UNIVERSITY OF STIRLING
COMPUTING SCIENCE AND MATHEMATICS

Search-based Approaches and Hyper-heuristics

Gabriela Ochoa
http://www.cs.stir.ac.uk/~goc/
Computing Science and Mathematics,
School of Natural Sciences
University of Stirling, Stirling, Scotland

Outline

1. **Optimisation problems**
   – Optimisation & search
   – Classic mathematical models
   – Two canonical examples (Knapsack, TSP)
2. **Optimisation methods**
   – Heuristics and metaheuristcis
   – Single point algorithms
   – Population-based algorithms
3. **Autonomous search and hyper-heuristics**

Optimisation problems

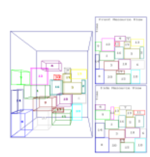
- Wide variety of applications across industry, commerce, science and government
- Optimisation occurs in the minimisation of time, cost and risk, or the maximisation of profit, quality, and efficiency
- Examples
  – Finding shortest round trips in graphs (TSP)
  – Finding models of propositional formulae (SAT)
  – Determining the 3D-structure of proteins
  – Planning, scheduling, cutting & packing, logistics, transportation, communications, timetabling, resource allocation, genome sequencing
  – *Software engineering*: test case minimisation and prioritisation, requirements analysis, code design and repair, etc.

Optimisation problems are everywhere!

Logistics, transportation, supply change management

Manufacturing, production lines

Timetabling



Cutting & packing

Computer networks and Telecommunications

Software - SBSE

Optimisation problems

General constrained optimisation problem:

Min/Max $f(\vec{x})$

Subject to:

$g_i(\vec{x}) \leq 0$ i = 1,…,p

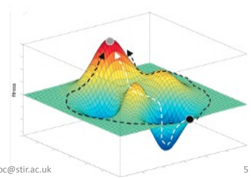
$h_j(\vec{x}) = 0$ j = 1,…,n

Search Space: set of candidate solutions. All possible combinations of the decision variables.

Optimisation through search

Iteratively generate and evaluate candidate solutions.

- Systematic search
- (Stochastic) local search



Search in Computing Science

At least 4 meanings of the word ***search*** in CS

1. Search for stored data • Finding information stored in disc or memory. • Examples: Sequential search, Binary search	**2. Search for web documents** • Finding information on the world wide web • Results are presented as a list of results
3. Search for paths or routes • Finding a set of actions that will bring us from an initial stat to a goal stat • Relevant to AI • Examples: depth first search, breath first search, branch and bound, A*, Monte Carlo tree search.	**4. Search for solutions** • Find a solution in a large space of candidate solutions • Relevant to AI, Optimisation, OR • Examples: evolutionary algorithms, Tabu search, simulated annealing, ant colony optimisation, etc.

Search and optimisation in practice

Many challenging applications in science and industry can be formulated as optimisation problems!

Real-world (SE) problem

→ **Formulation**

Model

→ **Algorithm**

Solution

Problem Model
- Problem representation
- Constraints
- A fitness function

Optimisation/search Algorithm
- Exact methods
- Approximate (heuristic) methods

Solution to the Model
- Feasible candidate solution
- Lead to the optimal (or good enough) value of the objective function

Optimisation problems: two categories

Continuous
- *Continuous* variables
- Looking for a set (vector) of real numbers [45.78, 8.91, 3.36]
- Objective function has a mathematical expression
- Special cases studied in mathematics and OR: *Convex, Linear, Dynamic programming*

Combinatorial
- *Discrete* variables
- Looking for an object from a finite set
  - Binary digits [1011101010]
  - Integer [1, 53, 4, 67, 39]
  - Permutation [3,5,1,2,4]
  - Graph

- Generally have quite different flavours and methods for solving them
- Have become divergent

Classic mathematical models

Linear Programs (LP)

- A single objective
- The objective and constraints are linear
- Decision variables, allowed to have *any* values
- Easy to solve numerically (*simplex* method)

Importance

- Many applications

Integer Programs (IP)

- LP in which some or all variables are constrained to take on *integer* values
- Harder to solve. Software packages: Excel, LINGO/LINDO and MPL/CPLEX,

Importance

- problems in which variables **required** to be integer
- many decisions are essentially discrete (yes/no, go/no-go)

Integer program: canonical form

maximise $c_1x_1+c_2x_2+\ldots+c_nx_n$ (objective function)

subject to

$a_{11}x_1+a_{12}x_2+\ldots+a_{1n}x_n \le b_1$ (functional constraints)

$a_{21}x_1+a_{22}x_2+\ldots+a_{2n}x_n \le b_2$

....

