Partial-Expansion A* with Selective Node Generation

Ariel Felner
Meir Goldenberg
Guni Sharon
Roni Stern, Tal Beja
ISE Department
Ben-Gurion University (Israel)
felner@bgu.ac.il
{mgoldenbe,gunisharon,roni.stern}@gmail.com

Nathan Sturtevant
CS Department
University of Denver (USA)
Sturtevant@cs.du.edu

Jonathan Schaeffer
Robert C. Holte
CS Department
University of Alberta (Canada)
{Jonathan,holte}@cs.ualberta.ca

Abstract

A* is often described as being 'optimal', in that it expands the minimum number of unique nodes. But, A* may generate many extra nodes which are never expanded. This is a performance loss, especially when the branching factor is large. *Partial Expansion A** (PEA*) (Yoshizumi, Miura, and Ishida 2000) addresses this problem when expanding a node, n, by generating *all* the children of n but only storing children with the same f-cost as n. We introduce an enhanced version of PEA* (EPEA*). Given *a priori* domain knowledge, EPEA* only generates the children with the same f-cost as the parent. State-of-the-art results were obtained for a number of domains. Drawbacks of EPEA* are also discussed. A full version of this paper appears in the proceedings of AAAI-2012 (Felner et al. 2012).

A*

It is well known that A* *expands* the minimum number of unique nodes. But A* also *generates* many nodes that it doesn't expand. Let X be the number of nodes that A* expands and let b be the average branching factor. Every time a node is expanded, b children are inserted into the OPEN list. Therefore, a total of $b \times X$ nodes are generated. However, once a solution of cost C is found, the algorithm only needs to verify that no solution with cost $< C$ exists. Therefore, when the minimal-cost node in OPEN has $f = C$, the algorithm halts and all other nodes in OPEN (most of them have $f > C$) are discarded. Nodes with $f > C$ are designated as being *surplus*. The number of surplus nodes in OPEN can grow exponentially in the size of the domain, resulting in significant costs.

Partial Expansion A*

*Partial Expansion A** (PEA*) (Yoshizumi, Miura, and Ishida 2000) never adds surplus nodes (with $f > C$) to OPEN (Yoshizumi, Miura, and Ishida 2000). When expanding node n, PEA* first generates a list of *all* the children of n, $CH(n)$. Only nodes c from $CH(n)$ with $f(c) = f(n)$ are added to OPEN. The remaining children are discarded but n is added back to OPEN with the smallest f-value greater than $f(n)$ among the remaining children. We refer to this algorithm as *Basic PEA** (BPEA*). The memory benefit of BPEA* is straightforward. Assume the goal node has $f = C$. *All* nodes with $f > C$ that were generated (as children of expanded nodes with $f \leq C$) will not be added to OPEN. However, BPEA* incurs extra time overhead when it repeatedly generates *all* the children of a node n every time n is expanded. Thus, while memory is always saved, a time tradeoff exist.

We borrow the terminology first used in RBFS. Denote the regular f-value ($g + h$) of node n as its *static value*, which we denote by $f(n)$ (small f). The value stored in OPEN for n is called the *stored value* of n, which we denote by $F(n)$ (capital F). Initially $F(n) = f(n)$. After n is expanded for the first time, $F(n)$ might be set to $v > f(n)$ when its minimal remaining child has a static f-value of v.

Enhanced PEA*

We introduce *Enhanced* PEA* (EPEA*). Assume we are now expanding a node n with $f(n) = K$. While BPEA* generates *all* the children of n, EPEA* uses a mechanism which *only* generates the children with $f = K$, without generating and discarding the children with values $f < K$ or $f > K$. Thus, each node is generated only once throughout the search process and no child is regenerated when its parent is re-expanded.

This is achieved with the following idea. In many domains, one can classify the operators applicable to a node n based on the change to the f-value (denoted Δf) of the children of n that they generate. The idea is to use this classification and only apply the operators of the relevant class. EPEA* creates a domain-dependent *Operator Selection Function* (OSF) which receives a state p and a value v. The OSF has two outputs: **(1)** a list of operators that, when applied to state p, will have $\Delta f = v$. **(2)** v_{next} — the value of the next Δf in the set of applicable operators. Now assume node n is expanded with a stored value $F(n)$. $F(n)$ might be larger than the static value of $f(n) = g(n) + h(n)$ in cases where n was already expanded in the past. In such cases, $F(n)$ was inherited from one of its children. We only want to generate a child c for which $f(c) = F(n)$. Thus, we only need the operators which will increase $f(n)$ by $\Delta f = F(n) - f(n)$. $OSF(n, \Delta f)$ is used to identify the list of relevant operators. Node n is re-inserted into OPEN with the next possible value for this node, i.e.,

$F(n) = f(n) + v_{next}(n, \Delta f)$. If no larger value is possible ($v_{next} = nil$) then node n is moved to the CLOSED list. Of course, if the goal node is found before n is moved to CLOSED, EPEA* never generates any of the surplus nodes with an f-value larger than the f-value of the goal.

In the full paper (Felner et al. 2012), we give deeper theoretical analysis by comparing the different operations needed by the different algorithms. In addition, we provide a number of methods to create OSFs.

