Set-based Search Example: Genetic Algorithms

CPSC 433: Artificial Intelligence
Fall 2024

Jonathan Hudson, Ph.D.
Assistant Professor (Teaching)
Department of Computer Science
University of Calgary

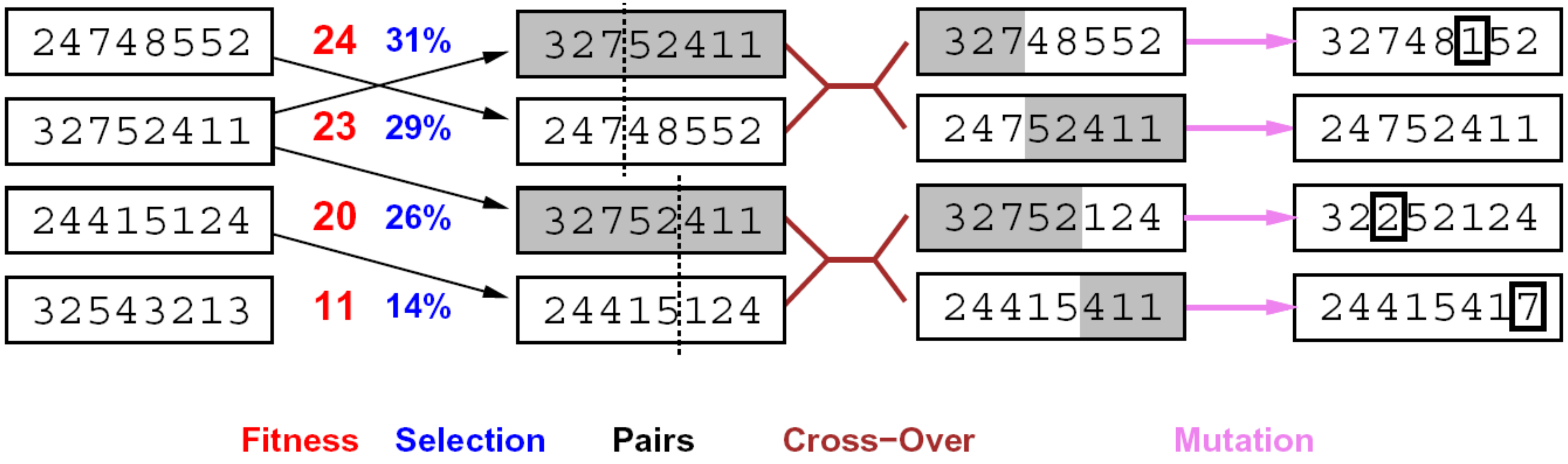
August 8, 2024

UNIVERSITY OF CALGARY

Genetic Algorithms

Genetic Algorithms

(not a representation for our knapsack problem)



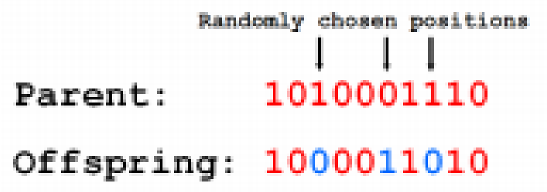
- Genetic algorithms use a natural selection metaphor
  - Keep best N hypotheses at each step (selection) based on a fitness function
  - Also have pairwise crossover operators, with optional mutation to give variety
- Possibly the most misunderstood, misapplied (and even maligned) technique around

Encoding

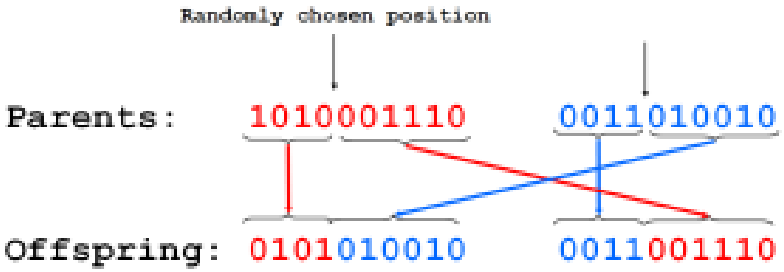
- (could be used for knapsack as alternatives to our Extension rules)