$a_{m1}x_1+a_{m2}x_2+\ldots+a_{mn}x_n \le b_m$

$x_1, x_2, \ldots, x_n \in \mathbf{Z}_+$ (set constraints)

In vector form:

maximise $\mathbf{cx}$ (objective function)

subject to $\mathbf{Ax} \le \mathbf{b}$ (functional constraints)

$\mathbf{x} \in \mathbf{Z}^n_+$ (set constraints)

The knapsack problem

- Given a knapsack of capacity *W,* and a number *n* of items, each with a *weight* and *value*. The objective is to maximise the total value of the items in the knapsack

Maximise $\sum_{i=1}^{n} v_i x_i$ Subject to $\sum_{i=1}^{n} w_i x_i \leqslant W, \quad x_i \in \{0,1\}$

- Search space size = 2^n
- $n = 100$, $2^{100} \approx 10^{30}$

maximise

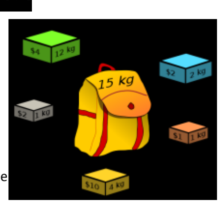
$4x_1+2x_2+x_3+10x_4+2x_5$

$x_i = \begin{cases} 1 & \text{If we select item } i \\ 0 & \text{Otherwise} \end{cases}$

subject to

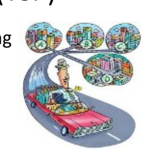
$12x_1+2x_2+x_3+4x_4+x_5 \le 15$

$x_1,x_2,x_3,x_4,x_5 \in \{0, 1\}$

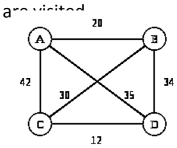


- Can be formulated as an Integer Programming problem and solved efficiently using Dynamic Programming
- Binary representation [11010], using heuristic methods

Travelling salesman problem (TSP)



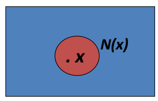
- Given a number of cities and the costs of travelling from one to the other, what is the cheapest roundtrip route that visits each city and then returns to the starting city?
- Objective: Min Sum(dist(x,y)). Total cost (distance) travelled
- Configurations: permutation (ordering) of cities. Representing the order in which cities are visited
  - *s1*= (A B C D), *f(s1)*= 20+30+12+35= 97
  - *s2*= (A B D C), *f(s2)*= 20+34+12+42=108
  - *s3*= (A C B D), *f(s3)*= 42+30+34+35= 141
- Size of the search space: (n-1)!/2
  - n= 10 (181,000); n=30 (10^{32})



Neighbourhoods

- Region of the search space that is "near" to some particular point in that space
- Define a distance function *dist* on the search space S
  - *Dist*: S x S → R
  - *N(x)* = {y ∈ S: *dist(x,y)* ≤ ε }

Examples:
- Euclidean distance, for search spaces defined over continuous variables
- Hamming distance, for search spaces definced over binary strings



A search space *S*, a potential solution *x*, and its neighbourhood *N(x)*

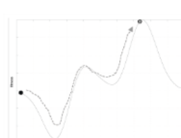
Defining neighbourhoods

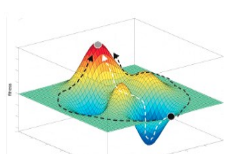
Binary
- 1-flip: Solutions generated by flipping a single bit in the given bit string
- Every solution has *n* neighbours
- Example:
  - 1 1 0 0 1 → 0 1 0 0 1

Permutation
- *2-swap*: Solutions generated by swapping two cities from a given tour
- Every solution has *n(n-1)/2* neighbours
- Example:
  - 2 4 5 3 1 → 2 3 5 4 1,

Fitness landscapes

- Describe dynamics of adaptation in Nature (Wright, 1932). Later, describe dynamics of meta-heuristics
- Search: adaptive-walk over a Landscape
- 3 Components *L = (S,d,f)*
  - Search Space
  - Neighborhood relation or distance metric (operator dependant!)
  - Fitness function





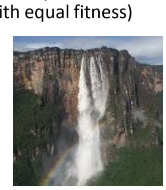
Features of landscapes relevant to heuristic search

