The Pruned N-Best Heuristic Search Algorithm

Jorge Pais
Instituto Superior de Engenharia de Lisboa
Instituto de Sistemas e Robótica
Instituto Superior Técnico
Lisbon, Portugal

and

Carlos Pinto-Ferreira
Instituto de Sistemas e Robótica
Instituto Superior Técnico
Lisbon, Portugal

ABSTRACT

This paper presents a semi-optimal heuristic search algorithm called *Pruned N-Best*, which has both the search behavior of the N-Best heuristic search algorithm and the minimization of backtracking search to applying a pruning technique over certain search paths. This minimization increases the Pruned N-Best algorithm efficiency when compared with others semi-optimal algorithms.

In domains where the number of operators grows up exponentially with the problem dimension the Pruned N-Best algorithm have been applied successfully (e.g. qualitative spatial reasoning area. Spatial reasoning planners usually tackle with problem models, where the interaction among elements is higher and change should be modeled not only taking the alteration of topological properties but also the creation, elimination and mutation of elements in the domain. As consequence, planners based on the search paradigm need special approaches to find out practical solutions. Even the N-Best search algorithm perhaps is inadequate to solve some practical problems specially those where the heuristic function is monotonic increasing (the condition for which the N-Best degenerates in the WA*).

The Pruned N-Best search algorithm here presented does not share the previous critical characteristics and it is complete under certain search conditions and it is effective in a wide class of problems.

Keywords
Search, Heuristic Search, Computing Techniques.

1. INTRODUCTION

Problem solving based on the search paradigm is always applicable in domains where the corresponding models can be defined by structured representations - *states* - and also when it is possible to identify all entities that can transform a state into another state - *operators*. The search process intends to finding out a solution path that is defined as the sequence of operators that makes possible to go from an initial state to a goal state. In a given domain the dimensional complexity of the search space is b^d in which b defines the number of operators (usually called theoretical branching factor) and d is the solution depth.

Reasoning processes based on a search algorithm can usually solve games (e.g., Maze), routing problems (e.g., TSP) or other A.I. problems since that they can be represented in terms of states and operators. However, when we try to modeling and solving certain real-world problems, where the number of operators grows up exponentially with the number of elements in the domain (e.g., qualitative spatial reasoning area [4]), special search approaches are needed to find out practical solutions. As though, this kind of problems and mainly their search space dimension raise practical difficulties, which are incomparable bigger than the NP-complete problems that usually appear in A.I. bibliography (e.g. chess problem). It means that when this kind of problems are to be solved (find out a semi-optimal solution) by a search algorithm, we do not find a search algorithm (RTA*[2], WA*[3]) with enough adequacy to solve them (in terms of space and time). To cope with this search inadequacy in practical terms, we have developed a search algorithm called N-Best but its behavior still depends hard from the heuristic function characteristic, as presented in [1]. As the N-Best search strategy is effective, we make some changes in the N-Best algorithm and generating thus the Pruned N-Best algorithm that in essential it remains the N-Best search behavior and prevails it over the characteristic of the adopted heuristic function.

The remainder of this paper describes the Pruned N-Best algorithm, its properties and some applications where it has been tested. In the next section, the heuristic search terminology is introduced. This is followed by a description of the Pruned N-Best algorithm and also an example of its behavior is depicted for different values of N. In section 4, the completeness of the algorithm is discussed. Section 5 presents a performance analysis with respect to the Pruned N-Best algorithm and others semi-optimal heuristic search algorithms in solving N-puzzle, Maze and TSP problems. Finally, the conclusions will be presented.

2. Terminology of Heuristic Search

Problem solving using a heuristic search process could be defined as a quadruplet (s_c, G, Γ, S) where s_c is the current best state, G is the set of goal states, Γ is the set of operators $\{\Gamma_0, \ldots, \Gamma_n\}$ and S is the generated state space. At the beginning of the search process s_c is usually called the initial state s_i and this process usually terminates when s_c is recognized as one of the goal nodes ($s_c \in G$). The ith successor state s_{s_i} is generated by applying the operator $\Gamma_i \in \Gamma$ to the current state s_c.

Any state s_n is said completely expanded when its all successor states are memorized and this state is kept in a list of all completely expanded states named *Closed*. A state s_k is said partially expanded whenever it does not have any memorized successor or a part of its successor states are kept, all of these states are included in a list of states called *Open*.

A path P is a sequence of states (s_1, …, s_n) generated by a successive operator application and when a path has the initial state as its first state and has the goal state as its last state then it is named a solution path.

