Predict the Number of Nodes Expanded by A*

Marvin Abisrror
Departamento de Informática
Universidade Federal de Vicosa
Viçosa, Brazil

Abstract Predict the time an algorithm will last to find a solution or which heuristic is better to use for an specific domain problem are questions we do when we start to planning the route to get the solution of the problem. There are approaches of predicting such as KRE (Korf et al., 2001) where use the unconditional heuristic distribution to guide the prediction, others that use the conditional heuristic distribution, and other assumptions. In this paper we propose a new approach to predict the number of nodes expanded by A*, our approach has two main stages which are: In the first stage we use the tree generated by Dijkstra algorithm to collect the number of nodes expanded by level. In the second stage we use Stratified Sampling (SS) to generate the distribution of the f-value by level. With these to stages completed for the problem we make predictions based on a mathematical formulas proposed here. We use the optimal problem domains from fast-downward planning system to applied our approach and the results are not precise so far, but we are working on improving it.

Key words: Prediction; Artificial Intelligence; A*

1 Introduction

Informed algorithms such as A* (Hart and Raphael, 1968) is very important in AI because guide in the resolution of interesting applications. This important algorithm have two elements: The heuristic value from a state s give us an estimate of the remaining goal distance named as h. Intuitively, nodes with lower heuristic value are more promising than nodes with higher heuristic value because they seem to be closer to the solution. And the cost to go from the origin state to the current state named as g. The sum of these elements $f = h + g$ is called cost function, help us to prioritize node expansion during the search chosing the nodes that generate a close approximation to the solution path.

Dijkstra search Algorithm generate a SPT(shortest path tree) with given source s as root. Dijkstra is a greedy algorithm where we maintain two sets, one set contains nodes included in shortest path tree, other set includes nodes not yet included in shortest path tree. At every iteration of the algorithm, we find a node which is in the other set (not yet included) and has minimum distance from the start state and is added to the shortest path tree set. The heuristic value when Dijkstra is used is 0 for all the nodes because the guide to get the solution is the cost to go from the parent node to their child node, so dijkstra is not considered as informed algorithm, for our purpose we will use the information of the nodes generated at each level by Dijkstra to make the prediction.
Stratification Sampling is applied to the problem to predict the number of nodes expanded in the tree. **SS** works iteratively expanding without counting nodes that represent the same state because it chose randomly one of the repeated node that have more chance to be used and removing or avoiding the insertion of the other. This behavior does that **SS** overestimates the real number of nodes of the tree and generate what we call in this paper the brute force search (BFS). We collect the nodes with *f-value* less than or equal to the threshold ***d*** which is equal to two times the heuristic value of the initial state:

$$d = 2 \times (s*)$$

In each level of the BFS we chose the nodes that meets the condition in order to generate our distribution. We build our distribution based on **SS** because (Lelis et al., 2013) have shown that **SS** produce much more accurate predictions of the search tree than competing schemes such as CDP (Zahavi et al., 2010).
The formula to predict the number of nodes expanded by A* are related with the information from Dijkstra Algorithm and the distribution of *f-value* from SS.

2 Background

We use as experiments the competitions domains problems from the fast-downward benchmarks. The planner contains satisficing and optimization track problems, the satisficing problem are so hard to work and also is difficicult to found an optimal solutions, so they could be used to research because we pretend to predict the number of nodes expanded in the tree but we won't found an optimal solutions and also it would skip our analysis using consistent heuristics. In this research we will use the problems that require optimization in order to test our approach.

2.1 Problem Domains

Two of the optimal domain problems we have worked are barman and parking.

2.1.1 Barman

This domain was created for the *International Planning Competition (IPC)* in *2011*. In this domain there is a robot barman that manipulates drink dispensers, glasses and shaker. The goal is to find a plan of the robot's actions that serves a desired set of drinks.

2.1.2 Parking

This domain was created for learning track *IPC-2008*. This domain involves parking cars on a street with N curb locations, and where cars can be double-parked but not triple-parked. The goal is to find a plan to move from one configuration of parked cars to another configuration, by driving cars from one curb location to another.