- Number, fitness, and distribution of local optima or peaks
- Fitness differences between neighboring points (ruggedness).
- Presence and structure of *plateaus*, *neutral networks* (terrains with equal fitness)

Trentino Mountains

Earth pyramids, Tyrol, Italy

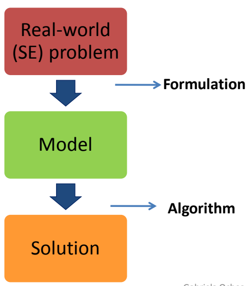
M. Fuji, Japan



M. Auyantepui, Venezuela (Angel Falls, Highest Waterfall)

Summary of optimisation problems

Many challenging applications in science and industry can be formulated as optimisation problems!



Problem Model
- Problem representation
- Constraints
- A fitness function

Optimisation/search Algorithm
- Exact methods
- Approximate (heuristic) methods

Solution to the Model
- Feasible candidate solution
- Lead to the optimal (or good enough) value of the objective function

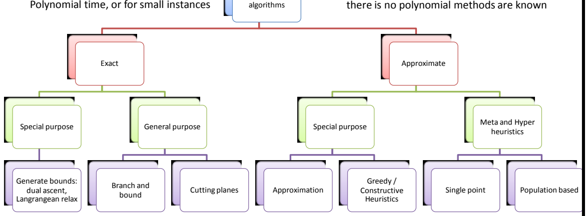
Outline

1. Optimisation problems
   - Optimisation & search
   - Classic mathematical models
   - Two canonical examples (Knapsack, TSP)
2. **Optimisation methods**
   - Heuristics and metaheuristcis
   - Single point algorithms
   - Population-based algorithms
3. Autonomous search and hyper-heuristics

Optimisation/search algorithms

- Guarantee finding optimal solution
- Useful when problems can be solved in Polynomial time, or for small instances

- Do not Guarantee finding optimal solution
- For most interesting optimisation problems there is no polynomial methods are known



Approximation algorithms:
- An attempt to formalise heuristics (emerged from the field of theoretical computer science)
- Polynomial time heuristics that provide some sort of guarantee on the quality of the solution

Terminology and dates

- *Heuristic*: Greek word *heuriskein*, the art of discovering new strategies to solve problems
- Heuristics for solving optimization problems, G. Poyla (1945)
  - A method for helping in solving of a problem, commonly informal
  - "rules of thumb", educated guesses, or simply common sense
- Prefix *meta*: Greek for "upper level methodology"
- *Metaheuristics*: term was introduced by Fred Glover (1986).
- Other terms: *modern heuristics*, *heuristic optimisation, stochastic local search*

- G. Poyla, *How to Solve it*. Princeton University Press, Princeton NJ, 1945
- F. Glover, *Future Paths for Integer Programming and Links to Artificial Intelligence*, *Computers & Ops. Res*, Vol. 13, No.5, pp. 533-549, 1986.

What is a heuristic?

- An optimisation method that tries to exploit problem-specific knowledge, for which we have no guarantee to find the optimal solution

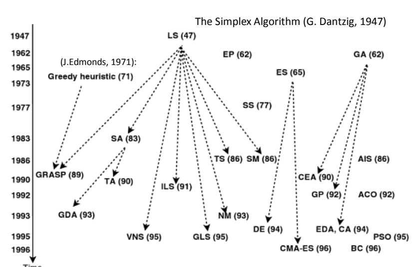
Construction
- Search space: partial candidate solutions
- Search step: extension with one or more solution components
- Example in TSP: nearest neighbour

Improvement
- Search space: complete candidate solutions
- Search step: modification of one or more solution components
- Example in TSP: 2-opt

What is a metaheuristic?

- Extended variants of improvement heuristics
- General-purpose solvers, usually applicable to a large variety of problems
- Use two phases during search
  - Intensification (exploitation): focus the applications of operators on high-quality solutions
  - Diversification (exploration): systematically modifies existing solutions such as new areas of the search space are explored

Genealogy of metaheuristics



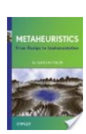


FIGURE 1.8 Genealogy of metaheuristics. The application to optimization and/or machine learning is taken into account as the original date.