GA Operators Brief Introduction

Mutation



Crossover



Knapsack Problem

Knapsack Problem

- Problem: Filling a knapsack with fixed capacity with items. Each item has a given weight and value to you. (0/1 Knapsack Problem)
- We have items I we will index from 1 to n
  - We have associated weights W and values V
  - $W = (w_1, \dots, w_n), w_i > 0$
  - $V = (v_1, \dots, v_n), v_i > 0$
  - To simplify each $item_i \in I$ can be considered as a pair $item_i = (w_i, v_i)$
  - We have a max capacity of C
- What ways can we do this?
  - Hill climbing? Dynamic programming? Set-based search?

Knapsack Solutions

- Hill climbing
  - Estimation (just add best ratio of value to weight of things until you run out of space), quick, low memory, simple
  - Greedy algorithm (*413*)
  - Could be done via a set-based search

- Dynamic Programming (DP):
  - ***413*,** Exact algorithm (optimal solution for sure), Long running time as problem grows, lots of memory to store sub-problem expansions, very simple to design, $O(n^2)$ time and O(n) space if we treat every item as unique

UNIVERSITY OF CALGARY

Knapsack Solutions

- Genetic Algorithm (GA):
  - Estimation, never know if you found the best solution
  - Runs quickly as the problem explodes in complexity
  - memory usage rather small with basic implementation
  - quality can be highly variable with no guarantees
  - harder to design unless you know GAs

Examples

Example problem with hill climb

- We have a 1-0 knapsack with maximum capacity of 7. We have the set of items I = {(3,4), (4,5),(5,7)} where every item is a pair such that i = (weight, value).
- Hill climb – get a ratio of each item, sort by descending ratio, add each item that you can.

Ratio (w/v)	Weight (w)	Value (v)	Knapsack W	Include?
0.8	4	5	0 + 4 = 4	Yes (val=5)
0.75	3	4	4 + 3 = 7	Yes (val=9)
0.72~	5	7	7 + 5 = 12	No, max reached

- This is best solution….for this instance of problem.
- Hill-climbing uses flawed heursistic.

Example problem with hill climb

- Estimation can cause problems.
- Add item (1,1).
- We have a 1-0 knapsack with maximum capacity of 7. We have the set of items I = {(1,1), (3,4), (4,5),(5,7)}

Ratio (w/v)	Weight (w)	Value (v)	Knapsack W	Include?
1	1	1	0 + 1 = 1	Yes (val=1)
0.8	4	5	1 + 4 = 5	Yes (val=6)
0.75	3	4	5 + 3 = 8	No, max reached

- {(1,1),{4,5}) has value 6 but there is a solution value 9 with {(3,4),(4,5)}!

Example problem with hill climb

- We have a 1-0 knapsack with maximum capacity of 7. We have the set of items I = {(3,4), (4,5),(5,7)} where every item is a pair such that i = (weight, value).
- Is the problem the ratio direction?

Ratio (v/w)	Weight (w)	Value (v)	Knapsack W	Include?
1.4	5	7	0 + 5 = 5	Yes (val=5)
1.333…	3	4	5 + 3 = 8	No, max reached

- No.
- An inverted ratio would also lead to sub-optimal solution {(5,7)} which has value 5 but there is a solution value 9 with {(3,4),(4,5)}!