Heuristic search algorithms usually utilize one or more heuristic functions along the search process. A heuristic function $\hat{f}(s)$ gives an estimated cost on the distance between the current state and a goal state $\hat{h}(s)$ plus the minimal cost path from the initial state to the current state $\hat{g}(s)$. A heuristic function is said to be admissible whenever it does not overestimate the real distance $f(s)$.

3. The Pruned N-Best Algorithm

The Pruned N-Best algorithm behavior is based on a climber behavior when it climbs an unknown hill – the next site to go is the best site selected among the first N better sites than the current one, all of them seeming accessible and closer to the top, any other site is not memorized by the climber (out of its view). This behavior can be rewritten in function of the accessibility condition, as: 1. Accessibility – the next state to follow is the best successor state with the best heuristic value among pre-defined N successor states, all of them with better heuristic values than the current state, 2. Inaccessibility – the absolute best state to follow is probably different from the selected state because this approach only memorizes the N successor states that are necessary to respect the accessibility point.

In real-world problems, such as certain chemical processes and physical domains, when a high number of interacting elements is modeled, the complexity of problems grow up exponential with the number of elements considered in the domain for each step level. This means that in these domains the problems usually have a number of operators that growing exponentially with the step level. This characteristic turns the process of complete successor generation computationally inadequate to find out a solution path in terms of time and memory.

The development of the Pruned N-Best algorithm (see figure 1) could be seen as an evolution of the N-Best algorithm [1] in terms of minimizing the successor generation needed to find out a solution path. This is to cope with high branching factors and solution depths that are always found in this kind of problems.

```
BestSuccessors(N, S, Γ)
{ i= 0, List=∅;
  While (i < N ∧ Γ≠∅)
  { S_s= Successor(S, Γ);
     If ( $\hat{f}(S_s) \le \hat{f}(S)$ ) ++i;
     add(List, S_s);
     Γ= NextOperator(Γ);
  }
  return (List, Γ);
}
```

```
PrunedNBest()
1. Add(Open, (s_i, Γ_0));
2. If (Open=∅) return failure;
3. (s_c, Γ_c)= GetStateWithMinF(Open);
4. (ListSuccessors, Γ_c)= BestSuccessors(N, s_c, Γ_c);
5. If (ListSuccessors=∅) add(Closed, s_c), goto 2;
6. If (ListSuccessors ∩ G ≠ ∅)  return success;
7. If (ListSuccessors ∈ (Open∪Closed) ∧ Γ_c≠∅) goto 4;
8. add(Open, ListSuccessors);
9. add(Closed, s_c);
10. goto 2.
```

Figure 1: The Pruned N-Best heuristic search algorithm.

Figure 1 is divided in two parts to make easier the understanding of the Pruned N-Best search algorithm. In the lower side is described the Pruned N-Best algorithm and in the upper side is described an auxiliary function named *BestSuccessors*. The *BestSuccessors* function has three parameters: N gives the number of successor states to be generated that have heuristic values better than the current best one; S represents the current best state to be expanded; and Γ is the first operator to be applied upon S. This function returns the list of all successor states generated until it finds out the value N (represented by *List*) and the first of the remaining operators that were not applied upon S (represented by the symbol Γ).

The Pruned N-Best behavior can be illustrated by an example. For instance, figure 2 depicts a tree that represents the complete state space where numbers correspond to $\hat{g} + \hat{h} = \hat{f}$ values, circles represent non-goal states and circles within squares are goal states.

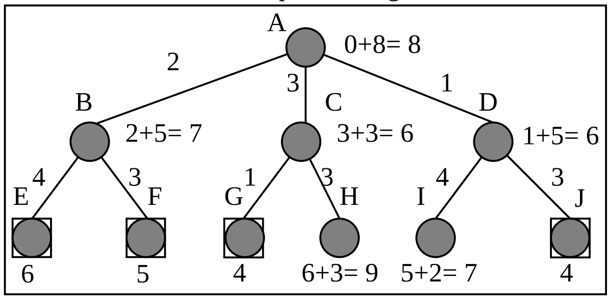

Figure 2: A complete state space

Figure 3 depicts the search strategy of the Pruned N-Best algorithm with N=1 and N= 2. Where slightly contoured circles represent *open states*. Boldly contoured circles represent *the current best state.* Black circles represent *closed states*. Slight straight lines represent paths. Bold straight lines represent solution paths. And, numbers are the corresponding $\hat{f}$ values.

The analysis of figure 3 allows extracting some characteristics of the Pruned N-Best algorithm. In each step of the generation process this algorithm memorizes all states until it finds out N states better than the current state. The best state is the state with the best heuristic value among all memorized states. States are expanded in depth whenever their $\hat{f}$ values are lowering than the current state value, especially in Pruned First-Best with N=1. And, states are expanded in breath whenever the successor states have worst $\hat{f}$ values than the current state.

