Estimating Search Tree Size with Duplicate Detection

Levi H.S. Lelis, Roni Stern, Nathan R. Sturtevant (2014)

Presentation by Sean Dobson

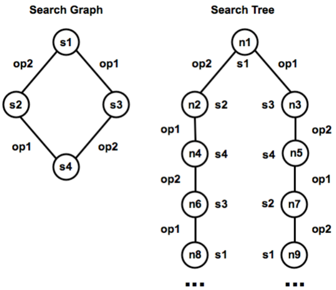
March 19, 2017

General Problem

- Predict the run time of a search algorithm (quickly).
- Think about the assignment we did:
  - Which heuristic should we choose?
  - How can we tell, prior to running the searches, which will be the fastest?
  - Any gain from choosing the best search could be outweighed by the extra time taken to predict which search is best.

Specific Problem

- Predict the number of nodes expanded by the search.
  - Not the only factor that decides search run-time.
- Account for search algorithms that use duplicate detection (Graph Search).
  - Pruning duplicates can greatly reduce the number of nodes expanded.
  - But it adds an extra layer of complexity to the problem.
  - How do we predict the number of nodes pruned by duplicate detection?



Claims

- Stratified Sampling (SS) can be used to predict the number of nodes expanded by Tree Search.
  - The algorithm is non-deterministic.
  - Run it multiple times, and then average (or max) over the outputs.
  - As the number of 'probes' approaches infinity, the prediction converges to the true value.
- Sampling-based Duplicate Detection (SDD) can determine if a node is a duplicate.
  - Similar reasoning to above.

Claims (cont.)

- Stratified Sampling with Duplicate Detection (SSDD), can predict the number of states expanded by Graph Search.
- With a little extra work, we can easily account for nodes pruned by A*'s heuristic.
- Another variation of the algorithm can be used to predict the state-space radius

Approach: Type Systems

- Many-to-one mapping from states to types (a partition of the state space).
- An example type system for the 8 Tile Puzzle, based on the position of the blank:

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

- Note that expanding a type c will always generate two type s nodes, expanding an s will generate two c + one m, and expanding an m will generate four s.

Example: Predicting 8-Puzzle Search Tree Size

- $w_{c,d+1} = 2w_{s,d}$
  $w_{s,d+1} = 2w_{c,d} + 4w_{m,d}$
  $w_{m,d+1} = w_{s,d}$

	0	1	2	3	4	...
c	1	0	$2 \times 2 = 4$	0	$16 \times 2 = 32$	...
s	0	2	0	$4 \times 2 + 2 \times 4 = 16$	0	...
m	0	0	$2 \times 1 = 2$	0	$16 \times 1 = 16$	...

- This is only possible because we knew the exact number and type of the children generated.
- Not all type systems are perfect like this one.
- How do we generalise this approach?
- At what depth do we stop?

Stratified Sampling

- $A[d]$ is the set of Representative-Weight pairs for depth d.
- 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.
- Start with $A[0] = \{(s, 1)\}$.
- For each pair $(n, w) \in A[d]$.
  - Expand the state 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 $c' := c$.
- Problem: SS assumes that nodes of the same type at the same depth will generate the same size subtree. This isn't true with a poor type system
- Solution: Run thousands of probes and average the results.

Sampling-Based 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?
- Note that if our heuristic is consitent, A* would consider n to be a duplicate if and only if there exists a path from s to n that costs less than $\pi(n)$.
- We could run BFS backwards from n and check for any 'shortcut' path to n from some $n' \in \pi(n)$.
- Problem: BFS is expensive.
- Solution: Do a random walk backwards from n, if at any point it intersects $\pi(n)$ and gives a shortcut, then we know n is a duplicate.
- Problem: A random walk probably isn't very likely to find a shortcut.
- Solution: Do thousands of random walks.

Stratified Sampling with Duplicate Detection

- SSDD works almost the same as SS.
- The paths that we used to produce representative nodes in an SS probe are not guaranteed to be optimal.
- Some representatives might actually be duplicate nodes.
- The search that we are trying to predict uses duplicate-detection so that it only expands non-duplicates.
- In SSDD, before we expand a representative, we run a bunch of SDD random walks on it to check if its a duplicate.
- If it is, then we simply don't expand it.