Example problem with dynamic programming

- We have a 1-0 knapsack with maximum capacity of 7 and a set of items I = {(1,1),(3,4), (4,5),(5,7)}, s.t. item = (weight, value)

i

Item value	Item weight	0	1	2	3	4	5	6	7
1	1	0	1	1	1	1	1	1	1
4	3	0	1	1	4	5	5	5	5
5	4	0	1	1	4	5	6	6	9
7	5	0	1	1	4	5	7	8	9

j

- Start from 0 and fill in the table, for each row in order add column first. Knapsack limit (top row) will be called i, current item will be j.
  - If weight(j) is bigger then i, write value at T[j-1][i],
  - Else select the maximum(value(j) + value at T[j-1][I – weight(j)], T[j-1][i])
- Maximum value will be at the lowest row, on the last index. (9 in this case)

GA Solution

What do we need to design set-based solution?

- Facts
- Extension Rules
- Search Control Direction (the choice of extension rules)
- Search Instance

What do we need to design set-based solution?

Want Model A_{set}

- Facts
  - Parts that will fill our state set
    1. Parts of a single solution (like in resolution)
    2. Different possible full solutions (like in genetic/evolutionary algorithms)
  - We get for free $S_{set} \subseteq 2^F$ (also know as the power set of F -> all subsets of F)
- Extension Rules

What do we need to design set-based solution?

Want Model A_{set}

- Facts (S_{set})
- Extension Rules
  - How we move between subsets of facts
  - If we are in some state s and moving to s', both are subsets of F
  - We take some subset A of facts from s and replace with another subset of facts B
  - Extension rules are how the B is determined based on which A is used
  - $Ext \subseteq \{A \rightarrow B \mid A, B \subseteq F\}$
  - We get for free $T_{set} = \{(s, s') | \exists A \rightarrow B \; with \; A \subseteq s \; and \; s' = (s - A\} \cup B\}$

What do we need to design set-based solution?

Have $A_{set} = (S_{set}, T_{set})$

- Facts (S_{set})
- Extension Rules (T_{set})

Want K_{set} to complete $P_{set} = (A_{set}, Env\ , K_{set})$

- Search Control Direction (the choice of extension rules)
  - $K_{set}(s,e) = (s - A) \cup B$
  - f_{wert}, f_{select} deal with choosing the A $\rightarrow B \in Ext$
  - f_{wert} gives a value to each extension rule in current state
  - f_{select} chooses between ties
- Search Instance

UNIVERSITY OF CALGARY

What do we need to design set-based solution?

Have $A_{set} = (S_{set}, T_{set})$

- Facts (S_{set})
- Extension Rules (T_{set})

Have K_{set} to complete $P_{set} = (A_{set}, Env\ , K_{set})$

- Search Control Direction (the choice of extension rules) (K_{set})

Want where to start

- Search Instance
  - Some initial set of facts $s_0 \in S_{set}$, and a goal G to decide when done (ex. time passing, quality of set of facts stops improving, no more extension rules apply [i.e. resolution])

Facts

What do we need to design evolutionary algorithm solution to 0/1 knapsack?

- Facts
  - We will deal with facts that are possible full solutions to the problem
    - That is each fact is some selection of items below the capacity C
    - (this is our choice, we could allow invalid solutions (too high capacity) as an alternative

  - $F = \{(x_1, .., x_m) \mid x_i \in \{0,1\} \; and \sum_{i=1..m} w_i * x_i \leq C\}$
    - We have chosen to presume some consistent ordering of the available items in set I

UNIVERSITY OF CALGARY

What do we need to design evolutionary algorithm solution to 0/1 knapsack?

- Facts

  - If $I = \{item_1, item_2, item_3\}$ = {(3,4), (4,5),(5,7)}

  - Then a f ∈ F could be f $= (1,1,0)$ which would mean choosing $item_1 = (3,4)$ and $item_2 = (4,5)$ but not the first item f $= (0,1,1)$ would not be valid fact as it is over capacity

Extension Rules

What do we need to design evolutionary algorithm solution to 0/1 knapsack?

