Search: The Core of AI

Keith L. Downing

January 4, 2024
Search is..

A (preferably methodical) process for finding something.

- Searching for:
  - **pre-existing** entities (information, objects, etc.)
  - strategies for creating/designing entities.

- Examples:
  - Web search (for just about anything)
  - AI search - creates something, doesn’t just find it.

- Uninformed -vs- Informed: Do points in the search space give information that helps the searcher to determine the next step?

- Intelligent, Goal-Directed -vs- Random - not so obvious a distinction.

- Partial -vs- Complete solutions (i.e., attempts): Could the current state of search always be considered a complete solution, though not necessarily good or optimal? Or is it often a partial state that must be incrementally enhanced to become a solution?
  - E.g. 2 approaches to origami.
  - Closed-loop -vs- Open-loop control
A Search Algorithm Classification

How to Solve It: Modern Heuristics (Michalewicz & Fogel 2000)

* Traditional methods can be parallel, but it's usually an independent parallelism

* There are many other ways to classify search algorithms.
Incremental Search: The 8-puzzle

Goal: Find a path (i.e. sequence of moves) from current state to goal state.
Popular AI puzzles: M and C, Towers of Hanoi, 8 puzzle, Cryptarithmetic, K-Queens, Knight’s Tour
3 Common Incremental Search Algorithms

Depth-First

Breadth-First

Best-First

General

Problem-Specific

"Intelligent"
Search = choosing the next partial solution to expand. 
Requires representations that are easy to extend.
High space complexity - need to save ALL nodes in the search tree (or graph).
Time complexity varies - generally high for uninformed versions but potentially much lower for informed approaches.
The heuristic is the key to A* performance. It needs to give good partial credit.
With a good (admissible) heuristic, optimal solutions are guaranteed to be found first with A*.
Usually portrayed as the growth of a search tree (or graph).
Incremental -vs- Local Search

Local Search
- The path is unimportant; only the final state matters.
- All search-space states = complete solutions (a.k.a. attempts).
- Search ↔ modifying complete solutions.
- Partial credit still very important, but now it’s given to whole (not partial) solutions.
Other Properties of Local Search

- Low space complexity - only need to save one (or a set) of current solutions, NOT paths back to the start state.
- Time complexity varies, though recent work indicates major improvements over incremental search for problems with densely-packed optimal solutions.
- Satisficing - can often find reasonably good solutions quickly.
- Requires representations that are easy to tweak to generate search-space neighbors.
- Uses an objective function to evaluate solutions. Similar to a heuristic but for complete solutions → less guesswork.
- Often portrayed as movement in a landscape.
The Basic Local-Search Algorithm

Begin with a set or population (P) of *individuals/solutions.*

REPEAT

- Use an objective function (F) to evaluate each individual in P.
- If an individual $p^* \in P$ produces an optimal value for F, return $p^*$ and halt.
- Produce *children* of solutions in P.
- Update P by including some or all children and removing some or all parent solutions in P.

UNTIL a pre-defined maximum number of iterations have been performed.

Return the best (though not necessarily optimal) solution found.

Algorithms differ in how they *produce* and *prioritize* children.
The Egg Carton Problem

Put as many eggs as possible in the M x N carton, but never with more than K eggs along any horizontal, vertical or diagonal line. Typically, K = 2.
Local search...

- is more likely than incremental search to have a syntactic-semantic distinction.
- often generates constraint-violating states, so relies on a good objective function.
Search space states are the syntactic representations.

Neighbor-generating operators designed to perform simple syntactic modifications, e.g., flipping bits.

**Direct Representations:** syntactic \( \approx \) semantic representation and neighbor-generation takes semantics into account \( \rightarrow \) **intelligent** but more work.

**Indirect Representations:** Syntactic \( \neq \) semantic representation \( \rightarrow \) **dumb** neighbor-state generator - easy. All intelligence is in the objective function.
The (Local) Search Landscape

Critical Design Decisions

- Objective function defines the landscape; it maps semantics to a score.
- Representation-modifying operators define legal moves in landscape.
**Hill Climbing**

**General Properties**

- **Greedy**: Always moves to child states with immediate improvement (over parent).
- **Fast**: But only on smooth landscapes.
- **Easily Fooled**: Gets stuck on rough landscapes.
Simulated Annealing

General Properties

- **Stochastic**: Hill climbing with some random movement.
- **Temperature-controlled**: SA’s temperature parameter regulates stochasticity level: often high early in search, and gradually decreasing.
- **Agile**: Stochasticity allows traversal of chasms and plateaus.
Local Beam Search: K Parallel Searches

Step S

K = 3

Step S+1

Jump to a neighbor of another state

Keith L. Downing
Search: The Core of AI
Local Beam Search Properties

- Best K neighbors (children) chosen globally at each step, so some states may contribute 0 children, others 2 or more.
- Good search $\Rightarrow$ efficient distribution of resources $=$ the K child states.
- Randomness possible via **Stochastic Beam Search**, wherein some of the K child states are chosen stochastically, weighted by their evaluation scores.