Evaluation

- They evaluate SSDD in two contexts:
  - Predicting A*'s search tree size: We only expand non-duplicate representatives with an f-level less than or equal to the optimal solution cost.
  - Predicting the search-radius: Run SSDD until we reach an f-level where every node is detected as a duplicate. (As an aside: Could we also use this to predict the size of the reachable state-space?)

A*

(17,4)-Topspin														
SS					**SSDD** ($m = 1$)					**Parallel SSDD** ($m = 100$)				
p	mean.	median	sign.	%	k	mean.	median	sign.	%	k	mean.	median	sign.	%
3,000	873.97	1,329.21	1,065.07	8.55	3,000	2.62	0.95	2.71	1.76	3,000	1.52	1.03	2.58	3.68
5,000	873.58	1,406.36	1,065.54	14.07	7,000	2.72	0.96	2.35	3.72	7,000	1.00	0.50	1.97	8.08
6,000	873.75	1,461.14	1,065.42	17.18	10,000	2.07	0.96	1.89	5.15	10,000	0.84	0.41	1.77	11.39
7,000	873.61	1,398.75	1,065.4	19.68	11,000	1.90	0.95	1.59	5.59	11,000	0.86	0.44	1.77	12.45
12-disks 4-pegs Towers of Hanoi														
SS					**SSDD** ($m = 1$)					**Parallel SSDD** ($m = 100$)				
p	mean.	median	sign.	%	k	mean.	median	sign.	%	k	mean.	median	sign.	%
300	4.00e+36	1.13e+40	8.37e+36	15.50	3,000	6,146.39	0.99	7,817.65	4.32	3,000	2.25	0.95	1.54	6.03
400	4.05e+36	1.16e+40	8.48e+36	20.65	7,000	185.45	0.99	234.70	9.01	7,000	1.07	0.97	0.26	12.35
500	3.83e+36	9.61e+39	8.03e+36	25.76	10,000	98.63	0.99	65.74	12.55	10,000	1.57	0.98	0.64	17.28
1,000	3.97e+36	1.08e+40	8.31e+36	51.89	11,000	52.91	0.99	41.55	13.74	11,000	1.74	0.98	0.80	18.59
15-Pancake														
SS					**SSDD** ($m = 1$)					**Parallel SSDD** ($m = 100$)				
p	mean.	median	sign.	%	k	mean.	median	sign.	%	k	mean.	median	sign.	%
3,000	7.74	7.80	10.27	12.09	3,000	7.92	2.94	12.36	4.70	3,000	6.07	5.44	8.35	4.89
5,000	7.76	7.54	10.27	19.79	7,000	6.18	2.35	8.63	10.80	7,000	5.41	4.90	7.55	11.30
6,000	7.75	7.64	10.29	23.22	10,000	5.22	2.20	6.80	15.50	10,000	5.31	4.73	7.33	16.12
7,000	7.79	7.43	10.34	23.09	11,000	5.33	2.02	7.09	16.98	11,000	5.05	4.54	7.03	17.75
20-Gripper														
SS					**SSDD** ($m = 1$)					**Parallel SSDD** ($m = 100$)				
p	mean.	median	sign.	%	k	mean.	median	sign.	%	k	mean.	median	sign.	%
3,000	4.88e+12	5.50e+12	1.62e+13	12.80	3,000	654,236.54	1.00	672,015.38	0.86	3,000	308,132.55	93,446.62	315,641.81	2.17
5,000	4.91e+12	5.37e+12	1.63e+13	21.20	5,000	184,995.55	0.99	199,937.87	1.08	5,000	71,951.11	13,132.61	74,350.97	3.09
7,000	5.06e+12	5.64e+12	1.68e+13	29.58	100,000	1,036.63	0.99	1,168.51	8.06	100,000	1.53	0.90	1.11	20.33
8,000	4.93e+12	5.26e+12	1.63e+13	33.76	140,000	6.95	0.99	5.80	10.35	140,000	1.24	0.93	0.64	26.98