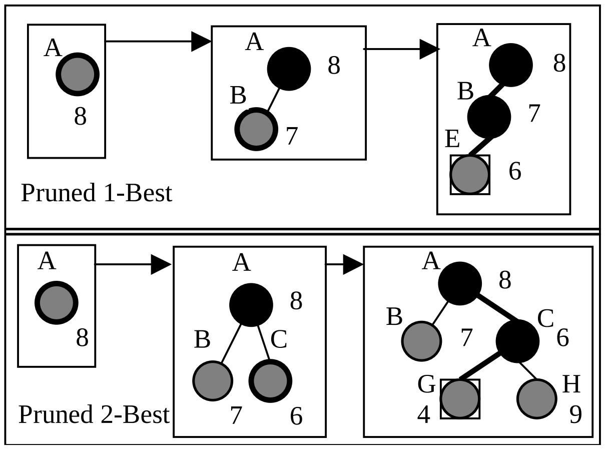

Figure 3: The Pruned N-Best algorithm behavior for N=1 and N=2

4. Completeness of the Pruned N-Best Algorithm

A search algorithm is said to be complete when there is at least one path from an initial state to a goal state into a finite graph state and the search algorithm is able to find out a path solution if there exists one. The Pruned N-best algorithm only respects the completeness condition if it keeps in the Open list at least a state belonging to the solution path. To guarantee this condition in any pruning search algorithm like ours, the problem to solve must have all states belonging to a solution path (e.g. TSP). If the problem respects the previous condition, this algorithm terminates without falling into an infinite loop because it does not expand an identical state more than once. However, if a state shares both the Open list and the solution path, the Open list never can be empty and the Pruned N-Best algorithm never ends in failure. Hence, the Pruned N-Best algorithm is complete whether all paths are solution paths in a finite state graph.

5. Performance Analysis

The performance and some characteristics of the Pruned N-Best algorithm are evaluated in solving TSP, Maze and N-puzzle problems.

The Traveling Salesman Problem(TSP) - a complete connected graph with 16 cities is randomly generated in a two-dimensional map with coordinates between (0, 0) and (599, 399), the distance between two cities is equal to the dimensional straight line in pixels between their locations. The search space initiates always in city 0 and the solution path includes all cities without repetitions and the return to the start city. The heuristic function utilized to solve this problem is the Minimum Spanning Tree (MST) admissible function to measuring the distance from the current state to the goal state plus the real distance from the initial state to the current state.

Each Maze problem is a two dimensional 600x400 map where the start site is located at (0, 0) and the goal site has the coordinate (599x399) and the obstacles are located randomly at a ratio of 40%. The change is performed using four operators Right, Left, Up and Down and each one costs 1. The heuristic function is the Manhattan distance to measuring the distance from the current position to the final position plus the number of operators applied to go from the initial position to the current position.

For the N-puzzle a 5x5 board with 24 numbered tiles (24-puzzle) was utilized. The *initial configuration* of the board must guarantees a path solution and it is randomly generated. The *final configuration* is a fixed and ordered configuration. The change of tiles is done by four operators *Right*, *Left*, *Up* and *Down* and each one costs 1. The heuristic function of a state is the number of operators applied to go from the initial state to the current state plus both the Manhattan distance and the Rotating sequence function (this function counts the number of tiles that have no correct sequence tile rotating over the board equidistant to the center plus 1 whether the center tile is not correct) to measuring the distance from the current state to the final state. The Rotating sequence function makes this heuristic function not admissible but it is much more informative.

The search algorithms used in this performance analysis are the RTA*[3], the WA*[4], the N-Best algorithm(with N=1) and the Pruned N-Best algorithm (see figure 1) with different values of N. The WA* algorithm uses a weighted version of $\hat{f}(s)$, namely $\hat{f}(s)=(1-W)\hat{g}(s)+W\hat{h}(s)$ for $0 \leq W \leq 1$, and in this test the value of W is fixed at 0,5. All search algorithms use two lists called Open and Closed. Completely expanded states are stored in the Closed list otherwise the states are kept in the Open list.

We have prepared 100 randomly problems of TSP, Maze and 24-puzzle and the obtained results are in table 1 for TSP, table 2 for Maze and table 3 for 24-puzzle. The column *Success Rate* is the percentage of success without abortion due to main memory limit. The column *Solution Cost* gives the average and the standard deviation in respect of the cost of solution paths for all successful searches. The column *Search Steps* shows the number of times that each algorithm repeats its looping steps in terms of the average and standard deviation of successful searches. The column *Memorized States* gives the total number of states kept in Open and Closed lists. And, the column *Run Time* shows the average and the standard deviation values of time in seconds that the successful searches delay. These tests have been done in a personal computer with both a Pentium III-450MHz processor and main memory able to memorize 600000 states without using virtual memory.

