Predicting Search Graph Size with Domain Abstractions

Presentation by Sean Dobson

May 20, 2016

General Problem

- The running time of a search algorithm is dependant on a number of factors:
  - Domain, Instance, Heuristic, Representation, etc.
- Some of these can be changed by the planner.
- How do we decide on the best combination of inputs?
- A method for predicting search runtime is needed.
- Estimating memory usage would also be nice.

Specific Problem

- Almost all of the work in this field focuses on Tree Search (IDA*)
- But how do we account for the Duplicate Detection done by Graph Search? (A*)
- Specifically, we want to predict the number of nodes expanded.
  - Pruning duplicates can greatly reduce the size of the Search Graph.
- For the moment, we will focus on predicting the size of a BFS Search Graph, bounded by a f^*.
- Hopefully the approach will extend to Heuristic-Guided Search.

Background and Related Work

- Domain Abstractions
- Type Systems
- KRE
- Stratified Sampling
- Stratified Sampling with Duplicate Detection

Domain Abstractions

- Domain Abstractions are a method of generating a state space homomorphism (an abstracted state space).
- In PSVN, we define a domain abstraction as a mapping, $\phi : L \to K$ where $|K| \leq |L|$
- Eg. $\phi : \{1,2,3,4\} \to \{1,2\}$:

$$\phi(x) = \begin{cases} x, & \text{if } x \in \{1,2\} \\ 2, & \text{if } x \in \{3,4\} \end{cases}$$

  So if $s = (1,1,2,3,4,2)$ then $\phi(s) = (1,1,2,2,2,2)$.
- We can apply this mapping to a PSVN problem in order to induce a state space homomorphism.

Type Systems

- Given a state-space, S
- And a set of types, T.
- We define a Type System for S as a function $t : S \to T$
- Example Type System for the 8-Tile Puzzle based on the position of the blank:

$$\begin{bmatrix} c & s & c \\ s & m & s \\ c & s & c \end{bmatrix}, \quad t\left(\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & B \end{bmatrix}\right) = c$$

- Yields the system of recurrance relations:

$$\begin{aligned} w_{c,d} &= 2w_{s,d-1} \\ w_{s,d} &= 2w_{c,d-1} + 4w_{m,d-1} \\ w_{m,d} &= w_{s,d-1} \end{aligned}$$

Where $w_{x,d}$ is the number of nodes of type x at depth d in the search tree.

KRE

- Pre-requisite: We can compute the number of nodes at depth i of the Blind Search Tree, $N_i(s)$, via the recurrance relation given by a perfect type system (not very general).
- Note that with a consistent heuristic, IDA* with threshold d will expand every node n such that $f(n) \leq d$.
- Let $P(s, i, d)$ be the percentage of nodes at depth i with $f(n) \leq d$.
- Then the number of nodes expaned at depth i by IDA* with threshold d is:
  $N_i(s)P(s, i, d)$.
- P is unknown, KRE attempts to approximate it.

KRE (cont.)

- Overall distribution: $D(v) =$ The probability that a state randomly chosen from the state-space has an h-value $\leq v$.
- Calculated from the PDB that we used for h.
- Equillibrium distribution: $P_{EQ}(v) =$ The probability that a node chosen randomly from the BFS Tree has h-value $\leq v$.
- Calculated from the equillibrium frequency of each type at large depths, and D for each type.
- $P(s, i, d) \approx P_{EQ}(d - i)$.

Stratified Sampling

- At each depth, choose nodes as **representatives** for their type.
- If $(n, w) \in A[d]$, then n is the representative for $t(n)$ at depth d, and we predict that there are w nodes of type $t(n)$ at depth d.
- There is no other $(n', w') \in A[d]$ such that $t(n) = t(n')$.
- For a given start state, s, and f-bound, f^*:
- Initialise $A[0] = \{(s, 1)\}$.
- For each pair $(n, w) \in A[d]$.
  - If $f(n) \leq f^*$, expand n to get its children.
  - For each child c, check if there is already some $(c', w') \in A[d+1]$ such that $t(c) = t(c')$.
    - If there isn't, update $A[d+1] := A[d+1] \cup \{(c, w)\}$.
    - If there is, update $w' := w' + w$, and with probability $\frac{w}{w'}$ set $c' := c$.