Observations and Analyses

- SS is the worst in almost every case, increasing the number of probes doesn't help that much.
  - SS doesn't account for duplicates, so it has a systematic bias toward overprediction.
- SSDD is run with only 1 probe, but does much better than SS
  - For each representative expanded, they run thousands of SDD walks.
  - By not expanding duplicates, SSDD mitigates SS's bias toward overprediction.
- The mean prediction for SSDD is always greater than the median
  - The predictions are unevenly distributed.
  - A small number of SSDD probes will fail to detect duplicates, and give extremely large overestimations.
- Parallel SSDD's mean is similar to it's median.
  - Parallel SSDD runs 100 probes in parallel, and stops when the first 95% have completed.
  - Thus cutting off the problematic 'tail' of the prediction distribution.

Search-radius

MAX Scheme

3x3x3 Rubik's Cube Radius = 20

m / k	50		1,000		2,000		3,000		4,000		5,000	
	prediction	time	prediction	time	prediction	time	prediction	time	prediction	time	prediction	time
5	14.8 ± 7.5	0.0	8.7 ± 4.6	0.0	10.0 ± 5.0	0.0	9.0 ± 4.1	0.0	8.2 ± 3.7	0.0	7.8 ± 3.8	0.0
10	18.1 ± 7.5	0.0	10.7 ± 4.3	0.0	11.5 ± 5.1	0.0	11.2 ± 4.6	0.0	11.2 ± 5.1	0.0	11.5 ± 4.8	0.0
20	23.1 ± 8.2	0.0	13.7 ± 4.5	0.0	15.3 ± 5.0	0.0	14.3 ± 4.8	0.0	14.9 ± 5.0	0.0	15.3 ± 5.0	0.0
30	26.3 ± 7.9	0.0	16.4 ± 4.9	0.0	16.8 ± 5.4	0.0	17.4 ± 5.3	0.0	17.4 ± 5.6	0.1	17.2 ± 5.6	0.1
40	28.3 ± 8.3	0.0	18.1 ± 5.3	0.0	18.5 ± 4.9	0.0	18.4 ± 5.2	0.1	18.5 ± 5.2	0.1	18.7 ± 5.9	0.1

4x4 Sliding-Tile Puzzle Radius = 80

m / k	50		1,000		2,000		3,000		4,000		5,000	
	prediction	time	prediction	time	prediction	time	prediction	time	prediction	time	prediction	time
5	132.9 ± 67.4	0.1	66.8 ± 32.9	0.3	57.1 ± 26.0	0.4	54.4 ± 23.9	0.5	52.9 ± 22.5	0.6	52.6 ± 22.4	0.7
10	165.4 ± 71.9	0.2	82.5 ± 34.6	0.6	70.2 ± 26.2	0.7	65.6 ± 23.8	0.9	68.8 ± 25.6	1.3	63.8 ± 24.4	1.3
20	203.8 ± 68.7	0.4	95.5 ± 31.4	1.0	87.9 ± 27.4	1.5	78.8 ± 23.1	1.8	77.5 ± 22.9	2.2	75.5 ± 24.1	2.7
30	228.6 ± 71.2	0.6	107.9 ± 33.4	1.6	97.7 ± 28.1	2.4	91.9 ± 27.1	3.0	86.6 ± 24.1	3.5	82.9 ± 21.6	3.9
40	243.6 ± 74.2	0.9	112.6 ± 27.5	2.0	101.3 ± 27.1	3.0	93.3 ± 24.3	3.7	88.1 ± 21.2	4.3	86.1 ± 18.6	5.0