2.1.3 Blocks World

This domain had origin in the *IPC-2000*. This domain consists of a set of blocks, a table and a robot hand. The blocks can be on top of other blocks or on the table; a block that has nothing on it is clear; and the robot hand can hold one block or be empty. The goal is to find a plan to move from one configuration of blocks to another. The problem probBLOCKS-4-0 is going to be our instance that will be used to explain our results in this paper. We chosen that because is a simple problem that can be resolved in less than few seconds and also contains the characteristics of duplicate nodes.

3 Problem Formulation

We define three graphs: The first one is the graph generated by dijkstra algorithm, which contains the tree without duplicate nodes. The second is the Brute Force Search tree generated by the Stratified Sampling algorithm which generate duplicate nodes and the last one is the tree generated by A* using a consistent heuristic which is also the tree we want to predict.

Formally, Let $G = (N, E)$ be a graph representing an expanded search graph (ESG) of each graph explained above where N is its set of states and for each:

$$n \in N / op(n) = op_i | (n, n_i) \in E$$

is its set of operators. We use the words edges and operators interchangeably. The prediction is to estimate the size of N without fully generate the ESG from A*.

4 Predicting the size of the Search Graph

We now explain how we pretend to achieve the prediction of the number of nodes expanded by A*.

4.1 Dijkstra Algorithm

Dijkstra Algorithm is an uninformed search algorithm. In our cost function the value of ***h*** is ***0*** for all the nodes of the search tree and the value ***g*** increase according to the level of the tree. It means that there is no guide to search the goal, in each iteratation the nodes from a level expand blindly according to the value of the level up to find the goal node. We collect the total number of nodes expanded by level without prunning. So, for our exemple in the level ***0*** we have one node which is initial state, in the level ***1*** we have ***4*** nodes, level ***2*** contains ***12*** nodes, etc. This information is collected in the planner executing A* using the dijkstra heuristic which assign the value zero for all the nodes in the search tree.

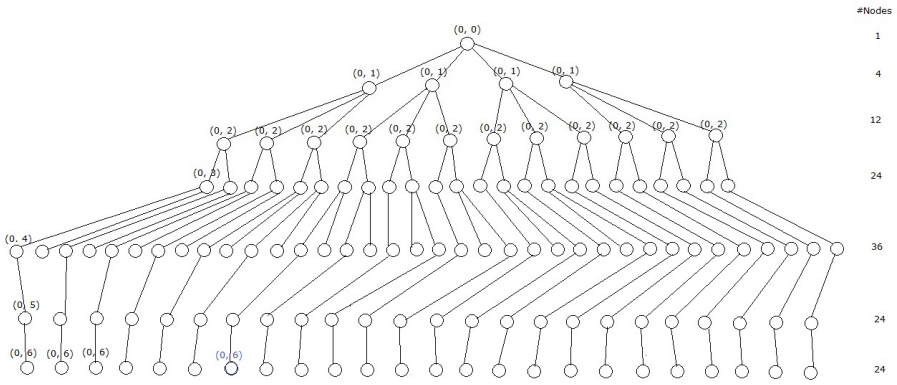

Fig. 1 The par value (***h***, ***g***) represent the heuristic value ***h*** and the cost to go from the start state to the current node ***g***. The tree generated by Dijkstra Search Algorithm to the problem of blocks worlds probBLOCKS-4-0.

In the Figure 1 are showed the total number of nodes expanded by level and also how the tree is distributed.

We also, present the Table 1 which explain the distribution of nodes expanded by level during dijkstra search algorithm.

Level	#Nodes expanded
0	1
1	4
2	12
3	24
4	36
5	24
6	24

Table 1: Number of nodes expanded by *Level* using Dijkstra algorithm.

4.2 f-value Distribution using Dijkstra and collecting f-value with consistent heuristic - Assumption

In order to keep the data consistent and validate the process of the prediction we make assumptions about what we expect to obtain, for example we make comparison between *f-value* distributions generated by two algorithms with the objective to see which algorithm is superior and fits fits well to our approach.