- Extension Rules
  - Extension rules create new solutions from some subset of solutions
  - We use two biologically inspired rules (mutation, crossover)
  - $Ext = \{A \rightarrow B \mid A, B \subseteq F \; and \; \left(Mutation\ (A,B) \; or \; Crossover\ (A,B)\right)\}$

UNIVERSITY OF CALGARY

Mutation

Mutation

- $Mutation(A, B)$
  - $A \ = \{(x_1, .., x_m)\}$
  - $B \ = \{(x_1, .., x_m), (y_1, .., y_m)\}$
  - where
    - $x_1 = y_1$ except for some $1 \leq i \leq m$ we will make $y_i = \ \neg x_i$
    - We will flip one 0 to 1, or vice versa to add or remove an item from new knapsack solution

  - Also known as single-point mutation

UNIVERSITY OF CALGARY

Mutation Example

- $Mutation(A, B)$

- If $I = \{item_1, item_2, item_3\}$ = {((3,4), (4,5),(5,7)}

$$C = 7$$

$$A = \{(0,1,0)\}$$
$$B = \{(0,1,0), (1,1,0)\}$$

Or even

$$B = \{(0,1,0), (0,0,0)\}$$

Crossover

Crossover

- $Crossover(A, B)$
  - $A \ = \{(x_1, .., x_m), (y_1, .., y_m)\}$
  - $B \ = \{(x_1, .., x_m), (y_1, .., y_m), (z_1, .., z_m)\}$
  - where
    - Each z_i is selected by flipping a coin and selecting either x_i or y_i

Crossover Example

- $Crossover(A, B)$

- If $I = \{item_1, item_2, item_3\} =$ {((3,4), (4,5),(5,7)}
$$C = 7$$

$$A = \{(0,1,0), (1,0,0)\}$$
$$B = \{(0,1,0), (1,0,0), (1,1,0)\}$$

Or even one of the following

$$B = \{(0,1,0), (1,0,0), (0,0,0)\}$$
$$B = \{(0,1,0), (1,0,0), (1,0,0)\}$$
$$B = \{(0,1,0), (1,0,0), (0,1,0)\}$$

Search Control

What do we need to design evolutionary algorithm solution to 0/1 knapsack?

- Search Control Direction (the choice of extension rules)
  - RNG function to produce values
  - f_{wert} set to constant
  - f_{select} technically has all rules possible
    - We will choose to define it procedurally
    - f_{select} will pick mutation x% of time, mutation y% of time
    - x+y = 100%
    - f_{select} also picks which individuals A are used by Mut/Cross to produce B
      - Fitness-based (value individuals by quality of solution (i.e. total value of items) and bias selection of A towards more fit individuals)

Search Instance

What do we need to design evolutionary algorithm solution to 0/1 knapsack?

- Search Instance
  - A generated set of random individuals (valid knapsack solutions) of some chosen population size (size of s_0)
    - For each random individual
      - select random knapsack items to add until capacity is reached

  - Goal function
    - Time-based?
      - Run for x minutes of real-world time
    - Counter-based?
      - Make x new solutions
    - Improvement-based?
      - Run until we aren't finding better solutions often enough
      - Often involves predicting a log curve of improvement and stopping when it flattens too much

UNIVERSITY OF CALGARY

Remarks

Considerations

- Do you always select the single top fittest individual?
  - Rank-based selection (odds based on fitness order)
  - Roulette wheel selection (odds based on fitness portion of total)
- How do you manage the growing population?
  - Do you delete one/multiple each time
  - Who do you delete
- What about diversity?
  - What if population stagnates, can you enforce valuable diversity?
- What about invalid solutions?
  - Do you define F to allow invalid facts (can move through invalid fact to better valid ones), or do you disallow valid facts but possibly make it harder to move around search space

Onward to …
and-tree-based search

Jonathan Hudson, Ph.D.
jwhudson@ucalgary.ca
https://cspages.ucalgary.ca/~jwhudson/

UNIVERSITY OF CALGARY