Stratified Sampling (cont)

- Running this procedure once constitutes a single "probe".
- A probe predicts the number of nodes expanded by a Tree Search bounded by f^*:

$$\sum_{d=0}^{f^*} \sum_{(n,w)\in A[d]} w$$

- The accuracy of these predictions depends on the assumption that nodes of the same type at the same depth have the same size subtree.
  - Strong assumption, but it seems alright for homogenous spaces (branching factor is constant).
  - Including the h-value of a node in your type system is a good idea (h-value estimates how deep the subtree will go).
  - In cases where it doesn't hold, we can run thousands of probes and average the results.

Stratified Sampling with Duplicate Detection

- Given a node, n, and the path, $\pi(n)$, that we used to get from s to n. How can we check if n is a duplicate?
- If our heuristic is consistent, then the first time A* expands a node, we are guaranteed to have found the shortest path to it, and that node will never be reopened.
- Then A* considers n to be a duplicate if and only if $\pi(n)$ is not an optimal path to n.
- Sampling-Based Duplicate Detection (SDD): Do k random walks backwards from n, if any random walk intersects $\pi(n)$ and gives a shortcut, then we know n is a duplicate.
- As $k \to \infty$, SDD will determine duplicates with 100% accuracy.
- Stratified Sampling with Duplicate Detection (SSDD): Like SS but we only expand representatives if SDD doesn't detect them as duplicates.

Hypotheses and Claims

- Predicting the Tile Puzzle's Search Tree Size
- Predicting the Duplicate Probability Distribution
- Predicting the Tile Puzzle's Search Graph Size

Predicting the Tile Puzzle's Search Tree Size

- H_1: The Type System for the Tile Puzzle can be used with SS to give excellent estimations for the BFS Search Tree Size (thus minimising a cause for error that is irrelevant to my research).
- When restricted to a perfect type system (i.e. one in which SS's key assumption holds), the choice of representative doesn't affect the prediction.
- SS acts as a bottom-up solver for the recursion.
- $H_{1,1}$: The Tile Puzzle type system only gives perfect predictions for Blind Search. Heuristic search requires a type system that also accounts for h-value.
  - Consider the case where $t(n) = t(n')$, and $g(n) = g(n')$, but $h(n) << h(n')$.
  - A* would probably expand many more descendants of n.
  - Perhaps a combination of type systems: $t_h(n) = (t(n), h(n))$.
  - Then if $t_h(s) = t_h(s')$, we have $t(s) = t(s')$ and $h(s) = h(s')$

Predicting the Duplicate Probability Distribution

- H_2: We can use Domain Abstractions to predict the probability that a node of a given type, and depth in the Search Tree, will be a duplicate.
- We randomly generate some n-value Domain abstraction, a, and apply it to our problem.
- Run BFS on the abstracted problem.
- Find $P_a(k,d) = \frac{expanded_a(k,d)}{generated_a(k,d)}$ = the probability that a node of type k, generated at depth d of the abstracted Search Graph, is a duplicate.
- Predict $P(k,d) \approx P_a(k, d\frac{f_a^*}{f^*})$, the equivalent probability for the actual problem.
- $H_{2,1}$: Increasing n will decrease the accuracy of our prediction (also decreases runtime).
- $H_{2,2}$: We can increase accuracy by aggregating over multiple abstractions (also increases runtime).

Predicting the Tile Puzzle's Search Graph Size

- H_3: We can extend Stratified Sampling (SS) to use the Duplicate Probability Distribution, thus providing predictions for the size of the Search Graph.
- SSDP: Like SS, but before we expand the representatives $(n, w) \in A[d]$, we update $(n, w) := (n, wP(t(n), d))$
- $H_{3,1}$: A more accurate estimation for the P will increase the accuracy of the SS predictions.
- $H_{3,2}$: We can also increase the accuracy by aggregating over multiple probes.
- $H_{3,3}$: We can compare this method with SSDD.

Research Plan

- Literature Review: Ongoing
- Implement Experiments: Ongoing
  - Get SSDD working: Next.
  - Perform Levi's experiments: Week 1 of inter-semester break
  - Implement algorithm for predicting Duplicate Probability Distribution: Weeks 1-2 of break
  - Implement SSDP: Weeks 2-3 of break.
  - Perform Experiments: Week 1,2 of Sem 2.
- Write Experiment Analysis: Weeks 3,4.
- Write Background: Weeks 5,6.
- Write Conclusion: Week 7.
- Write Introduction, Abstract: Week 8.
- Formatting, Reference compilation: Week 9.