

CS4618 Artificial Intelligence I

Today: Introduction to Heuristic Search
Randomised Search Heuristics

Thomas Jansen

October 17th

Plans for Today

① Randomised Search Heuristics

Motivation and Overview

Concrete Randomised Search Heuristics

② Evolutionary Algorithms

Randomised Search Heuristics

Evolutionary Algorithms

③ Summary

Summary & Take Home Message

Plans for the Next Weeks

so far **blind search**

knows initial state, transition-model, actions, costs, goal-test

Examples breadth-first search, depth-first search, depth-limited search, iterative deepening depth-first search, uniform-cost search, bidirectional search

and **informed search**

knows in addition evaluation function f

(often with heuristic h)

Examples A* search, greedy best-first search, iterative deepening A* search, simplified memory-bounded A* search, recursive best-first search

Observation **not** only goal considered

also path to solution

Now only 'optimal goal' interesting

Searching for a Solution

Consider **problem** given by

- set of potential solutions $S \approx$ state space
- values of potential solutions $f: S \rightarrow \mathbb{R} \approx$ path cost

What about actions? What about costs of actions?

Observation **not** useful notions in some problems

Example problems

- 8-queens problem
- integrated circuit design
- factory floor layout
- job-shop scheduling
- telecommunication network optimisation
- vehicle routing
- automatic programming
- ...

Solving General Search Problems

Consider **problem** given by

- set of potential solutions S
- values of potential solutions $f: S \rightarrow \mathbb{R}$

Observation search problem of this kind
 $\hat{=}$ general **optimisation problem** with **objective function** f
'Search for optimal solution $s \in S$.'
(optimal $\hat{=}$ minimal or maximal under f)

Assumptions

- can construct arbitrary $s \in S$ efficiently
- can compute $f(s)$ for arbitrary $s \in S$ reasonably fast

General Randomised Search Heuristics

Consider **problem** given by

- set of potential solutions S
- values of potential solutions $f: S \rightarrow \mathbb{R}$

Assumptions

- can construct arbitrary $s \in S$ efficiently
- can compute $f(s)$ for arbitrary $s \in S$ reasonably fast

Remark

- additional assumptions about **search space** S may be made
Example **neighbourhood** N ($N: S \rightarrow 2^S$)
- additional assumptions about **objective function** f **not made**

In the following

- **always** S finite
- **sometimes** $S = \{0, 1\}^n$
- maximisation of f

Local Search

Local Search

1. $t := 1$; Choose $x_t \in S$ uniformly at random.
2. Repeat
3. Select $y \in N(x_t)$.
4. $t := t + 1$; If $f(y) > f(x_{t-1})$ then $x_t := y$ else $x_t := x_{t-1}$.
5. Until 'decide to stop'
6. Output x_t .

Observation '3. Select $y \in N(x_t)$.'
 not sufficiently precise

Solution precise specification
 \rightsquigarrow different **variants** of **local search**

Observation '5. Until decide to stop' **not** sufficiently precise
 precise definitions **later**

Local Search Variants

Local Search

1. $t := 1$; Choose $x_t \in S$ uniformly at random.
2. Repeat
3. Select $y \in N(x_t)$.
4. $t := t + 1$; If $f(y) > f(x_{t-1})$ then $x_t := y$ else $x_t := x_{t-1}$.
5. Until 'decide to stop'
6. Output x_t .

First-Improvement Local Search

Consider all $y \in N(x_t)$ in **fixed** order
and stop at first y with $f(y) > f(x_t)$.

Best-Improvement Local Search

Consider all $y \in N(x_t)$ (in arbitrary order)
and use one y with $f(y) = \max\{f(z) \mid z \in N(x_t)\}$.

Random Local Search

Consider all $y \in N(x_t)$ in **random** order
and stop at first y with $f(y) > f(x_t)$.

Deciding to Stop

Remember **need** to specify 'decide to stop'

Possible Stopping Criteria

- Stop if $f(x_t) \geq \max\{f(y) \mid y \in N(x_t)\}$.
 $\hat{=}$ Stop if no further improvement possible. ($\hat{=}$ local optimum)
Caution Example $S := \mathbb{Z}$, $N(z) = \{z - 1, z + 1\}$, $f(z) = z$.
Observation never stops
- Stop after a pre-specified number of steps.
simple to implement, but **difficult** to set
- Stop after a pre-specified f -value is reached.
simple to implement, but **difficult** to set
- Stop if time is up.
simple to implement
- ...

Metropolis Algorithm

Parameter $T \in \mathbb{R}^+$ (temperature)

Metropolis Algorithm

1. $t := 1$; Choose $x_t \in S$ uniformly at random.
2. Repeat
3. Select $y \in N(x_t)$ uniformly at random.
4. $t := t + 1$
 With probability $\min\{1, e^{(f(y)-f(x_{t-1}))/T}\}$ set $x_t := y$
 else $x_t := x_{t-1}$.
5. Until 'decide to stop'
6. Output x_t .

Remark decision about stopping as for local search
 'Stop, if no improvement within pre-specified number of steps.'
 also **useful**
simple to implement, but **difficult** to set

About T in the Metropolis Algorithm

Parameter $T \in \mathbb{R}^+$ (temperature)

Metropolis Algorithm

1. $t := 1$; Choose $x_t \in S$ uniformly at random.
2. Repeat
3. Select $y \in N(x_t)$ uniformly at random.
4. $t := t + 1$
 With probability $\min\{1, e^{(f(y)-f(x_{t-1}))/T}\}$ set $x_t := y$
 else $x_t := x_{t-1}$.
5. Until 'decide to stop'
6. Output x_t .