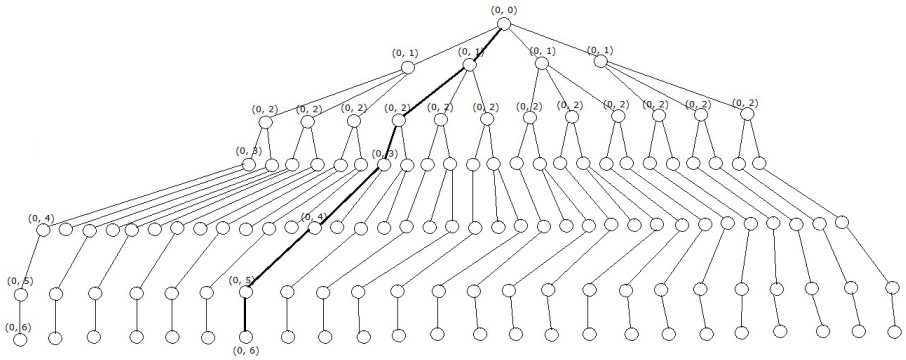

Fig. 2 The tree generated by Dijkstra Search Algorithm and the solution route expressed by bold solid line.

Starting with our assumptions we present the distribution of *f-values* in the tree generated by dijkstra, but collecting the *f-values* using a consistent heuristic.

The search tree generated by Dijkstra Search Algorithm expand each node blindly with heuristic value zero. In the Figure 2 the solid line represents the connection from the parent nodes to their childs nodes. Each node is represented with the par ***(h, g)*** which represent the heuristic value ***h*** and the cost to go from the origin node to the current node ***g***.

Dijkstra Algorithm iteratively expand each node in each level up to found the solution. The initial node is given by ***(0, 0)*** and is located at level ***0*** and the goal node is ***(0, 6)*** and is at level ***6***.

We expanded each node and collected the ***f-value*** based on a consistent heuristic ***h*** and the ***g*** value of the node. The bold solid line represent the solution of using a consistent heuristic in the search. The route solution is given by: ***(6, 0), (5, 1), (4, 2), (3, 3), (2, 4), (5, 1), (0, 6)***

We also present the table with the distribution of *f-value* during dijkstra algorithm execution.

Level	*f-value*	quantity
0	6	1
1	6	1
	8	3
2	6	1
	8	3
	10	8
3	6	1
	8	2
	10	6
	12	15
4	6	1
	12	2
	14	6
	16	15
5	6	1
	12	2
	14	6
	16	15
6	6	1
	14	2
	16	6
	18	15

Table 2: *f-value* distribution expanding the graph using dijkstra and collecting the *f-value* using a consistent heuristic.

The Table 3 represent the distribution of *f-value* in the tree search expanded by dijkstra and using a consistent heuristic to collect the *f-value*. The first column represent the level in which the node is located. The second column is the *f-value* and the third column is the quantity, which represent the number of nodes in the level that have the same *f-value*.

In order to make a prediction we will require the Formula (2) on page 10. $P_e x$ is the percentage of nodes expanded by level with *f-value* ***d*** less than to the threshold.

Level	Ni	$P_e x(s, d = 6, Level)$
0	1	1
1	4	0.250
2	12	0.083
3	24	0.041
4	36	0.027
5	24	0.041
6	24	0.041

Table 3: *f-value* distribution expanding the graph using dijkstra and collecting the *f-value* using a consistent heuristic.

Applying the Formula (1) on page 10 we obtain the prediction.

	threshold	#nodes expanded	Prediction
probBLOCKS-4-0	6	7	7
probBLOCKS-4-1	10	11	11
probBLOCKS-4-2	6	7	7
probBLOCKS-5-0	12	16	16
probBLOCKS-5-1	10	14	14
probBLOCKS-5-2	16	29	29
probBLOCKS-6-0	12	13	13
probBLOCKS-6-1	10	11	11
probBLOCKS-6-2	18	44	44
	19	97	97
	20	399	399
probBLOCKS-7-0	17	8	8
	18	30	30
	19	35	35
	20	243	243

Table 4: Prediction of the number of nodes expanded by A* for the first 10 blocks world domain problems.

The Table 4 contains four columns: In the first column we have the blocks world domain problems chosen from fast-downward benchmark. We chosen 10 problems, as well as the number of the problem increases it turns more complex in terms of duplicate and expansions nodes. The second column in the threshold which represent the branches where the nodes are expanded. The third column is the number of nodes expanded by A* in each threshold, and the last one column is the prediction.

