search tries to avoid revisiting previously seen solutions
- Avoid storing complete solutions by basing the memory on attributes of recently seen solutions



Tabu Search – Fundementals III

- Tabu solution attributes are often defined via local search moves
- *Tabu-list* contains moves which have been made in the recent past
  - *tabu list length* or *tabu tenure*
- Solutions which contain tabu attributes are forbidden for a certain number of iterations.
- Often, an additional *aspiration criterion* is used: this specifies conditions under which tabu status may be overridden (e.g., if considered step leads to improvement in incumbent solution).

Designing a Tabu Search Algorithm for MAX-SAT

- <u>Initialisation</u>: random – select an assignment randomly from set of all truth assignments
- <u>Neighbourhood</u>: 1-bit flip neighbourhood
- <u>Memory</u>: Associate tabu status (Boolean value) with each variable in given CNF.
  - variables are tabu iff they have been changed in the last T *steps*
    - for each variable x_i store the search step when its value was last changed denoted as t_i; x_i is tabu iff $ci - t_i \leq T$, where ci is the current iteration/search step number.
    - use a queue of length T indicating the bit that was flipped in which step

An iteration of Tabu Search for MAX-SAT

$s^* = 101100; f(s^*) = 1$

- Problem:
  - $(a \vee b) \wedge (\neg d \vee f) \wedge (\neg a \vee c) \wedge (b \vee \neg f) \wedge (\neg b \vee c) \wedge (c \vee e)$
- Neighbourhood: 1-bit flip
- Assume *T* = 2
- Current iteration (*ci*) is 1207
- Current list content: [1205,59,1206,11,0,928]
  - This means that 1st bit/variable (a) was flipped in the 1205th step, second bit/variable (b) was flipped in the 59th step, and so on…

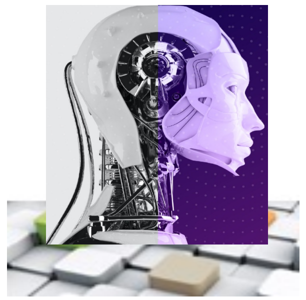
- Consider all 1-bit flip neighbours which are not in the tabu list
  - (001100) **X** $tabu$ $(1207\text{-}1205) \leq 2$
  - $f(111100) = 012345 = 1$
  - (100100) **X** $tabu$ $(1207\text{-}1206) \leq 2$
  - $\underline{f(101000) = 012345 = 0}$
  - $f(101110) = 012345 = 1$
  - $f(101101) = 012345 = 1$
  - $s' \leftarrow 101000$
- Update list: [1205,59,1206,1207,0,928]

An iteration of Tabu Search for MAX-SAT

$s^* = 101100; f(s^*) = 1$

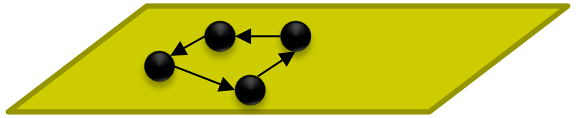
- Problem:
  - $(a \vee b) \wedge (\neg d \vee f) \wedge (\neg a \vee c) \wedge (b \vee \neg f) \wedge (\neg b \vee c) \wedge (c \vee e)$
- Neighbourhood: 1-bit flip
- Tabu tenure: 2
- Current iteration (*ci*) is 1207
- Current tabu list content: <1,3> (tail)
  - This means that 1st bit/variable (a) was flipped two steps ago (i.e., at step 1205), while 3rd bit/variable (c) was flipped in the immediately previous step (i.e., at step 1206).

- Consider all 1-bit flip neighbours which are not in the tabu list
  - $(001100)\ \mathbf{X}\ tabu$
  - $f(111100) = 012345 = 1$
  - $(100100)\ \mathbf{X}\ tabu$
  - $\underline{f(101000) = 012345 = 0}$
  - $f(101110) = 012345 = 1$
  - $f(101101) = 012345 = 1$
  - $s' \leftarrow 101000$
- Update tabu list: <3,4> (tail)



Practical Considerations

- Appropriate choice of tabu tenure critical for performance
- Tabu tenure: the length of time/number of steps t for which a move is forbidden
  - t too low- risk of cycling
  - t too high - may restrict the search too much
  - t =7 has often been found sufficient to prevent cycling
  - $t = \sqrt{n}$
  - number of tabu moves: 5 – 9
- If a tabu move is smaller than the aspiration level then we accept the move (use of aspiration criteria to override tabu status)



Computational Optimisation & Learning Lab

5. Introduction to Scheduling

Scheduling

- Scheduling deals with the allocation of resources to tasks over given time periods and its goal is to optimize one or more objectives. The resources and tasks in an organization can take many different forms.
- It is a decision-making process that is used on a regular basis in many manufacturing and services industries. Many real-world applications including
  - Project planning
  - Semiconductor manufacturing
  - Gate assignment at airports
  - Scheduling tasks in CPUs/parallel computers, …

Scheduling – Framework and Notation

- In all the scheduling problems, considered number of jobs and machines are assumed to be finite.