* All of these properties indicate that the K parallel searches are **not** completely independent.
Evolutionary Algorithms

General Properties

- Stochastic Local Beam Search with (some of) the K children created via **recombination** (e.g., crossover of binary or real vectors).
- Recombination enables **long jumps** in the search landscape, and can merge the best of both parents...but can also be destructive.
- Stochasticity provided by **selection mechanisms** (Learn more about these in the Bio-Inspired AI class).
What a Computer Sees during Local Search

Evaluations

0.54

15, 22, 37, 99...

Am I at a local optimum?

0.89

18, 33, 22, 86...

Where's the global optimum?

0.22

43, 11, 66, 84

Representations

For real problems, it only sees points, not the whole landscape.
Incremental Search for Queens

Search: The Core of AI

Keith L. Downing
Local Search for Queens

Generate successors by randomly choosing a queen and moving it to a random new location.

Whereas incremental search can only solve $K$ queens for $K$ up to 20 or 25 (due to search-tree depth), local search works for $K$ in the millions!!
Example of the Minimal Spanning Tree (MST) problem.
Kruskal's and Prim's algorithms do this easy. It's not even considered search!
But the problem becomes much more complex if we can add intermediate nodes.

There is no omnipotent algorithm for STP that guarantees optimality (of the solution) and efficiency (of the algorithm). It is an NP-Complete problem.
Constructing Steiner Points

1. Find longest side of triangle ABC (AC).
2. Draw equilateral triangle AQC opposite to point B.
3. The intersection of line BQ and the circle that circumscribes AQC is the Steiner point (P).
4. If angle ABC \( \geq 120 \) degrees, then P ← B.
Max of Steiner points is $|V| - 2$; $V =$ original vertices (Bern, 1989).

But there are $T = \binom{|V|}{3} = \frac{|V|(|V|-1)(|V|-2)}{6}$ possible point triples.

$\Upsilon = \sum_{i=0}^{|V|-2} \binom{|V|}{i}$ - For each possible number ($i$) of Steiner points, from 0 to $|V| - 2$, there are $\binom{T}{i}$ ways to pick them.

| $|V|$  | $\Upsilon$               |
|-------|--------------------------|
| 3     | 2                        |
| 4     | 11                       |
| 5     | 176                      |
| 10    | 903601306070             |
| 20    | 1466796429263525233849320188625720221593 |

$8 \times 10^{18} \approx \text{Number of sand grains on earth.}$
Node-expansion ops: a) add the next 1, 2 or 3 edges using Kruskal’s algorithm, and b) add 1 or 2 new Steiner points (drawn as stars).

Vertical bar = relative arc cost.

Horizontal triangle = relative heuristic (h) value: estimated dist to goal.
Best-First Search Solutions

Length of MST of 6 original points = 24. Bottom left tree is optimal.
Representations in Local Search

**Syntactic Representation**

\[
\{ [f^1, p_1^1, p_2^1, p_3^1], [f^2, p_1^2, p_2^2, p_3^2], \ldots, [f^k, p_1^k, p_2^k, p_3^k] \}
\]

- Encodes a list of point triples = bases for Steiner points.
- \( p^i \) = index of a point.
- \( f^i \) = flag; Only when flag =1 is steiner point created.
- Steiner-point indices can also appear in triples.
- Search occurs in the space of possible syntactic representations.

**Semantic Representation**

All points (original + Steiner) and edges (generated by Kruskal’s or Prim’s algorithm).

- This has *meaning* (i.e. semantics) in the problem domain.
- The evaluations of semantic representations govern search in syntactic space.
Development in Local STP Search

Kruskal's Algorithm:

\[ F = \frac{24.00 - 22.66}{24.00} \]
STP Search Landscape

Search: The Core of AI
STP Solutions using Local Search

Using an evolutionary algorithm with 20 individuals and run for 200 generations.
Steiner Brains (Cherniak et. al., 1999)

Constraints of neural development produce volume-minimizing Steiner trees.

Top: A Steiner tree for transporting material from A to B and C.
Middle: Thicker conduits require smaller branch angles → turbulence ↓.
Bottom: A network that minimizes edge volume.
Intelligent generators in Classic AI (GOFAI), but not in nature nor in Bio-AI.
Complex Deep Learning problems require MANY layers of neurons and LOTS of data!!