The results in the Table 4 are 100% accurate, and this predictions are the expected, because we are testing with the same data the two terms of the prediction formula (1).

4.3 f-value Distribution using DFS - Assumption

We have an assumption that if the distribution of the *f-values* of the tree generated by dijkstra have similar structure to the distribution of the *f-values* generated by Depth First Search (DFS). It would mean that the prediction using the number of nodes expanded by level with dijkstra will generate accurate values conforme to the number of nodes by *f-value* generated by A*.

Our assumption about the similarity of the *f-value* distribution of Dijkstra and Depth First Search was tested with the same problem mentioned above obtaining the following behaviors: The *f-value* distribution of Dijkstra was quickly elaborated, three steps were necesary, first the nodes expansions, the collection of the *f-value* of the nodes and then the generation of the *f-value* distribution file. Nevertheless, DFS last approximately 5 seconds to expand all the nodes and generate the *f-value* which make sense because it expands 1873081 nodes. For other instances for example the probBLOCKS-5-0 just expand the nodes takes more than two hours.

So, try to generate the *f-value* distribution is expensive in computational time because expand nodes the way how DFS expand the nodes is a iterative task that could take much time. So we are going to present the distribution obtained from probBLOCKS-4-0. We found that the nodes ***(6, 0),***

(5, 1), (4, 2), (3, 3), (2, 4), (5, 1), (0, 6) were expanded in the same way as expanded by dijkstra algorithm, see the Table 3 on page 6. It means that the distribution of *f-value* equal to 6 in each level generated by DFS is the same as the *f-value* distribution with *f-value* equal to 6 in each level generated by Dijkstra.

Level	*f-value*	quantity
0	6	1
1	6	1
	8	3
2	6	1
	8	7
	10	8
3	6	1
	8	9
	10	27
	12	15
4	6	1
	8	11
	10	59
	12	88
	14	25
5	6	1
	8	12
	10	79
	12	239
	14	186
	16	15
6	6	1
	8	13
	10	103
	12	507
	14	832
	16	321
	18	15
7	8	14
	10	116
	12	689
	14	2095
	16	1824
	18	210
8	8	14
	10	130
	12	921
	14	4387
	16	7480
	18	3194
	20	210
9	10	144
	12	1051
	14	5997
	16	18279
	18	16629
	20	2160

10	10	144
	12	1195
	14	8099
	16	38063
	18	65944
	20	29299
	22	2160
11	12	1339
	14	9294
	16	52159
	18	159155
	20	147444
	22	20085
12	12	1339
	14	10633
	16	70747
	18	330615
	20	576668
	22	260469
	24	20085

Table 5: *f-value* distribution expanding the graph using depth first search and collecting the *f-value* using a consistent heuristic and threshold twice the initial heuristic value.

The Table 5 represent the distribution of *f-value* in the tree search expanded by depth first search using a consistent heuristic and threshold twice the initial heuristic value. Depth first search found seven nodes with *f-value* equal to six in the levels ***0, 1, 2, 3, 4, 5***, and ***6*** which are the same distribution of *f-value* equal to six in Table 3 on page 6.

This result encourage to continue with our research, because the distribution of dijkstra is more quickly to obtain and also make prediction over the distribution in comparison with the depth first search tree generated that is more expensive to obtain.

	Threshold	#nodes expanded	Prediction
probBLOCKS-4-0	6	7	3.4657
probBLOCKS-4-1	10	11	3.83765
probBLOCKS-4-2	6	7	3.47631
probBLOCKS-5-0	12	16	4.65024
probBLOCKS-5-1	10	14	5.34157
probBLOCKS-5-2	16	29	5.68251
probBLOCKS-6-0	12	13	3.96905
probBLOCKS-6-1	10	11	3.98732
probBLOCKS-6-2	18	44	6.79542
	19	97	10.9438
	20	399	56.65
probBLOCKS-7-0	17	8	3.8663
	18	30	6.49635
	19	35	15.8169
	20	243	45.0983

Table 6: Prediction of the number of nodes expanded by A* using the number of nodes by level from dijkstra and the *f-value* distribution obtained from DFS.