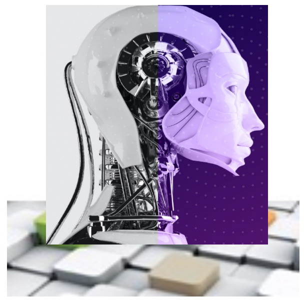
<u>jobs</u> $j=1,\ldots,n$ **<u>machines</u>** $i=1,\ldots,m$

(i, j) processing step, or operation of job j on machine I

Classification of Scheduling Problems – Notation

- **<u>Scheduling problem</u>**: $\alpha \mid \beta \mid \gamma$
  - α machine characteristics (environments)
  - β processing/job characteristics
  - γ optimality criteria (objective to be minimised)

Sample Machine Characteristics (α)



$\alpha \mid \beta \mid \gamma$

Single-stage problem **(vs. Multi-stage problem: O, F, J)**

1 **Single machine**

Pm **Identical machines in parallel**

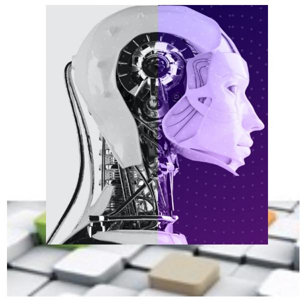
- *m* machines in parallel
- Job *j* requires a single operation and may be processed on any of the *m* machines

Qm **Machines in parallel with different speeds**

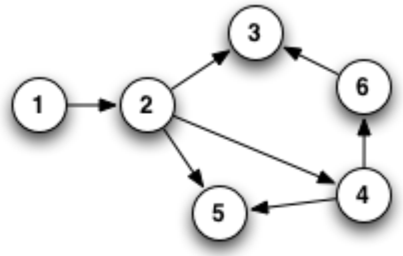
Rm **Unrelated machines in parallel** machines have different speeds for different jobs

Sample Job Characteristics (β)

- The processing time may be different, or equal for all jobs, or even of unit length. All processing times are assumed to be integers.
- **Processing time** p_{ij} - processing time of job *j* on machine *i* (if a single machine then p_j)
- **Due date** d_j - committed shipping or completion (due) date of job *j*
- **Weight** w_j - importance of job *j* relative to the other jobs in the system



Sample Job Characteristics (β) II



- **Release date** r_j - earliest time at which ɔ *j* can start its processing
- **Precedence** *prec* – Precedence relations might be given for the jobs. If *k* precedes *l*, then starting time of *l* should be not earlier than completion time of *k*.
- **Sequence dependent setup times** s_{jk} - setup time between jobs *j* and *k*
- **Breakdowns** *brkdwn* - machines are not continuously available

Sample Optimality Criteria (γ)

We define for each job *j*:

- C_{ij} **completion time** of the operation of job *j* on machine *i*
- C_j **time** when job *j* exits the system
- $C_{\max}$ **makespan** is the time difference from the start (often, t=0) to finish when the last job exits the system
- $L_j = C_j - d_j$ **lateness** of job *j*
- $T_j = \max(C_j - d_j, 0)$ **tardiness** of job *j*
- $U_j = \begin{cases} 1 & \text{if } C_j > d_j \\ 0 & \text{otherwise} \end{cases}$ **unit penalty** of job *j*

See Lecture 4
Scheduling Notation – Examples

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

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

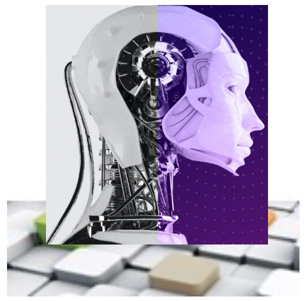
- $P3 \mid d_j, s_{jk} \mid \sum L_j$

  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.

See Lecture 4
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*

See Lecture 4

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

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

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

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

See Lecture 4
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 often dealt with in manufacturing/ production using heuristics/hyper-/metaheuristics.

Q&A

ender.ozcan@nottingham.ac.uk

www.cs.nott.ac.uk/~pszeo/