Metaheuristics: From Design to Implementation
By El-Ghazali Talbi (2009)

Key components of metaheuristics

Problem Representation	• Describes encoding of solutions • Application of search operators
Fitness Function	• Often same as the objective function • Extensions might be necessary (e.g.. Infeasible solutions)
Search/Variation Operators	• Closely related to the representation • Mutation, recombination, ruin-recreate
Initial Solution(s)	• Created randomly • Seeding with higher quality or biased solutions
Search Strategy	• Defines intensification/diversification mechanisms • Many possibilities and alternatives!

Problem representations in SBSE

M. Harman, S. A. Mansouri, and Yuanyuan Zhang (2012) Search-based software engineering: Trends, techniques and applications. *ACM Comput. Surv*. 45, 1, Article 11, 61 pages.

Bit String/Matrix
- Network protocols: synthesis secure protocols
- Requirements: selection, system design,fairness analysis, etc.
- Design Tools: learning automate, sw integration, OO, etc.
- Coding tools: splitability analysis
- SW Verification: model checking
- Testing: test selection, …
- Maintenance: clone refactoring, library refactoring

Permutation
- Test case prioritisation
- Requirements prioritisation

Vector of Integers
- …

Vector of Real No.
- …

String
- …

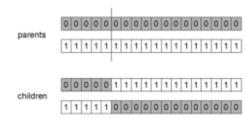
Trees
- …

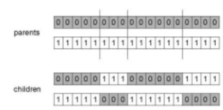
Graphs
- …

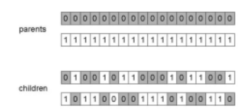
Search operators for binary representation

Mutation:
- Alter each gene independently with a probability P_m (mutation rate)
- Typically: 1/chromosome_length

Recombination:
- One-point
- N-point
- Uniform
- P_c typically in range (0.6, 0.9)







Search operators for permutation representation

Mutation: Small variation in one permutation, e.g.: swapping values of two randomly chosen positions,

1 3 5 2 6 4 7 8 → 1 3 7 2 6 4 5 8

Recombination: Combining two permutations into two new permutations:
- choose random crossover point
- copy first parts into children
- create second part by inserting values from other parent:
  - in the order they appear there
  - beginning after crossover point
  - skipping values already in child

1 3 5 | 2 6 4 7 8 → 1 3 5 4 2 8 7 6
8 7 6 | 5 4 3 2 1 → 8 7 6 2 4 1 3 5

Hill-climbing search

Like *climbing a mountain in thick fog with amnesia*

Algorithm 1 Best-improvement (left) and first-improvement (right) algorithms.

Best-improvement:

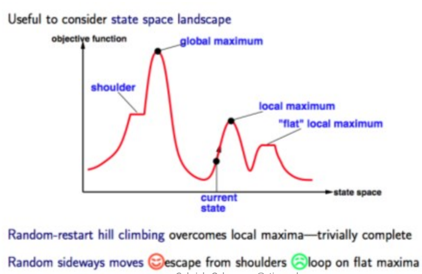
Choose initial solution $s \in S$
repeat
  choose $s' \in V(s)$, such that $f(s') = max_{x \in V(s)} f(x)$
  if $f(s) < f(s')$ **then**
    $s \leftarrow s'$
  end if
until s is a Local optimum

First-improvement:

Choose initial solution $s \in S$
repeat
  choose $s' \in V(s)$ using a predefined random ordering
  if $f(s) < f(s')$ **then**
    $s \leftarrow s'$
  end if
until s is a Local optimum

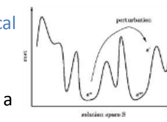
Hill-climbing search

Problem: depending on initial state, can get stuck in local maxima



Simulated annealing

- Key idea: provides a mechanism to escape local optima by allowing moves that worsen the objective function value
- *Annealing*: the physical process of heating up a solid and then cooling it down (slowly) until it crystallizes
  - candidate solutions → states of physical system
  - objective function → thermodynamic energy
  - globally optimal solutions → ground states
  - parameter *T* → physical temperature



Google Scholar citations: 31,477

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. (1983). Optimization by simulated annealing. *Science*, 220, 671–680.