The Table 6 displays the results of apply the Formula (1) using two terms: The first one, the number of nodes expanded by dijkstra, and the other is the *f-value* distribution obtained from DFS. For the first row the terms are being obtained from the Table 1 and the Table 5.

5 Stratification Sampling

Knuth (1975) did experiments to predict the number of nodes expanded by IDA* under the assumption that all branches contained the same structure. He realized that the method was not effective when the branches were unbalanced. Chen (1992) addressed the problem with stratification of the search tree through a *type system*. He assumed that nodes of the same type at a level of the search tree would generate subtrees of the same size. Then, only one node of each type **SS** estimates the size of the tree.

The distribution the *f-values* is going to be obtained from the Brute Force Search, in this case the **SS**.

5.1 Type System

We require a property that allow us to represent each node in the state space search and this property can be defined using any information about the node. For example we could use a *type system* which counts how many children a node generates, or how many children the parent of the current node generates or the parent of the parent of the current node, the *f-value*, the ***g*** value or the heuristic value ***h*** of the node, etc.
This property is used in **SS** to distinguish the nodes or stratified the state space. In our research we present a *type system* that use the heuristic value of the node ***h*** and the level in which the node is located ***L***. We use the following *type system*:

$$T(s) = (h(s), L(s))$$

Two nodes at a level of the search tree will have the same *type system T* and randomly one of them will be chosen because the asumption is that both generate the same subtree. Then, the subtree not chosen is removed from the tree and the number of nodes in the chosen node is upated withh the sum of nodes of the subtree of the current node and the nodes of the subtree removed.

6 Prediction Formula

For a given state ***s*** and A* threshold ***d***, **N(s, d)** is the prediction of the number of nodes that A* will expand if it use ***s*** as its start state and does a complete search with an A* threshold of ***d***.

$$N(s,d) = \sum_{i=1}^{d} N_i(s,d) \tag{1}$$

Where *N(s, d)* is the number of nodes expanded by A* at level i when its threshold is d. One way to decompose $N_i(s,d)$ is as the product of two terms. (Zahavi et al., 2010)

$$N_i(s,d) = N_i(s) \times P_e x(s,d,i) \tag{2}$$

Where N_i is the number of nodes in level i of *BFS*, the brute-force search tree (*i.e.,* the tree created by dijkstra search without heuristic pruning.) of depth $\boldsymbol{d}$ rooted at start state $\boldsymbol{s}$, and $P_ex(s,d,i)$ is the percentage of nodes in level i of BFS(*i.e,* the distribution with threshold two times the heuristic of the initial state generated by the Stratified Sampling.) that are expanded by A* when its threshold is $\boldsymbol{d}$.

7 Conclusions

In this paper we presented a new approach to predict the number of nodes expanded by A*, we take as assumption that the expansion of the number of nodes by level that dijkstra generates can be usefull to our prediction because according to the behavior of the distribution of nodes that dijkstra give us we can extend it using linear regression. Another information we get from the *f-value* distribution that Stratified Sampling give us when search the solution of the problem. Joining both informations we can use a mathematical formula of prediction.

We are still working on produce good results. We started testing the differents assumptions in order to be sure about the approach we are proposing.

References

Richard E Korf, Michael Reid, and Stefan Edelkamp. Time complexity of iterative-deepening-a*. *Artificial Intelligence*, 129(1):199–218, 2001.

N. J.; Hart, P. E.; Nilsson and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics SSC*, 4(2):100–107, 1968.

Levi HS Lelis, Sandra Zilles, and Robert C Holte. Predicting the size of ida* search tree. *Artificial Intelligence*, 196:53–76, 2013.

Uzi Zahavi, Ariel Felner, Neil Burch, and Robert C Holte. Predicting the performance of ida* using conditional distributions. *Journal of Artificial Intelligence Research*, 37(1):41–84, 2010.

Donald E. Knuth. Estimating the efficiency of backtrack programs. *Mathematic of Computation*, 29(129):121–136, 1975.

P.-C. Chen. Heuristic sampling: A method for predicting the performance of tree searching programs. *SIAM Journal on Computing*, (21):295–315, 1992.