Gradient Descent

\[
\text{Gradient (G)} = \frac{\Delta \text{Error}}{\Delta \text{Weight}}
\]

\begin{itemize}
  \item Each state in search space = complete set of weights + biases.
  \item Greedy locally (each \( w_i \) changes to reduce \( E \)) but not globally. Can often escape local optima.
\end{itemize}
The essence of gradient descent learning (for simple and complex nets) is this basic update for EVERY weight $w_i$ in the network:

$$\triangle w_i = -\eta \frac{\partial E}{\partial w_i}$$

where $\eta$ = learning rate.

Calculating these partial derivatives (a.k.a. gradients) just gets more complicated for large networks with diverse layers.

ML packages such as Tensorflow, PyTorch and JAX automate this!!
The basic backpropagation algorithm and most of its variants permit the calculation of $\frac{\partial E}{\partial w_i}$ across any number of layers.

So all weights and biases can be updated in a **semi-intelligent, goal-directed** manner based on the **semantics** of the network, but not of the problem domain.
Even Transformers Use Gradients

Standard Transformers have 2 copies of the weight matrix and RELU, in series, in both encoder and decoder modules.
Reinforcement Learning (RL)

- **Value Function (of a Critic):** mapping: state $\rightarrow$ evaluation. Similar to an objective function, where evaluation = total expected reward from the state to a goal state.
- **Policy (in an Actor):** mapping: state $\rightarrow$ action.
- **Value functions and policies typically adapt together, with each depending upon the other for intelligent updates.**
- **RL Search = combination of exploration and exploitation:** intelligent trial and error.
RL’s Incremental Search

Basic Search Space

- Model = mapping: (state, action) → state.
- In Dynamic Programming, the model is given. In many other RL approaches, it must be learned by experience.
Model-free: The mapping \(((\text{state, action}) \rightarrow \text{state})\) is learned by exploration of state space.

**Bootstrapping**: Evaluations of states are updated based on the evaluations of their child, grandchild, etc. states.
Bootstrapping in Temporal Difference (TD) Learning

$$V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)]$$

- $\alpha$ = learning rate,
- $\gamma$ = discount factor,
- $r$ = reward after moving from state $s$ to $s'$
- $\delta = r + \gamma V(s') - V(s)$ = TD Error = Level of *Surprise*

This is 1-step TD.
Explosion of possible states → tables are impossible to build.

Neural net = a function approximator (*funcapp*) from states to evaluations (critic) or states to actions (actor).

The *funcapp* must handle every possible state as input.

**Key generality assumption:** For the vast majority of cases, similar states map to similar values.
TD-Gammon (Tesauro, 1995)

$S_t$ = state of the board

$$V(S_t) = r^* + V(S^*_{t+1})$$

$$\delta_t = r^* + V(S^*_{t+1}) - V(S_t)$$

Do move $a^*$

Learn
The AlphaGo Suite: Descendents of TD-Gammon

Deep Reinforcement Learning (DRL): It is now common to combine RL and neural networks, where the NN serves as actor and/or critic.
Alternatives for Evaluating Search States

How should an AI search algorithm evaluate this node?

Keep searching (expanding nodes) and hope to reach goal states (★)

Apply a heuristic

Monte Carlo Simulations (Depth-first trips to goal states)

Supplement with a neural network to evaluate all states at this level, or possibly anywhere in the search tree, depending upon the scope of the training cases.

* Monte Carlo methods are used in AlphaGo, since no heuristics are good enough.

Keith L. Downing
Search: The Core of AI
Search is an essential aspect of AI algorithms.

This involves generating and testing alternative hypotheses often represented as states in a search space.

Testing and getting feedback always has some connection to the problem domain, such as knowledge, constraint, reward, etc.

But generation can be anything from purely random to tightly focused, goal-directed and seemingly intelligent.

The syntactic-semantic distinction is normally only discussed in the context of local search, typically evolutionary algorithms.

Incremental search such as A* and RL work directly on the semantic representations.

However, when feeding a state into a neural network, we typically transform it into a more syntactic representation.
State-generating operators that work on syntactic representations typically cannot incorporate much domain knowledge and thus do not appear very intelligent.

Operators working on semantic representations can incorporate domain knowledge and can appear more intelligent.

Neural networks trained via backpropagation do semi-intelligent search, but not using domain knowledge. They act in a goal-directed manner to reduce error via gradients. This contrasts with NN’s trained via evolutionary algorithms, which generate weight sets randomly.

Intelligence may include randomness: **exploration + exploitation**.

For example, Simulated Annealing (stochastic - exploratory) is much more useful than Hill Climbing (greedy - exploiting).

Heuristics are often very domain-oriented, encapsulating human expertise (intelligence); but not all domains support useful heuristics. AI search in these domains requires a good dose of exploration.