Simulated Annealing – Algorithm

1. Start with a random solution *s*
2. Choose some "nearby" solution *s'*
3. If the new solution is better (i.e. $f(s') \le f(s)$) , take it as the current solution (= accept it)
4. If it is worse, accept it with a probability that depends on the deterioration $f(s)-f(s')$ and a global parameter *T* (the temperature)

Cooling schedule: a mechanism for reducing the temperature

Metropolis acceptance criterion

$$Paccept(T,s,s') = \begin{cases} 1 & \text{if } f(s') \le f(s) \\ exp(\frac{f(s)-f(s')}{T}) & \text{otherwise} \end{cases}$$

Tabu search

- Key idea: use aspects of search history escape local optima by allowing moves
- *Simple Tabu search*
  - Associate tabu attributes with candidate solutions or solution components
  - Forbid steps to search positions recently visited based on tabu attributes

```
Procedure Tabu Search (TS)
    determine initial candidate solution s
    while NOT termination criterion {
        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'
    }
```

The word *'tabu'* comes from *Tongan*, a language of Polynesia, used by the locals to indicate things that cannot be touched because they are sacred.

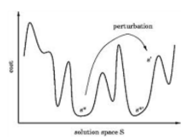
F. Glover (1989). Tabu Search - Part 1. *ORSA Journal on Computing* 1 (2): 190–206. Google cites: 5,675
F. Glover (1990). Tabu Search - Part 2. *ORSA Journal on Computing* 2 (1): 4-32. Google cites: 3,684
R. Battiti, G. Tecchiolli (1994) The reactive tabu search . *ORSA journal on computing* 6 (2): 126-140.

Iterated local search

- Key idea: use two stages
  - Subsidiary local search for efficiently reaching local optima (intensification)
  - Perturbation stage, for effectively escaping local optima (diversification)
- Acceptance criterion: to control diversification vs. intensificaction

Procedure Iterated Local Search (ILS)

determine initial candidate solution *s*
perform *subsidiary local search* on *s*
while NOT *termination_criterion* {
  r = *s*
  perform *perturbation* on *s*
  perform *subsidiary local search* on *s*
  based on *acceptance criterion*
    keep *s* or revert to *s* = *r*
}



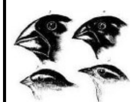
Key idea rediscovered several times with different names (80s &90s). Term *iterated local search* proposed HR Lourenço, OC Martin, T Stützle(2003). Iterated local search. *Handbook of metaheuristics,* 320-353, Springer . Google cites: 964

Gabriela Ochoa, goc@stir.ac.uk 33

Evolutionary algorithms: inspiration

Natural Selection
1. Variation
2. Hereditary transmission
3. High rate of population growth
4. Differential survival and reproduction

Charles Darwin and Alfred Wallace: Theory of evolution by means of Natural Selection (1859)



NATURE		COMPUTER
Environment	⟷	Problem
Individual	⟷	Candidate Solution
Fitness	⟷	Quality

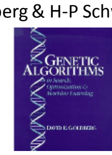
Fitness → chances for survival and reproduction

Quality → chance for seeding new solutions

Gabriela Ochoa, goc@stir.ac.uk 34

Origins of evolutionary algorithms

- Evolutionary Programming
  - Fogel, Owens, Walsh (1962)
- Evolution Strategy:
  - 60s and 70s. I. Rechenberg & H-P Schwefel
- Genetic Algorithms:
  - John Holland (1975).
  - David Goldberg (1989)



Google Scholar citations: 63,968

Alan Turing (1912 – 1954). Mathematician, wartime code-breaker and pioneer of computer science Article: “Computing Machinery and Intelligence,” (1950) described how evolution and natural selection might be used to automatically create an intelligent computer program

Gabriela Ochoa, goc@stir.ac.uk 35

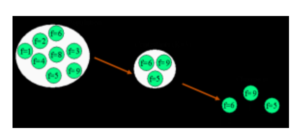
Genetic algorithms

Procedure GA

Generate [P(0)]
t = 0
while NOT *Termination_Criterion* {
  Evaluate [P(*t*)]
  P' (*t*) = *Select* [P(*t*)]
  P''(*t*) = *Apply_Operators* [P'(*t*)]
  P(*t*+1) = *Replace* [P(*t*), P''(*t*)]
  t = *t* + 1
}

Replacement (population models)
- **Generational:** each generation set of parents replaced by the offspring
- **Steady-state**: one offspring is generated per generation. One member is replaced
- **Generation gap**: a proportion of the population is replaced

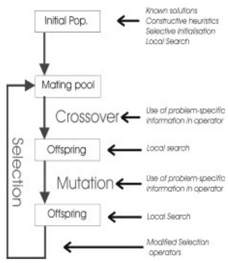
Parent selection: Better individuals get higher chance (proportional to fitness).
- Proportional selection (roulette wheel, stochastic universal sampling)
- Scaling methods
- Rank selection
- Tournament selection
- ($\mu + \lambda$)- and (μ , λ) selection