		Solution Cost		Search Steps		Memorized States		Run Time	
Algorithm	Success Rate	Average	Standard Deviation	Average	Standard Deviation	Average	Standard Deviation	Average	Standard Deviation
RTA*	100	5795	397	19	0	20	0	0.1	0
WA*	62	2369	167	31740	5647	254348	49919	43	9
1-Best	64	2350	233	39287	12105	201038	66103	34	12
Pruned 1-Best	100	2432	172	16125	10623	91919	62237	15	11
Pruned 2-Best	77	2300	125	26124	8308	197308	64296	33	11

Table 1: Performance of Search algorithms in TSP with 18 cities.

		Solution Cost		Search Steps		Memorized States		Run Time	
Algorithm	Success Rate	Average	Standard Deviation	Average	Standard Deviation	Average	Standard Deviation	Average	Standard Deviation
RTA*	0								
WA*	100	1813	426	63857	21537	64029	21528	8	3
1-Best	100	1930	322	187867	33422	74841	13238	13	3
Pruned 1-Best	0								
Pruned 2-Best	100	1930	322	74687	13266	74844	13258	12	5

Table 2: Performance of Search algorithms in Maze 600x400.

		Solution Cost		Search Steps		Memorized States		Run Time	
Algorithm	Success Rate	Average	Standard Deviation	Average	Standard Deviation	Average	Standard Deviation	Average	Standard Deviation
RTA*	0								
WA*	42	170	13	124722	67232	293795	77837	73	41
1-Best	46	174	6	237513	100086	223703	88538	156	83
Pruned 1-Best	27	181	7	189651	107972	320278	185639	171	111
Pruned 2-Best	63	169	10	144687	51886	259027	88008	236	120

Table 3: Performance of Search algorithms in 24-puzzle.

The results presented in tables 1, 2 and 3 demonstrate both the effectiveness and the computational adequacy of the Pruned N-Best search algorithm to solving problems in domains where it is complete (particularly with N=1, e.g. TSP). For the others problems, Maze and 24-puzzle, the Pruned N-Best Algorithm shows again its effectiveness when the selected value of N is adequate to the characteristics of the problem.

The main drawback of this algorithm is that it is neither optimal nor complete (for all problem conditions) as it proceeds in a greedy fashion. Notwithstanding, the search paradigm can be optimal in solving simpler NP-complete problems but in solving hard NP-complete problems the search paradigm must be semi-optimal (and why not semi-complete) to ensure computational adequacy.

6. Conclusion

This paper introduces the Pruned N-Best semi-optimal heuristic search algorithm, which allows to dealing with hard NP-problems, limited computational resources and providing practical and useful time solutions. An evaluation of its performance in classical problems like N-puzzle, Maze and TSP problems is done, such as, a performance comparison with WA*, RTA* and N-Best semi-optimal algorithms.

The Pruned N-Best algorithm is a generic semi-optimal search algorithm that can be applied to other kinds of problems. In this paper, we do not discuss both the optimal value of N and the adequacy of this algorithm to be used with learning techniques that varies the value of N during the run time search to optimize solutions in terms of memory and time. But, in our opinion these two aspects are very important to be studied in further research.

References

[1] Pais J., and Pinto-Ferreira C. (1999), "The N-Best Heuristic Search Algorithm," in: Proceedings of the 18th Workshop of the UK Planning and Scheduling Special Interest Group PLANSIG99, England.

[2] Korf, R.E. (1990), "Real-Time Heuristic Search" Artificial Intelligence, 62(1): 41-78.

[3] Pearl, S. (1984), *Heuristics: Intelligent Search Strategies for Computer Problem Solving*, Addison-Wesley.

[4] Cohn A. (1995), The Challenge of Qualitative Spatial Reasoning, Division of Artificial Intelligence, School of Computer Studies, University of Leeds, England.

[5] Bisiani R. (1992):”Beam Search,” in Encyclopedia of Artificial Intelligence, 1467-1468, New York: Wiley-Interscience Publication.

[6] Harvey W., and Ginsberg M. (1995):”Limited discrepancy search,” in: Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, 607-613.

[7] Russel S., (1992):”Efficient memory-bounded search method," in: Proceedings of the Tenth European Conference on Artificial Intelligence, 1-5.

[8] Pais J., Pinto-Ferreira C.(1998), "Search Strategies for Reasoning about Spatial Ontologies," in: 10th IEEE International Conference On Tools with Artificial Intelligence, Taiwanese Association for Artificial Intelligence (ICTAI98), Taiwan.