Observation

- Metropolis can **escape** from local optima
- large $T \rightsquigarrow$ easy escape $\hat{=}$ willing to take large risks
- small $T \rightsquigarrow$ difficult escape $\hat{=}$ unwilling to take large risks

Simulated Annealing

Parameter $T: \mathbb{N} \rightarrow \mathbb{R}^+$ (cooling schedule)

Simulated Annealing

1. $t := 1$; Choose $x_t \in S$ uniformly at random.
2. Repeat
 3. Select $y \in N(x_t)$ uniformly at random.
 4. $t := t + 1$
 With probability $\min\{1, e^{(f(y)-f(x_{t-1}))/T(t)}\}$ set $x_t := y$
 else $x_t := x_{t-1}$.
5. Until 'decide to stop'
6. Output x_t .

Remark Metropolis $\hat{=}$ simulated annealing with fixed temperature
 most often cooling schedule non-increasing

Randomised Search Heuristics

Scenario Find some $s \in S$ that optimises $f: S \rightarrow \mathbb{R}$.

Always necessary assumptions

- $\forall s \in S: s$ is easy to construct
- $\forall s \in S: f(s)$ is reasonably easy to compute

Here we make the following additional assumptions

- S finite (often $S = \{0, 1\}^n$)
- 'optimise' means maximise

Common additional assumption

- neighbourhood $N: S \rightarrow 2^S$ known

Some Randomised Search Heuristics

- local search
 - first-improvement local search
 - best-improvement local search
 - random local search
- Metropolis algorithm
 - $\hat{=}$ random local search **but** accepting new search point with probability $\min\{1, e^{(f(y)-f(x_{t-1}))/T}\}$, parameter $T \in \mathbb{R}^+$
- simulated annealing
 - $\hat{=}$ Metropolis algorithm **but** with annealing schedule $T: \mathbb{N} \rightarrow \mathbb{R}^+$ instead of fixed temperature

Evolutionary Algorithms

Parameters

- population size $\mu \in \mathbb{N}$
- offspring population size $\lambda \in \mathbb{N}$
- crossover probability $p_c \in [0, 1]$

Evolutionary Algorithm

1. Choose $x_1, x_2, \dots, x_\mu \in S$ uniformly at random.
2. Repeat
 3. For $i \in \{1, 2, \dots, \lambda\}$ do
 4. With probability p_c select $z_1, z_2 \in \{x_1, x_2, \dots, x_\mu\}$.
 $z := \text{crossover}(z_1, z_2)$
 Else select $z \in \{x_1, x_2, \dots, x_\mu\}$.
 5. $y_i := \text{mutation}(z)$
 6. Select new x_1, x_2, \dots, x_μ out of old x_1, x_2, \dots, x_μ and $y_1, y_2, \dots, y_\lambda$.
7. Until 'decide to stop'
8. Output x_i with $f(x_i) = \max\{f(x_j) \mid 1 \leq j \leq \mu\}$.

Modules of Evolutionary Algorithms

Observation **need** to specify

- Selection for Reproduction

- **uniform** $\text{Prob}(\text{select } z) = 1/\mu$
- **fitness-proportional** $\text{Prob}(\text{select } z) = f(z) / \sum_{i=1}^{\mu} f(x_i)$
- **tournament** Select k uniform and of those a best.

- Selection for Survival

- strictly fitness-based among all (**plus-selection**)
- strictly fitness-based among offspring (**comma-selection**)
- **uniform**

- Variation here only for $S = \{0, 1\}^n$

- Crossover(z_1, z_2)

- **uniform crossover** Independently for all $1 \leq i \leq n$ select $y[i] \in \{z_1[i], z_2[i]\}$ uniformly at random.
- **k -point crossover** Choose $p_1 < p_2 < \dots < p_k \in \{0, 1, \dots, n\}$ uniformly at random.

$y = z_1[1]z_1[2] \dots z_1[p_1]z_2[p_1 + 1]z_2[p_1 + 2] \dots z_2[p_2]z_1[p_2 + 1] \dots$
 number of crossover points $k \in O(1)$, most often $k \in \{1, 2\}$

Modules of Evolutionary Algorithms (continued)

Observation **need** to specify

- Selection for Reproduction
- Selection for Survival
- Variation here only for $S = \{0, 1\}^n$
 - Crossover(z_1, z_2)
 - Mutation(z)
 - **standard bit mutation** Independently for all $1 \leq i \leq n$ with probability p_m set $y[i] := 1 - z[i]$ else $y[i] := z[i]$.
mutation probability $p_m \in (0, 1/2]$, most often $p_m = 1/n$
 - **b -bit mutation** Select $p_1 \neq p_2 \neq \dots \neq p_b \in \{1, 2, \dots, n\}$ uniformly at random. Set $y[i] := 1 - z[i]$ for $i \in \{p_1, p_2, \dots, p_b\}$ and $y[i] := z[i]$ otherwise.
number of flipping $b \in O(1)$, most often $b \in \{1, 2\}$

Summary & Take Home Message

Things to remember

- search problems, optimisation problems
- local search (first improvement, best improvement, random)
- stopping criteria
- Metropolis algorithm, simulated annealing
- evolutionary algorithms
 - modules: selection (uniform, fitness-proportional, tournament, plus, comma)
 - modules: variation (uniform crossover, k -point crossover, standard bit mutations)

Take Home Message

- Optimisation is searching for an optimal solution.
- RSH are easy to implement and easy to apply.
- Many different randomised search heuristics exist.