Gabriela Ochoa, goc@stir.ac.uk 36

Memetic (hybrid) algorithms

- Combination of GAs with local search operators, or GAs that use instance specific knowledge in operators
- Orders of magnitude faster and more accurate than GAs on some problems, and are the "state-of-the-art" on many problems



- The term *meme* was coined by R. Dawkins (1976)
- The term memetic algorithms by P. Moscato (1989)
- The idea of hybridisation in GAs is older

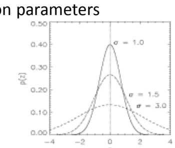
(Eiben, Smith, 2003)

Evolution strategies

- Specialised in continuous search spaces: min. $f : R^n \rightarrow R$
- Rechenberg & Schwefel in the 60s, Technical University of Berlin. Applied to hydrodynamic shape optimisation
- Special feature: self-adaptation of mutation parameters

Procedure (1+1)-ES

```
t = 0;
initialise solution x^t = ⟨ x_1^t,…,x_n^t ⟩
while NOT Termination_criterion) {
    Draw z_i from a Normal distr. for all i = 1,…,n
    y_i^t = x_i^t + z_i
    if  f(x^t) < f(y^t)  then  x^{t+1} = x^t
    else x^{t+1} = y^t
     t = t+1
 }
```



- p_s is the % of successful mutations
- $0.8 \leq c \leq 1$

- z values from Normal dist. N(0, σ)
- σ, step size, varied on the fly
- 1/5 *success rule* sets σ every k iterations
  - $\sigma = \sigma / c$ if $p_s > 1/5$
  - $\sigma = \sigma \times c$ if $p_s < 1/5$
  - $\sigma = \sigma$ if $p_s = 1/5$

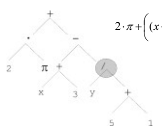
Modern evolution strategies

- Use a population: μ parents, λ offspring
- $(\mu + \lambda)$-ES: next generation crated from the *union* of parents and offspring
- (μ , λ)-ES: the best μ solutions from the offspring are chosen
- Recombination used for exchanging information
- Self-adaptation: Incorporate strategy parameter (σ, std. dev mutation strength) into the search process
- CMA-ES: (Covariance Matrix Adaptation ES, N. Hansen, A. Ostermeier, 1996)
  - State-of-the-art ES, unconstrained or bounded constraint, 3 – 100 dim.
  - Source code: https://www.lri.fr/~hansen/cmaes_inmatlab.html
- Differential Evolution (K. Price and R. Storn, 1996)
  - Recent and powerful EA for continuous optimisation, elegant and simple
  - Key idea: using vector differences for perturbing the vector population
  - Source code: http://www1.icsi.berkeley.edu/~storn/code.html

Genetic programming

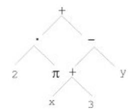
- Evolve a population of computer programs
- Applied to: machine learning tasks (prediction, classification…)
- Representation
  - Non-linear genomes: trees, graphs
  - Linear genomes: grammatical evolution (Ryan, 1999)
- Main difference with GAs:
  - Search space of tree structures different sizes
  - Solutions are *parse-trees,* syntactic structure according to some grammar
  - Nodes in the parse tree are either:
    - *Terminal set T* (leaf nodes): independent variables of the problem, zero argument functions, random constants, terminals with side effects (eg. "turn left")
    - *Function set S* (interior nodes): arithmetic (+,-,*)/logic operations (^,˅)

Genetic programming



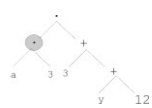
$$2\cdot\pi+\left((x+3)-\frac{y}{5+1}\right)$$

Parent 1

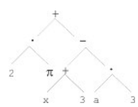


Mutation: replace randomly chosen sub-tree by randomly generated tree

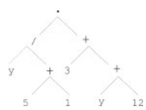
Recombination: interchange randomly chosen sub-trees



Parent 2



Child 1

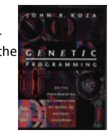


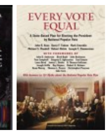
Child 2

Genetic programming origins and sources

Origin 1985: NL Cramer (1985) A Representation for the Adaptive Generation of Simple Sequential Programs. *In Proceedings of the 1st International Conference on Genetic Algorithms*, John J. Grefenstette (Ed.). 183-187.