Enhanced partial expansion IDA* (EPE-IDA*)

Partial expansion and OSF work naturally in IDA*. In fact, IDA* can be viewed as using basic partial expansion. Assume that the IDA* threshold is T. Once a node n is expanded, all the children are generated. Children with $f \leq T$ are expanded, while children with $f > T$ are generated and discarded. This satisfies the basic partial expansion criteria. However, augmenting IDA* with an OSF (enhanced partial expansion) may significantly reduce the number of node generations. This is done with EPE-IDA*.

EPE-IDA* is simpler than EPEA*. EPE-IDA* associates each node only with its static value. There is only one "stored" value for the entire tree — the value for the next iteration. Given an expanded node n, let $d = T - f(n)$. The OSF will identify the applicable operators with $\Delta f \leq d$. These operators will be applied. Operators with $\Delta f > d$ need not be applied as the children they produce will have f-value $> T$. Thus, for every iteration, EPE-IDA* generates exactly the nodes that will be expanded. The next IDA* threshold is set to the minimal $v_{next}()$ among the expanded nodes.

Experiments: Rubik's cube

OSFs can be built on top of PDB heuristics. The resulting PDB is called Δ-PDB. The entry Δ-PDB$[a, \sigma]$ indicates how much the heuristic value $h(p)$ will change when an operator σ is applied to any state p such that $\phi(p)$=a where $\phi(p)$ gives the relevant entry in the PDB for state p. This technique was applied to Rubik's Cube PDB based on the corner cubies (Korf 1997). This abstraction has $88,179,840$ abstract states. In the Δ-PDB, each of these states further included an array of size 18, one index per operator (each requiring 2 bits) for a total of 396,809,280 bytes of memory (396 MB, compared to 42 MB for the original PDB). Note that the Δ-PDB can be potentially compressed to reduce its memory needs.

Results using the corner Δ-PDB are given in Table 1 (top). Each line is the average over 100 instances of depth 13-15. The reduction (*ratio* column) in the number of nodes generated is a factor of 13.3 (the known effective branching factor) and the time improvement is only 3.7-fold. The reason for the discrepancy is that the constant time per node of EPE-IDA* is larger than that of IDA* since it includes the time to retrieve values from the Δ-PDB.

Experiments: Pancake puzzle (GAP heuristic)

In the pancake puzzle a state is a permutation of the values $1...N$. Each state has $N-1$ children, with the k^{th} successor formed by reversing the order of the first $k+1$ elements of the permutation ($1 \leq k < N$). The experimental results for 100 random instances for 10 to 70 pancakes are given in Table 1 (bottom). The heuristic used is the most effective heuristic on this puzzle, known as the GAP heuristic (Helmert 2010). The GAP heuristic iterates through the state and counts the number of neighboring pancakes that are not consecutive in their numbers. This heuristic is admissible. For 70 pancakes, EPE-IDA* generated 68 times fewer nodes than IDA*. Most of this is reflected in the running time (54-fold). To the best of our knowledge, these are the state-of-the-art results for this puzzle.

#	IDA*	EPE-IDA	ratio	IDA*	EIDA*	ratio
Rubik's Cube (Corner PDB)						
	Generated Nodes - Thousands			Time (mm:ss)		
13	434,671	32,610	13.32	0:53	0:15	3.53
14	3,170,960	237,343	13.37	5:31	1:32	3.68
15	100,813,966	7,579,073	13.30	175:25	47:16	3.71
Pancake Puzzle (GAP Heuristic)						
	Generated Nodes			Time (ms)		
20	18,592	1,042	17.84	1.5	0.1	11.23
30	241,947	8,655	27.95	24.9	1.2	20.00
40	1,928,771	50,777	37.98	247	8.5	30.75
50	13,671,072	284,838	47.99	2,058	57	36.15
60	92,816,534	1,600,315	57.99	16,268	359	45.32
70	754,845,658	11,101,091	67.99	155,037	2,821	54.90

Table 1: Rubik's Cube and pancake puzzle results

Other domains

In the full paper (Felner et al. 2012), we provide additional experimental results for various domains including the 15 puzzle and multi-agent path finding. These results confirm the advantage of EPEA* over BPEA* and A*.

However, we also show that EPEA* has limitations and in some cases it is not beneficial to use it. In particular, in polynomial domains with small branching and many cycles, there is the danger of inflating the OPEN list, thus reducing the potential for performance gains despite fewer distinct nodes being generated. Experimental evidence for this is shown for grid-based navigation.

References

Felner, A.; Goldenberg, M.; Stern, R.; Sharon, G.; Beja, T.; Holte, R.; Schaeffer, J.; Sturtevant, N.; and Zhang, Z. 2012. Partial-expansion a* with selective node generation. *Proc. AAAI, To appear.*

Helmert, M. 2010. Landmark heuristics for the pancake problem. In *SOCS*, 109–110.

Korf, R. E. 1997. Finding optimal solutions to Rubik's Cube using pattern databases. In *AAAI*, 700–705.

Yoshizumi, T.; Miura, T.; and Ishida, T. 2000. A* with partial expansion for large branching factor problems. In *AAAI*, 923–929.