AVG Scheme

3×3×3 Rubik's Cube Radius = 20

m / k	50		1,000		2,000		3,000		4,000		5,000	
	prediction	time	prediction	time	prediction	time	prediction	time	prediction	time	prediction	time
5	14.8 ± 2.1	0.0	8.7 ± 1.0	0.0	10.0 ± 1.2	0.0	9.1 ± 0.8	0.0	8.0 ± 0.6	0.0	7.7 ± 0.5	0.0
10	18.1 ± 2.4	0.0	10.7 ± 1.1	0.0	11.6 ± 1.1	0.0	11.2 ± 0.7	0.1	11.2 ± 1.1	0.2	11.5 ± 1.4	0.2
20	23.1 ± 2.4	0.0	13.6 ± 0.9	0.0	15.2 ± 1.0	0.2	14.3 ± 0.9	0.3	14.8 ± 1.1	0.6	15.5 ± 0.9	0.8
30	26.3 ± 2.4	0.0	16.5 ± 1.0	0.1	16.9 ± 1.2	0.4	17.4 ± 1.2	0.8	17.4 ± 1.4	1.1	17.4 ± 1.3	1.3
40	28.3 ± 2.3	0.0	18.1 ± 1.1	0.2	18.5 ± 0.7	0.7	18.5 ± 1.2	1.0	18.6 ± 1.3	1.5	18.8 ± 1.1	1.9

4×4 Sliding-Tile Puzzle Radius = 80

m / k	50		1,000		2,000		3,000		4,000		5,000	
	prediction	time	prediction	time	prediction	time	prediction	time	prediction	time	prediction	time
5	132.9 ± 15.8	2.0	66.8 ± 7.7	5.7	57.1 ± 5.3	7.1	54.4 ± 6.3	9.3	52.9 ± 5.1	11.3	52.6 ± 5.8	13.8
10	165.4 ± 16.6	4.0	82.5 ± 6.1	11.4	70.2 ± 6.2	14.0	65.6 ± 6.6	17.5	68.8 ± 6.1	25.9	63.8 ± 4.7	26.5
20	203.8 ± 12.9	8.2	95.5 ± 6.7	19.9	87.9 ± 6.5	30.8	78.9 ± 6.1	36.2	77.7 ± 5.2	45.2	75.7 ± 5.7	53.9
30	228.6 ± 15.8	12.9	108.0 ± 6.7	31.1	97.9 ± 8.3	48.0	92.2 ± 5.4	60.8	86.4 ± 4.4	68.7	82.6 ± 5.0	76.8
40	243.6 ± 15.2	17.4	112.6 ± 5.7	39.7	101.2 ± 6.5	59.7	93.4 ± 5.8	73.5	88.2 ± 4.9	86.5	86.1 ± 5.5	99.8

Observations and Analyses

- Increasing the number of probes increases the predicted value
  - We MAX over all probes
- Increasing the number of SDD walks decreases the predicted value
  - We only stop when every new representative is detected as a duplicate.
- Increasing both gets us closer to the true value.
  - Assuming we ran an infinite number of SDD walks so that we predicted duplicates with 100% accuracy, then none of the probes would exceed the radius.
  - If we then did an infinite number of random SSDD probes, eventually one of the probes will produce the actual path of the maximum optimal solution.
- AVG reduces the variance of the predictions, but it is slower.
  - AVG runs MAX a bunch of times, and then averages the results.

Future Research

- There are other runtime prediction methods that could be enhanced by SSDD.
  - Korf, Reid, and Edelkamp's KRE predicts IDA*.
  - Zahavi et. al's Conditional Distribution Prediction (CDP) is an improvement of KRE.
- Is there some better alternative to SDD?
  - We don't want to do thousands of random walks.
  - Move pruning is a technique that eliminates redundant sequences of operators.
  - If we knew the probability of nodes of each type being a duplicate, could we use that to inform SS? (my Honours dissertation)
- How can we generate a good type system automatically?
  - They use $t(n) = h(n)$ for A* and $t(n) = |\pi(n)|$ for radius.
  - Combine different type systems?
    $t_a(n) = x,\ t_b(n) = y,\ t_{ab}(n) = xy$
- Can we predict the size of the reachable state space with SSDD?
  - This could be used in conjunction with an abstract (PDB) search in order to estimate the size of the actual search. (find the 'Space Compression Factor')

Discussion

- How important is the problem being addressed?
- How significant are the claims?
- How convincing is the support for these claims?
- How general is this approach?
- Does this work/approach generalise to other problems?
- Does this work lead on to further research?