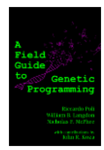
1992 book: *On the Programming of Computers by Means of Natural Selection* from The MIT Press.





John R. Koza

Scientist and business man. Popularised GP, proposed and funds the HUMMIES award. Millionaire, co-inventor of rub-off instant lottery game ticket, proposed a plan for electing the US president by popular vote.



Bill Langdon

The GP Bibliography

http://www.cs.bham.ac.uk/~wbl/biblio/README.html

(Poli, Langdon, and McPhee, 2008)

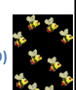
http://www.gp-field-guide.org.uk

Other population-based algorithms: the social behaviour metaphor

Ant colony optimisation (ACO)

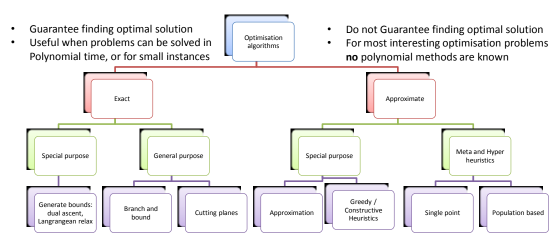
- Dorigo, Di Caro & Gambardella (1991).
- Inspired by the behaviour of real ant colonies
- A set of software agents artificial ants search for good solutions
- Problem transformed to finding the best path on a weighted graph.
- Ants build solutions incrementally by moving on the graph
- http://www.aco-metaheuristic.org/
- http://www.scholarpedia.org/article/Ant_colony_optimization

Particle Swarm Optimization (PSO)



- Eberhart & Kennedy, 1995
- Inspired by social behaviour of bird flocking or fish schooling
- Solutions (called particles) fly through the search space by following the current optimum particles
- At each iteration they accelerate towards the best locations
- http://www.swarmintelligence.org/
- http://www.scholarpedia.org/article/Particle_swarm_optimization

Summary: Optimisationalgorithms



Metaheuristcs, modern heuristics, stochastic local search (key components):

1. Problem representation
2. Fitness function
3. Search/variation operators
4. Solution initialisation
5. Search strategy (balance exploration & exploitation, avoid local optima)

Outline

1. Optimisation problems
   - Optimisation & search
   - Classic mathematical models
   - Two canonical examples (Knapsack, TSP)
2. Optimisation methods
   - Heuristics and metaheuristcis
   - Single point algorithms
   - Population-based algorithms
3. **Autonomous search and hyper-heuristics**

Increase in complexity

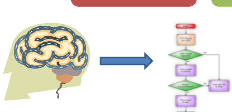
- Real world problems are complex
- Heuristic search algorithms are powerful but
  - There are too many variants
  - They are getting increasingly complex
    - Many parameters
    - Many design/algorithmic components

- Advantage
  - More variety and more flexible algorithms
  - Fit to different problems

- Disadvantage
  - Need to select an algorithm, or
  - Select the algorithm components/operators and/or set their parameters

Algorithm selection, configuration and tuning

Holy-Grail: *Finding the most suitable optimisation/search algorithm and its correct setting for solving a given problem*



Static/dynamic

Can we automate these processes?

Autonomous/adaptive (self-*) search approaches

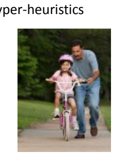
- Different approaches (that share common principles) have been developed in different communities (OR, OP, AI, ML, CS)
- Incorporate ideas from machine learning and statistics

Offline, Static Configuration

- Algorithm selection
- Algorithm portfolios
- Algorithm configuration and Parameter tuning
  - Racing, ParamILS, SPO
- Hyper-heuristics

Online, Dynamic Control

- Adaptive operator selection
- Parameter control
- Reactive search
- Adaptive memetic algorithms
- Hyper-heuristics



What is a Hyper-heuristic?

- A *higher level* heuristic which manages a *set* of *low-level* heuristics
- An optimisation algorithm with a modular design
- Benefits from combining the strengths of several simpler heuristics
- Uses only limited problem-specific information

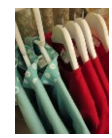
Heuristics to choose heuristics

Hyper-heuristic

Heuristic 1 | Heuristic 2 | Heuristic 3 | Heuristic n

What Motivates Hyper-Heuristic Research?

- Decision support systems that are *off the peg vs. Taylor made*
- Develop the ability to automatically work well on different problems
- Increase the generality and applicability of these methods to solve complex real-world problems



vs.

Classification of hyper-heuristics

Hyper-heuristics

Heuristic ***Selection***

Heuristic ***generation***

Construction heuristics | Improvement heuristics | Construction heuristics | Improvement heuristics

Fixed, human-designed low level heuristics

Heuristic *components*

Hyper-ILS or adaptive ILS

Procedure Hyper-ILS

```
s0 = GenerateInitialSolution
s* = HyperImproveStage(s0)
while NOT Termination_criterion) {
    s'= HyperPerturbStage(s*)
    s'*= HyperImproveStage(s')
    if f(s'*) < f(s*)
       s* = s'*
}
```

- Pool of operators of different type
- Reinforcement learning used to adaptively select the best operator to apply at each iteration
- Either or both
  - Improvement stage
  - Perturbation stage

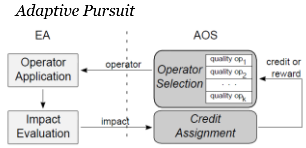
- Successful applications to both Vehicle routing and Course time-tabling
- Research questions
  - Metrics to gather feedback from the search, how to combine them
  - Mechanism for *adaptive operator selection*

Given a pool of operators

- Simple Random Perturbation (SRP)
- Best Single Perturbation (BSP)
- Statistical Dynamic Perturbation (SDP)
- Double Dynamic Perturbation (DDP)
- Swap (SWP)
- Two Points Perturbation (2PP)
- Move to Less Conflict (MLC)
- Burke-Abdhulla (BA)
- Conant-Pablos (LSA)

QUESTION: Given f K search operators

- How to select **(on the fly)** the operator to be applied next, considering the history of their performance?
- Measuring performance ➔ Assigning credit ➔ Selecting the operator: *Fitness Improvement + Extreme Credit + Adaptive Pursuit*



Application to Timetabling

Summary of hyper-heuristics

A hyper-heuristic is an automated methodology for selecting or generating heuristics to solve computational search problems

- Main feature: search in a space of heuristics
- Term used for '*heuristics to choose heuristics*' in 2000
- Ideas can be traced back to the 60s and 70s
- Two main type of approaches
  - Heuristic selection
  - Heuristic generation
- Ideas from online and offline machine learning are relevant, as are ideas of meta-level search

References: Books

- Burke , E. K; Kendall, G., (Eds.) (2005) Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques, Springer.
- A.E. Eiben and J.E. Smith (2003), Introduction to Evolutionary Computing, Springer,
- Hoos, H; Stutzle, T. (2004) Stochastic Local Search Foundations and Applications, Elsevier.
- Z. Michalewicz, D.B. Fogel (2000) How to Solve It: Modern Heuristics, Springer.
- Rothlauf, F. (2011) Design of Modern Heuristics Principles and Application. Natural Computing Series, Springer
- S. Russell, P. Norvig (2009) Artificial Intelligence: A Modern Approach (3rd Edition) Prentice Hall.
- Talbi, E-G (2009), Metaheuristics: From Design to Implementation, Wiley.

References: Hyper-heuristics

- E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan and R. Qu (2013) Hyper-heuristics: A Survey of the State of the Art, *Journal of the Operational Research Society*. 206(1): 241-264
- E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and J. Woodward (2010). A Classification of Hyper-heuristics Approaches, Handbook of Metaheuristics, International Series in Operations Research & Management Science, M. Gendreau and J-Y Potvin (Eds.), Springer, pp.449-468.
- E. K. Burke, M. Gendreau, G. Ochoa, J. Walker. Adaptive Iterated Local Search for Cross-domain Optimisation. *Genetic and Evolutionary Computation Conference (GECCO-2011)*, ACM, pp. 1987-1994.
- G Ochoa, EK Burke (2014) HyperILS: An Effective Iterated Local Search Hyper-Heuristic for Combinatorial Optimisation, *International Conference on the Practice and Theory of Automated Timetabling (PATAT 2014)*
- J.A Soria-Alcaraz, G. Ochoa, J. Swan, M. Carpio, H. Puga , E.K. Burke (2014) Effective Learning Hyper-heuristics for the Course Timetabling Problem. *European Journal of Operational Research*. 238(1): 77-8.